Tracing Unsolvability: A Mathematical, Historical and Philosophical analysis with a special focus on Tag Systems.

Liesbeth De Mol

Thank You

Writing down the results of about three years of research has not only been an intellectual but, maybe even more, an emotional struggle, that resulted in what is now lying before the eyes of the reader. I would never have been able to do the research I have been doing without the existence of several people. I would like to thank all these people here.

First of all, I want to express my respect and gratitude towards my supervisor prof. dr. E. Weber. I could not have wished a better supervisor. You believed in me, and offered your help when necessary. Equally important, is the fact that this research would literally not have been possible were it not for the fact that you offered me a way out at a given time. The most valuable thing for me is freedom of thinking, and I almost lost this freedom and would thus haven given up on this research, were it not for you.

Anthony, I will not spend many words describing why I mention you here, but I think generosity was and still is the best way to describe what it is all about. Thank you, just for being who you are.

My mother and father have been the most wonderful parents for me. Your support to me has been invaluable. But it is not only your support. More important for me has been the way you have made me the person I am now. I could not have wished for better parents! Steven, I know we didn't make much time for each other. Still, you have done your best to motivate me in your own typical ways. I am looking forward to the dinner I owe you.

Tom, we share a rather strange history with each other. Although we have not spent lots of time with each other the last few years, both working like hell, most of the moments we were able to talk were, at least for me, very important: talking, laughing, babbling! I so much enjoy our conversations.

Julian, Renate, Michael, Alberto and all the others who were there: for me *Mathematik für Kunstler* has been such an important event during my research. I think it only rarely happens that a significantly large group of people with equally many backgrounds, are able to have so many rather intensive discussions. For me, Hamburg was magic.

Lut, we know each other for so long now. The way you are capable of approaching scientific research in your own creative ways is very inspiring for me! I am looking forward to our next beer at the "Geitenboer"!

I am very much indebted to the people at the *Center for logic and philosophy of science*. Prof. dr. D. Batens and Prof. dr. J. Meheus, together with Prof. dr. E. Weber, I want to thank all three of you for the way you have welcomed and accepted me in the research group. Leen and Hans, you supported me when I had my crash-down during the last weeks of finishing this dissertation. Thank you for that! Peter, thank you for having taken the time to discuss with me some matters related to tag systems. Stephan, I think you are actually a great philosopher and our discussions have been valuable to me. Tim and Nel, you helped me at a time I really needed help. I am still very grateful for your support during that time.

Daniel, I so much enjoy our conversations late in the night, with a bottle of wine and lots of cigarettes. Thank you for your support. May the Nietzscheans learn some realism from your dissertation!

Prof. dr. M. Davis, I don't think you have been aware of how important a role you have played for me during the last 1.5 years of my research. After my first encounters with you, I better understood that one should be extremely careful in approaching certain parts of mathematics from a more philosophical point of view. The comments you have given me, showed me that I was actually too careless about certain things, and in having abstracted from these comments they have served as a kind of advise I often thought about in writing the dissertation.

Prof. dr. K. Kirby, I want to thank you here for our delay-discussions on the facing the computer topic. I am still glad I pushed the button that one evening and am very much looking forward to our future correspondences.

I also would like to thank the five readers of this dissertation, Prof. dr. D. Batens, Prof. dr. M. Davis, Prof. dr. M. Margenstern, Prof. dr. J.P. van Bendegem and Prof. dr. E. Weber, for being prepared to read this “behemoth” of a dissertation. I really do hope you’ll enjoy it in some way.

I know some people will laugh about it, but I think my cats deserve mention here. They are fat and lazy, but they managed to distract me for now and then from my research, in a way no human could.
Maarten, to sum up all the reasons why you are mentioned here, would ask for a rather lengthy text. Thank you for all the things that would be on this list. All I can say is that this thing between you and me surpasses any bond I could think of before meeting you, even that with tag systems (hihihi).

Contents

Thank You **iii**

1 Introduction **3**

1.1 Undecidability everywhere? 3

1.2 And now for something completely different? 4

1.2.1 Computability and “Computability” 5

1.2.2 Diagonalization and Reducibility 7

1.3 Questioning unsolvability. 8

1.4 General Outline and research Questions 10

1.4.1 Short Description of the chapters. 11

1.5 A small note to the reader. 18

I Re-Tracing **21**

2 The Beginnings **25**

2.1 General Background 26

2.2 From solvability to unsolvability 35

2.2.1 Introduction 35

2.2.2 Focus on Form 41

2.2.3 The Problem of “Tag”. 47

2.2.4 Further reductions: From tag systems to Post’s thesis 53

2.2.5 Conclusion 64

2.3 “To deny what seems intuitively natural” 67

2.3.1 Introduction 67

2.3.2 Towards variant systems of logic. 70
2.3.3 An Inconsistent Set of Postulates 74
2.3.4 λ - The Ultimate Operator 78
2.4 From typewriters to universal computing machines 95
2.4.1 Introduction 95
2.4.2 Typewriters and "Little wonders" 96
2.4.3 The central limit theorem 97
2.4.4 Newmann's course on the foundations of mathematics . . 100
2.5 Conclusion 100

3 1936 103

3.1 Different questions, different answers. 105
3.1.1 "An Unsolvable Problem of Elementary Number Theory" . 105
3.1.2 "On computable numbers, with an application to the Entscheidungsproblem" 112
3.1.3 "Finite Combinatory processes. formulation 1" 126
3.2 On the status of the identification 138
3.2.1 Church's reviews of Post's and Turing's paper 139
3.2.2 On the adequacy of Turing's identification: From definition to theorem. 142
3.2.3 The identification as (hypo)thesis or law. 147
3.2.4 Some further developments. 159
3.3 Strategies against intuition. 163
3.3.1 The thesis as a definition. On the significance of using the right formalism for the development of the theory. 167
3.3.2 Different intuitions, different formalisms 169
3.3.3 "Testing in practice": Studying formalisms instead of intuitions 170
3.3.4 Can we trust our intuition? 171

4 The computer. 175

4.1 The first computers 178
4.1.1 The ENIAC and the EDVAC 180

4.1.2 Alan Turing's work on computers and programming 199
4.1.3 Zuse's Z1, Z2, Z3, Z4 and Plankalkül 208
4.1.4 Conclusion. 219
4.2 Exploring the "universe of discourse" 222
4.2.1 von Neumann and theoretical physics. 225
4.2.2 Lehmer's computational work on number theory. 231
4.2.3 Conclusion 237
4.3 Going beyond or not beyond the Turing limit? 240
4.3.1 On the practical feasibility of the computable: Computational Complexity Theory. 240
4.3.2 The land of Tor'bled-nam. To solve the unsolvable. 246
4.3.3 Conclusion 256
4.4 Conclusion. 260

II Tagging 263

5 Why Tag systems? 267

6 Preliminaries 273
6.1 Discussion of published results on tag systems. 274
6.1.1 General Theoretical results 274
6.1.2 "Tag – you are it": Some concrete research on tag systems. . 292
6.1.3 Conclusion 304
6.2 General classes of behaviour in tag systems 305
6.2.1 Description of classes of behaviour 305
6.2.2 "Unpredictable iterations." 308
6.2.3 Classes of behaviour and the two forms of the problem of "tag". 310
6.2.4 Conclusion 312
6.3 Shifting through tags 313
6.3.1 An example of a solvable tag system. 313
6.3.2 Generalization of the example. 319

7 Constraints for intractable behaviour 323

7.1 Introduction 323

7.2 Notational Conventions 326

7.3 Description of the Constraints 326

7.3.1 Constraint 1: Post's condition 327

7.3.2 Constraint 2: The Wang condition 327

7.3.3 Constraint 3. Proportions between $\#a_i$. 328

7.3.4 Constraint 4. The table method. 333

7.3.5 Constraint 5. On the number of iterations. 338

7.4 Generating intractable tag systems 342

7.4.1 Algorithm 1: Two-symbolic tag systems, $v - l_{w_0} = l_{w_1} - v$ . . 342

7.4.2 Algorithm 2: Two-symbolic tag systems, $v - l_{w_0} \neq l_{w_1} - v$ . . 347

8 Playing with Tag Systems 353

8.1 Purpose of the chapter 355

8.2 Some further restrictions 356

8.2.1 On the programming language used. 356

8.2.2 Size of Sample space vs. computation time 357

8.2.3 Focus on 2-symbolic tag systems 358

8.3 Experiment 1 361

8.3.1 Set-up of experiment 1 361

8.3.2 Discussion of the results 364

8.4 Experiment 2 376

8.4.1 Set-up of experiment 2 380

8.4.2 Discussion of the results 382

8.5 Experiments 3–6: Summary of the results. 404

8.5.1 Experiment 3 405

8.5.2 Experiment 4 405

8.5.3 Experiment 5 406

8.5.4 Experiment 6 406

8.6 Conclusion 407

9 Universality and Unsolvability in Tag Systems 409

9.1 Why are (small) universal machines interesting? 411

9.1.1 Size and definition of universal machines. 412

9.1.2 Small universal machines: an overview. 416

9.2 Why are (small) universal systems not interesting? 418

9.2.1 Minsky's 4-symbol, 7-state machine. 423

9.2.2 Generating a universal tag system. 434

9.2.3 Conclusion 449

9.3 Studying the "universe of discourse" 450

9.3.1 The Busy Beaver Game 451

9.3.2 Busy Beavers and Collatz-like Problems 463

9.3.3 On the "universality" of cellular automaton rule 110. 471

9.3.4 Conclusion 478

9.4 Solvability and Unsolvability in Tag systems 479

9.4.1 Tag systems and Collatz-like problems 480

9.4.2 Solvability of the class $\mu = \nu = 2$. 487

9.4.3 Universality in tag systems. 517

9.4.4 Discussion on the limits of solvability and unsolvability in tag systems 542

9.5 Conclusion. 547

10 Conclusion. Tracing Unsolvability 549

A. Algorithm 3 for generating Tag Systems: *N*-ary tag systems. 557

B. Plots from Experiment 1 559

C. Detailed Description of Four experiments on Tag Systems 587

> [...] the creativeness of human mathematics has a counterpart inescapable limitation thereof – witness the absolutely unsolvable (combinatory) problems. Indeed, with the bubble of symbolic logic as universal logical machine finally burst, a new future dawns for it as the indispensable means for revealing and developing those limitations. For [...] Symbolic Logic may be said to be Mathematics become self-conscious.

Emil L. Post, 1920–21.[1]

> Much of modern mathematics is being developed in terms of what can be proved by general methods rather than in terms os what really exists in the universe of discourse. Many a young Ph.D. student in mathematics has written his dissertation about a class of objects without ever having seen one of the objects at close range. There exists a distinct possibility that the new machines will be used in some cases to explore the terrain that has been staked out so freely and that something worth proving will be discovered in the rapidly expanding universe of mathematics.

Derrick H. Lehmer, 1951.[2]

> The entscheidungsproblem does have practical importance in addition to it's philosophical significance. Mathematical proof is a codification of more general human reasoning. An automatic theorem prover would have wide application within computer science, if it operated efficiently enough. Even though this is hopeless in general, there may be important special cases which are solvable. It would be nice if Church's or Turing's proofs gave us some information about where the easier cases might lie. Unfortunately, their arguments rest on "self-reference," a contrived phenomenon which never appears spontaneously. This does not tell us what makes the problem hard in interesting cases.

Michael Sipser, 1992, 1951.[3]

[1] From [Pos65], footnote 12, p. 343

[2] From [Leh51], p. 146

[3] From [Sip92], p. 603

Chapter 1

Introduction

1.1 Undecidability everywhere?

“es schneit” ist eine wahre Aussage dann und nur dann, wenn es schneit[1]

When does one state of a problem that it is undecidable? In everyday life one is undecided if one is not sure about something. You can doubt about the most divergent things, from what one will eat this evening to the more fundamental problems of life itself related to jobs, friends,... The reasons for not being able to make a decision and thus resolve the doubt can be very different. If the more fundamental decisions of life are involved, one of the main reasons for not being able to make such a decision is on the one hand a lack of all the relevant information, and on the other hand, the incapability of foreseeing all the possible consequences in all their details given a certain decision.

Doubt and the related problem of making decisions, is one of the leading motives in the history of Western philosophy, with one of the most famous texts being Descartes’ *Méditations Métaphysiques* [Des47]. Starting from the problem of doubt, one of his main conclusions is the fact that mathematics is the only branch of human knowledge which is undoubtable, containing truths so obvious in every circumstance you can think of (like the fact that 3 + 2 will always equal 5) that it cannot be the subject of uncertainty or mistakes, let alone

[1] [Tar35], p. 453

undecidability ([Des47], p. 38):

> C'est pourquoi peut-être que de là nous ne conclurons pas mal, si nous disons que la physique, l'astronomie, la médicine, et toutes les autres sciences qui dépendent de la considération des choses composées, sont fort douteuses et incertaines; mais que l'arithmé-tique, la géometrie, et les autres sciences de cette nature, qui ne traitent que des choses fort simples et fort générales, sans se mettre beaucoup en peine si elles sont dans la nature, ou si elles n'y sont pas, contiennent quelque chose de certain et d'indubitable. Car, soit que je veille ou que je dorme, deux et trois joints ensemble formeront toujours le nombre de cinq, et le carré n'aura jamais plus de quatre côtés; et il ne semble pas possible que des vérités si apparantes puissent être soupçonnées d'aucune fausseté ou d'incertitude.

Since Descartes wrote this beautiful text, mathematics has changed a lot. It is no longer absolutely true that $3+2=5$, depending as it does on the mathematical framework you are working in.[2] The grown understanding of a mathematical truth being defined relative to a certain framework however, is not the only reason for these words by Descartes to sound rather naive. About four centuries after the publication of this text it would be proven that there is undecidability at the very heart of mathematics – its foundations.
Here one of course does not speak of undecidability in terms of its everyday meaning since exact mathematical results are involved. Instead one uses *undecidable propositions* and *unsolvable decision problems*. Contrary to the everyday use of “doubt” and “undecidedness” these concepts have been defined formally and thus don't seem to allow for any doubt as far as their meaning is concerned.

1.2 And now for something completely different?

What do we mean *exactly* in stating that a certain mathematical system is undecidable? There are two possible answers: it can be the case that the formalism considered has an unsolvable decision problem or there exist undecidable

[2] When working with modulo arithmetic, 3 + 2 can e.g. become 1, with a modulus 4.

propositions for the formalism (it is incomplete). Here focus will not be put on undecidable propositions but on unsolvable decision problems.[3]

But what exactly is an unsolvable decision problem? Does it mean that for certain mathematical problems, there is no way to make certain decisions or to give a definite answer? In a sense yes, although one must be careful here: as was stated before, we are dealing with mathematics, so in order to make clear what is intended here, one must give precise and clear definitions of the concepts involved. This was exactly the problem mathematicians were facing in the late twenties and the early thirties.

In this section we will shortly look at the two pillars that made (and make) it possible to prove certain decision problems unsolvable. First of all, one needs a way to formally capture certain intuitive notions. Secondly, on acceptance of the formalization of these notions, one implements specific methods to actually prove a certain decision problem unsolvable.

1.2.1 Computability and "Computability"

In their *Grundzüge der theoretischen Logik*, published in 1928, Hilbert and Ackerman gave the classic statement of what is now known as the Entscheidungsproblem ([AH28], p. 73):

> The Entscheidungsproblem is solved if one knows a procedure which will permit one to decide, using a finite number of operations, on the validity, respectively the satisfiability of a given [first-order] logical expression.[4]

In 1936 Alonzo Church and Alan Turing [Chu36d, Chu36e, Tur37] independently of each other proved that there exists no "finite procedure" that decides

[3] I am indebted to Martin Davis for drawing my attention to the significance of explicitly differentiating between unsolvable decision problems and undecidable propositions. In the literature one often uses the term "undecidability", where it can both refer to undecidable propositions or unsolvable decision problems. Since this 'habit' can give rise to some confusion, it should be pointed out here that every time the word "undecidability" is used, the author is actually pointing at unsolvable decision problems unless stated otherwise.

[4] "*Das Entscheidungsproblem is gelöst, wenn man ein Verfahren kennt, das bei einem vorgelegten logischen Ausdruck durch endlich viele Operationen die Entscheidung über die Allgemeingülltigkeit bzw. Erfüllbarkeit erlaubt.*" Translation to English from [Gan88].

for any given formula in first-order predicate calculus whether it is deducible within this calculus. This result together with Gödel's completeness theorem [Göd30] implies the negative answer to the Entscheidungsproblem in its above formulation.[5] But what does one *exactly* mean with an "effective finite decision procedure"? What is meant if one states of a problem that it is "(un)solvable by finite means"?

In order to prove the Entscheidungsproblem unsolvable, one first had to find a mathematically satisfying answer to these questions. One needed formalisms that can be considered as proper formalizations of certain intuitive concepts. This was done by Church [Chu36c], Post [Pos36][6] and Turing [Tur37]. Each proposed their formal equivalent of *intuitive* notions such as effective calculability (due to Church), computability (due to Turing), generated set (due to Post) and solvability (due to Post) – formalisms which were shown to be equivalent to each other. The Entscheidungsproblem could now be proven unsolvable since the identification between the intuitive notion of a procedure solving a problem in a finite number of steps with certain mathematical forms was considered satisfactory.

The question posed at the beginning of this section can now be answered: an unsolvable decision problem is a general mathematical problem for which there exists no formalism, equivalent to those considered to be able to compute anything which is intuitively computable, that can be used to *effectively* solve each case of the problem in a finite number of steps – it is a non-computable problem. The unsolvability of decision problems like the Entscheidungsproblem however, is merely valid in as far as one accepts the identification between the intuitive notions and the respective formalisms, i.e., if one accepts "theses" such as those proposed by Church, Post and Turing. In the remainder of this

[5] While the classic formulation of the Entscheidungsproblem indeed refers to validity and satisfiability, these terms are normally not involved in the statement of other decision problems. A general form of decision problems is usually something like: *Does there exist a finite procedure to decide for an arbitrary x, whether y is yes/no the case for x within a certain formal system (e.g. to decide whether an arbitrary formula (x) in first-order predicate logic is yes/no provable (y)).*

[6] It should be noted however that Post did not prove the Entscheidungsproblem unsolvable (See Sec. 2.2.4).

text we will use the concept of computability to cover the cluster of intuitive notions captured by the several different formalisms. Still, the reader should always be aware that "computability" is merely one of the intuitive notions.

1.2.2 Diagonalization and Reducibility

To prove a certain decision problem unsolvable one needs more than a well-argued correspondence between an intuitive notion and a formalism. There are two basic methods used in this context, the first being necessary for the second to work.

The first proofs by Church, Post and Turing that show certain decision problems unsolvable (relative to their respective formalisms) all rely on some variant of a Cantorian diagonalization.[7] Basically, this method can be used to prove for certain infinite lists of letter or number sequences, that there exist sequences that are not part of the list.

After the first unsolvable decision problems had been proven to exist, the formalisms they are rooted in could in their turn be used to prove the unsolvability of (further removed) decision problems. This is done by *reducing* a known unsolvable decision problem (call it A) to the problem one wants to prove unsolvable (called B). This comes down to finding a method for translating A into B, i.e. one must show that any specific instance of A can be reduced to a specific instance of B. For example, as Turing showed, any Turing machine can be expressed in first-order predicate calculus. Based on the unsolvability of the *printing problem*, i.e., the problem to determine for any given Turing machine whether it will ever print a given symbol, he could then show that the Entscheidungsproblem is unsolvable.

Although diagonalization lies at the basis of the majority of unsolvable decision problems, reducing one problem to another has become the standard method.

[7] Diagonalization was first used by Georg Cantor to prove that there are, in a way, different kinds of infinite sequences – the infinite sequence of natural numbers being merely a first step up to the transfinite. While Cantor had already proven in 1874 that there are, in a certain well-defined way, more real numbers than there are natural numbers [Can74], he gave an alternative, shorter proof in 1891 [Can91], using diagonalization.

Nowadays hundreds of decision problems have been proven unsolvable in several branches of mathematics and computer science most of them being (directly or indirectly) rooted in the problems shown to be unsolvable by Church, Post and Turing.

1.3 Questioning unsolvability.

As was said, Post, Church and Turing were among the first to prove that there exist certain unsolvable decision problems. They first defined certain formalisms considered capable to capture the intuitive notion of a "computation", and then used some kind of method of diagonalization to complete the proof. From the first moments I read those proofs, I felt thrilled and dissatisfied at one and the same time.

On the one hand I loved almost every aspect of the constructions and methods leading to the proofs. On the other hand I could not get rid of the idea that the link between the general unsolvability of a whole class of systems and (the *actual execution* of a) *specific* instance from this general class, is not clear from the proofs, at least not on an intuitive level. Of course, there is such a link, since the fact that we are confronted with a class of systems for which their exists no algorithm to solve every instance of a certain decision problem for that class implies that there must exist specific instances for which we can in no way find e.g. a Turing machine that will decide its halting problem. Still, going from proving a class of systems unsolvable by diagonalization over an infinite list, to proving a specific instance from that class to have an unsolvable decision problem or to prove it solvable is a non-trivial step.

The first most obvious way to study this link between the general unsolvability of a problem and (the execution of) specific instances, is given by the thing we can hardly live without nowadays: the computer. It is, in the end, a physical realization of the intuitive notions it is all about here. Furthermore, given the intractability one is confronted with in executing particular computational systems, the computer is the perfect tool to study the properties of the systems it is the physical realization of, on the executional level.

My first experiences with computers were rather negative, and up to some years ago I even intensively hated and despised them, trying to avoid them where possible. But then I started to program. Even the famous first "hello world" exercise thrilled me, an excitement heavily influenced by my recent explorations of Martin Heidegger's philosophy. "From that day on, I became overnight a supporter of computers.".[8] The thing I liked so much about programming is that I became much more aware of the reciprocal communication process that is constantly "running" while I am doing something with my computer. Even now, is I am writing this text, there is a double translation going on, between me hitting buttons, and my computer translating them back into a visual output understandable by me, the method behind these translations being basically the same as the one mentioned in the previous section, of reducing problem A to B.

One of the first things I did with my computer when I learned to program, was to simply test out several kinds of specific instances of classes of computational systems shown to be equivalent to Turing machines. I explored many of these systems, by simply changing several parameters, visualizing the output in some or the other primitive way on my monitor, seeing what effect changing a parameter has on the behaviour of the system,... For some reason I began to experiment more and more with one specific class of such systems called tag systems, due to Emil Post who invented them in 1921 [Pos65]. It are these systems which, for me, clearly exposed the link between the theory of unsolvable decision problems and the "discourse" of the systems it is based on. As will become clear later, they began to dominate my whole research.

One of the typical features of tag systems is that they seem to have no clear link with our intuitive notion of computability. In this respect tag systems are not at all "good" formalizations of this intuitive notion. Still, since they can "compute" anything computable by Turing machines they are, from a theoretical point of

[8] This is a kind of silly annotation of a quote by Kleene, explaining his first reaction when Church mentioned his thesis to him: "*When Church proposed his thesis, I sat down to disprove it by diagonalization out of the class of the λ-definable functions. But, quickly realizing that the diagonalization cannot be done effectively, I became overnight a supporter of the thesis.*" ([Kle81a], p. 59)

view, equally suitable formalizations. It was this observation, strengthened by me having worked (a bit) with several different kinds of formalisms, that led me to the conclusion that, while the identification between "computability" and computability is basic to any proof of an unsolvable decision problem, one should be careful in fixing one such identification as being the best one. Although I do not want to doubt Turing's thesis, or any other equivalent version, it will be argued here that it is important not to necessarily doubt the formalisms, but to challenge our intuitions. And it is often in looking at the execution of such systems that this intuition can be changed.

1.4 General Outline and research Questions

As is clear from the previous, this dissertation never started from one specific research question, but rather from a general fascination with unsolvable decision problems, giving rise to many questions. Some of these will be answered here, others won't. The general purpose of this dissertation is triple.

First of all, this dissertation wants to trace back the origin of the first proofs of unsolvable decision problems and the formulation of several (theoretically equivalent) theses, in the work of Emil Post, Alonzo Church and Alan Turing. It will then be shown how the theoretical developments induced by these (and of course some other) authors is related to the rise of the computer, resulting in new problems.

Besides this more historical analysis this dissertation also wants to offer new results in the domain of computer science. In this respect an extensive part will be added on tag systems. We will describe some results on tag systems, focussing on the significance of studying specific instances of tag systems for the abstract results of unsolvability and the closely connected theses of identifying the intuitive notion of computability with a given formalism. Although the second part of this dissertation is rather extensive, we would like to warn the reader that the research presented there is merely a start. Many of the results that will be described should be regarded as initial results that are in need of more research.

The main purpose behind this dissertation though remains a philosophical one, philosophy being understood here as a method to ask questions which do not necessarily have a clear and exact answer (yet). The most obvious philosophical aspect of this dissertation is our questioning of the identification between an intuitive notion and a mathematical form. Tracing the evolution of the several forms of this identification through history, including its transformation induced by the computer, it will be argued that, although finding a "convincing" such identification – one that has a "direct appeal to our intuition" – is very basic, starting from formalisms that challenge the intuition is at least worth more consideration, both from a mathematical as well as from a philosophical point of view.

In general, I cannot but understand this research as being rooted in my philosophical background. As was said before, although Heidegger will never be mentioned here again, I know for myself that in a way his philosophy, or how I understood it, formed the main trigger for me being fascinated with computers, computations and unsolvable decision problems.

Although I consider this dissertation as philosophical in nature, it is important to stress here that the references to any philosophical literature will be restricted to a minimum. This might sound contradictory, but it isn't. From the first beginning of this research, I made a choice to stay as far away from any philosophical texts as possible. The ultimate challenge for me is to see how far I could get philosophically in restricting myself to papers and books written by logicians, mathematicians and computer scientists, complemented by my own thoughts in reading these texts, and the implementation of these thoughts into programs and forms. It is left to the reader whether I failed or succeeded in this tricky business of combining mathematics with philosophy, in the way I tried to do it.

1.4.1 Short Description of the chapters.

This dissertation will be subdivided into two main parts. The first part, called *Re-tracing*, includes the analysis and discussion of some of the original papers by Church, Post and Turing in the context of unsolvable decision problems, as well as the connection between these theoretical results and the rise of the computer. The results from this study will be linked to a more philosophical discussion on the identification between "computability" and certain formalisms.

The second part, called *tagging*, is the result of my research on tag systems. The main purpose of this part is to emphasize the significance of studying specific (classes of) systems in the context of unsolvability. We will start this part from the assumption that tag systems are particularly well-suited for this kind of study because of their focus on form rather than interpretation.

Part I: re-tracing

The first two chapters of part I, focus on the work by Church, Post and Turing. The third chapter will discuss the rise of the computer, the connection with its theoretical counterparts, as well as its significance for certain developments in the context of computability and unsolvability.

Chapter 2: The beginnings. An analysis of Church's, Post's and Turing's work before 1936. In this first chapter, the following questions will be answered: How is the work preceding the unsolvability results by Post, Church and Turing connected to their major results? How did they arrive at their results?

First of all, it should be noted that, although Post published a paper in 1936 in which a formalism is described that is quasi-identical to Turing machines, he already arrived at certain unsolvable decision problems in 1921. The analysis of Turing's work will be very short with respect to the other two analyses, since he was only 24 when his 1936 paper [Tur37] was published.

Through the discussion of this earlier work, it will be shown that the way Church, Post and to a lesser extent Turing arrived at their results, differs significantly. Despite the 'confluence of ideas' in 1936,[9] an inquiry into the work resulting into this confluence shows that the systems and ideas then developed originated in work in which the equivalences are not that evident. Based on these analyses, it will be argued that the actual use and execution of the respective

[9] As it was termed by Gandy in [Gan88].

formalisms developed by Church and Post was an important factor for the formulation of their theses. We will show that, for Church and Post, it was initially not the theoretical question of finding a proper formalization of computability to prove certain decision problems unsolvable that actually led them to their results. Rather it were the results established about the systems they developed that led them to the idea of such identification. This will be contrasted with Turing's work.

Chapter 3: 1936 In this chapter we will discuss the different theses as originally proposed by Church, Post and Turing. Focus will be put on the fact that, despite the theoretical equivalences, there are some basic differences between the theses proposed by each of these authors. In the first section, our starting point will be the 1936 papers by Church, Post and Turing, as well as Post's posthumously published manuscript [Pos65]. We will describe and discuss their several theses as originally put forward, as well as the arguments they each considered important for supporting their theses. Starting from the reviews Church wrote on Post's rsp. Turing's paper, we will then show in a next section that the significance one attaches to certain of the arguments underlying the respective theses is closely connected to the different interpretations of the actual status of the theses, and discuss in this setting several of the interpretations of the status of the theses. In the last (short) section of this chapter we will formulate some further questions with respect to the status of the theses, drawing from our results from this and the previous chapter. To be more specific, we will question in how far it can be interesting philosophically and mathematically to start from a thesis that is considered to have no serious appeal to intuition, in contrast with one that does.

Chapter 4: The computer In this chapter we will take into account the physical realization of computability, the computer. This chapter will be subdivided into three main sections. In a first section, we will discuss the rise of the computer. Focus here will be on the question in how far the developments sketched in the previous chapters can be linked to the construction of the first computers. In a second section, we will argue that already from its first use on, several pioneers understood that computers can be used to perform “experiments” not only in the context of say physics but also in the context of mathematics. Without going into much details here, the most important thing to note here is that the computer made it possible to access certain aspects of the objects studied in mathematics hardly accessible before. It allowed the analysis of the behaviour of certain functions underlying so many mathematical problems. In this respect, it will be argued that the computer is a suitable instrument to study the link between general unsolvability and particular (classes of) systems. In the last section, we will consider some developments in the context of computability and unsolvability, that are very closely connected to the actual execution of computations on the computer. To be more specific, we will consider two developments that are connected to the Turing limit of computability, i.e., computational complexity theory and hypercomputability. Our main focus in this chapter is the ongoing discussion on what is sometimes called the physical Church-Turing thesis [Gan80] and hypercomputability, a discussion that directly arises from, on the one hand, the theses as proposed by Church, Post and Turing (although one focuses in most of the cases on Turing computability) and, on the other hand, from the fact that, in a way, the computer has undermined our intuitions of computability, i.e., the computer has not remained restricted to “pure” calculations. What is at stake here is the question of whether there exist physical processes that cannot be simulated by a Turing machine, leading one to the question whether there exists procedures that can be effectively implemented but go “beyond” Turing computability.

Part II: tagging

In this second part we will make a huge jump through history, from Emil Post’s work from 1920–1921 to our own research on tag systems. It should be noted that Emil Post’s posthumously published manuscript [Pos65] describing this research, has been basic to the ideas put forward in this part. There are several reasons for this. I will merely give two of them. First, and most obvious, it was during this period that Post invented his tag systems. Secondly, one of his goals was to find the most general *form* of logic and ultimately mathematics, a feature that made it possible for him to reduce the presence of any "meaningful" concept to a minimum. This is exactly the feature that has always attracted me the most in tag systems. If you just run them on your computer, varying several parameters, it is terribly hard to superimpose any concrete interpretation on these systems, let alone, to understand how these systems are capable to compute anything we consider *intuitively* computable. It is exactly this last feature that makes tag system well-suited for a study of unsolvability that starts from an analysis of the behaviour of particular (classes of) systems.

Chapter 6: Why Tag systems? In the first (short) introductory chapter of part II, we will discuss some general features of tag systems and give an overview of the several chapters to follow.

Chapter 7: Preliminaries. Some basic aspects of tag systems. In a second chapter the reader will be made more familiar with tag systems. In a first section we will describe most of the existing literature on tag systems. Based on certain of the results described, we will propose a definition of the size of tag systems. Furthermore, some of the results will be used to show how hard it actually is to get a more formal grip on tag systems. In general, this discussion of the literature on tag systems gives an impression of the kind of approaches and the problems connected to them, that can be used to study tag systems.

In a second section we will discuss the so-called general classes of behaviour a tag system can lead to and connect them to the two forms of the problem of "tag" Post formulated, i.e. two different decision problems for tag systems.

In the next section we will give an example of a solvable tag system, to give a first idea how one might proceed to prove specific instances of tag systems solvable. Based on this example, we will prove a certain theorem for tag systems, that shows for certain tag systems that they can be decomposed in a certain number of other tag systems, their solvability thus depending on the tag systems into which they can be decomposed.

Chapter 8. Constraints for intractable behaviour. In this chapter, we will describe several "constraints" which can be used to find examples of tag systems that might be very hard to prove solvable. Two algorithms implementing these constraint will be described, and used to generate a whole class of tag systems. The tag systems generated with the second algorithm are the ones used in the experiments of chapter 8. Given the tag systems generated through these two algorithms, we will consider the idea of what will be called tag systems for which the concatenation of their respective words are rotations of the same combination. At one time, we believed this approach might result in a method to define equivalence classes for tag systems. As will be shown however, there are several problems connected to the approach, and it is in need of more research.

Chapter 9: Playing with tag systems In this chapter we will describe the results from 6 computer experiments performed on a class of tag systems generated by the second algorithm described in the previous chapter. After a short introduction of the idea of computer experiments in mathematics, we will first discuss some of the restrictions involved in the computer experiments. These concern the programming language used, the size of the sample space and the use of one specific class of tag systems.
The experiments serve several different goals, that will not be discussed here in any detail. The main purposes behind these experiments are 1. to show heuristically that the class of tag systems with 2 symbols can at least be called intractable 2. to get a better idea on what levels this intractability can be observed 3. to search for a method to define classes of tag systems. To spare the reader at least a little bit, only the first two experiments will be included in the main text, since they are considered as the most important ones. For the remaining four we will merely include the conclusions, the details of the experiments will be described in appendix C. It should be noted that we consider the result from the second experiment as the most important one. It allows one to differentiate several tag systems according to the types of periodic structures that they can generate. Of course, these classes are based on heuristic evidence and have not been proven, although it seems possible to provide such proofs

(as will be shown through an example).

Chapter 10: Universality and Unsolvability in tag systems: Some questions concerning the usefulness of small universal systems. In this last chapter, we will consider the problem of the connection between general unsolvability and the discourse of the systems the unsolvability is proven for, by starting from questions concerning the significance of small universal systems. These are considered important in this context because, on the one hand, they are particular instances of systems with an unsolvable decision problem, and, on the other hand, they are used in research on the limits of solvability and unsolvability. In a first section, a historical account will be given of some of the results significant in this context, showing why (small) universal systems are interesting. Very important here for a discussion to follow is Martin Davis's definition of universality [Dav56].
In a second section we will look at some of the reasons why the known small universal systems are in fact uninteresting. It will be argued that if one studies particular computational systems on the level of their behaviour, the known universal systems are not able to bring us any further than a study of the systems they are able to represent. In the next section it will be shown, by discussing several examples from the literature, that a study of the discourse of particular (classes of) systems is a very valuable approach, giving rise to new results in the context of studying limits of solvability and unsolvability.
In the last section, we will study limits of solvability and unsolvability in tag systems. In this section we will first of all prove the solvability of one specific class of tag systems. It should be noted that while this result was already proven by Post, the result was never published. A second result concerns the reducibility of an intricate problem from number theory, the $3n+1$-problem, to a very small tag system. This reduction will then be generalized resulting in the reducibility of any Collatz-like function to a tag system. Finally we will describe a method that *might* be used to prove that the class of tag systems with 2 symbols contains a universal tag system. This section will be ended with a general discussion of the limits of solvability and unsolvability in tag systems. In this concluding subsection, we will propose two conjectures concerning the limit

of unsolvability in tag systems, based on the results from this and the previous chapter and describe some possible approaches to tag systems that might show useful to obtain a more complete theory of tag systems.

1.5 A small note to the reader.

As is clear from the outline of this dissertation I have not restricted myself to one specific research domain or methodology. Because of the fact that I have not chosen a clear well-cut research subject, there are several problems connected to this dissertation. First of all, the dissertation as a whole is not as coherent as it could have been if I would have written a dissertation on say the history of computers. There is not one clear general research question, that is answered at the end of the dissertation. Rather there are several results presented in the different chapters, that, although they have a clear connection with each other, cannot be summarized under one heading. Secondly, the research presented here is not complete, in two respects. On the one hand, for each of the chapters, there are some gaps in the literature discussed. This shortcoming was impossible to overcome, given the time limit. On the other hand, and this is especially valid for part II, this research is still research in progress. Although I have some results on tag systems, these can only show their merit if they could be included in a more complete theory of tag systems. Thirdly, the presentation of the results, and this again especially concerns part II, is still in a rather informal style. I know for myself that if I had had more time, the formulation would have been more mathematically decent. I really hope that due to the informality of some parts, we have not given rise to too much ambiguities, unclear statements or plain mistakes.

I do not intend to safeguard myself here with this note. I do not want to give bad excuses here for the shortcomings of this dissertation. What I do intend to make clear with this note is that these shortcomings are a consequence of a choice I have made when I started with this research, i.e., to combine several domains and methods. Despite these shortcomings, I do not regret this choice. The things I have learned during the past three years are invaluable, and I am convinced that I would not have learned as much as I have if I would have chosen to restrict myself more. Even if I do not have the specialized knowledge I maybe should have on e.g. recursion theory, I know that for me the true value of this dissertation lies in its combination of philosophy, history and mathematics. During my research these three domains were never separated. The most striking example of this mix has been for me, my research on tag systems and my study of Post's work.

It is a current trend both of the humanities as of the exact sciences that one needs to specialize into one sub-sub-...-sub domain to get anywhere. This is even becoming a harder reality for the younger researchers who seem to have no other choice but to specialize. This evolution I regret. Although I am the last to say that one should not know his or her subject well-enough, otherwise one can only make mistakes, I think being able to cross the borderlines of domains, trying to link them up, is at least as important as this specialization. In my personal opinion, one of the ways to make progress is to have the freedom to mix up domains. I still do not know for myself whether I have managed to do this in any decent way. I leave it to the reader to judge whether I have failed or succeeded in this attempt to combine.

Part I

Re-Tracing

> [...] time after time I found that *because* of my ignorance of these antecedents, *I had not, nor could have, really understood those ideas.* All the logical analysis in the world will not reveal the intentions behind ideas, and without these intentions one all too easily misunderstands and misjudges the ideas and theories of a writer no longer living. [...] one also finds that current ideas and results can illuminate older and crustier ideas. The lesson seems to be this: we cannot fully understand our own conceptual scheme without plumbing its historical roots, but in order to appreciate those roots, we may well have to filter them back through our own ideas.
>
> Judson C. Webb, 1980.[10]

In the first part of this dissertation we will trace the roots of the first unsolvable decision problems and the closely connected problem of formalizing the intuitive notion of computability. Through an analysis of the work by Emil Post, Alonzo Church and Alan Turing we will show that the several formal systems considered by these logicians/mathematicians played a significant role in the actual formulation of the solutions to these problems and the later interpretation of the several theses proposed. We will then connect these more theoretical results to the rise of the computer and show how this physical realization of computability and solvability has given rise to new (philosophical and mathematical) problems and possibilities in the domain of computability and unsolvability.

[10] From [Web80], p. xii.

Chapter 2

The beginnings: An analysis of Church’s, Post’s and Turing’s work before 1936.

In this first chapter we will dig into the early beginnings of unsolvable decision problems by focussing on the work preceding the publication of the 1936 papers of the three “masters” of unsolvability: Emil Post, Alonzo Church and Alan Turing. As is pointed out in [Gan88], p. 55 by Gandy:

> It is not uncommon in mathematics – and in the other sciences – for concepts, methods, and theorems to be discovered independently and almost simultaneously [...] There is, so to speak, something in the air which different people catch.

There was definitely something in the air and Church, Post and Turing were the ones who captured it and wrote it down.[1] Before looking at what they exactly captured, it is fundamental to take a closer look at their earlier work. Especially Church’s and Post’s work deserve special attention here, since Turing was only 24 when his seminal 1936 paper was published. As was stated in the introduction the main question to be answered in this chapter is: How is the work preceding the unsolvability results by Post, Church and Turing connected to these

[1] It should be pointed out that besides Church, Post and Turing, also Kleene should be mentioned here. His contributions will be discussed in a bit more detail in Sections 2.3 and 3.2.3.

major results? How did they arrive at their results? In answering this question, it will be shown that despite the "confluence of ideas" in 1936, the preliminary work resulting into this confluence differs to an extent that can help to clarify the different interpretations and formalizations by Church, Post and Turing of the intuitive notion of "computability".

Before starting with an analysis of this earlier work, it is important to give at least some background on what was going on in the domain each of these mathematicians/ logicians worked in, i.e. mathematical logic and research on the foundations of mathematics (Section 2.1). It is impossible to be complete here. Instead of trying to give a detailed overview we will thus refer the reader to several papers and books on this subject trying to give at least a feeling of some of the work started in the 19th century on the foundations of mathematics. In a next section (Section 2.2.5), we will discuss Emil Post's work from 1918–1921. Although there is a rather huge gap between 1921 and 1936 there are clear reasons why the results from this period of research are basic to gain a better understanding of Post's 1936 paper [Pos36]. In fact, we consider these results of more significance than the 1936 paper, since they anticipate much of what was "in the air" in the thirties.

In section 2.3 we will give a detailed analysis of the work preceding Church's [Chu36c] starting from 1924 till 1935. In a next to last section 2.4 we will shortly discuss some aspects of Turing's earlier work and occupations that might help to clarify his 1936 paper [Tur37].

2.1 General Background

> For recent times have seen the development of the calculus of logic, as it is called, or mathematical logic, a theory that has gone far beyond Aristotelian logic. It has been developed by mathematicians; professional philosophers have taken very little interest in it, presumably because they found it too mathematical. On the other hand, most mathematicians, have taken very little interest in it, because they found it too philosophical.
>
> Thoralf Skolem, 1928.[2]

[2] From [Sko28], translated in [vH67], p. 512.

The developments in mathematics during the second half of the 19th century and the first quarter of the 20th have been basic to many of the developments leading to Church's, Turing's and Post's seminal work on unsolvable decision problems and the closely connected theses they each formulated. In this section we will give a *brief* overview of some of the most significant developments in this context.[3]

During the 19th century mathematicians became more aware of the significance of axioms and thinking about the foundations of mathematics thus became more explicit. The parallel postulate that had been considered true for centuries because of its appeal to intuition, and considered to follow as a theorem from the axioms of Euclidean geometry, was no longer considered absolutely true and replaced by certain other postulates, leading to the development of non-Euclidean geometries. The most famous mathematicians associated with these geometries are János Bolyai, Nikolai Ivanovich Lobachevsky and Carl Friedrich Gauss.[4]

Given these results it was now clear that no axiomatic system, close as it might be to intuition, is necessary true. As a consequence one had to develop other criteria to evaluate a given set of axioms. As is pointed out by Gandy [Gan88], for Hilbert this criterium was the consistency of a system and, as we will see in Sec. 2.3, this was also the criterium used by Church. After having described his 21 axioms for Euclidean geometry in [Hil99], Hilbert proved its consistency by interpreting the system in the real plane thus reducing the consistency of Euclidean geometry to the consistency of analysis. A decent axiomatization

[3] Many names and results such as e.g. Boole's algebraization will not be mentioned here. There are probably hundreds of papers and books on the history of the debate on the foundations of mathematics. Grattan-Guinness's [GG00] gives a very detailed bibliography, and discusses some of the less well-known contributors. An indispensable source book on the late 19th, early 20th century debate on the foundations of mathematics is Jean van Heijenoort *From Frege to Gödel* [vH67] Also Webb's book [Web80] must be mentioned here. It gives a detailed historical and philosophical analysis of the development of formalism and its connection with mechanism. It also includes a very good analysis of the Church-Turing thesis. We should furthermore point out that we will not mention Gödel's important contributions here.

[4] After having read Bolyai's treatise on hyperbolic geometry Carl Friedrich Gauss stated in a letter that he had already found similar results.

let alone a consistency proof for analysis, however, was lacking. Hilbert provided a foundation for analysis but soon understood that proving its consistency would be very hard and depended on the consistency of arithmetic. This last problem, the consistency of arithmetic, was added as the second problem of his famous list of 23 problems [Hil01].[5] Although Hilbert made a sketch for such proof [Hil05] "*searching for a completely satisfying foundation for the notion of number*",[6] it would take some years before Hilbert would again publish on the foundations of mathematics. There were several reasons for this delay. Not only had his [Hil05] been seriously criticized by Poincaré, but he also understood that his foundational research required a more logical formalism, that would be better suited for the further study of the foundations of mathematics. In the meantime, Russell had pointed out a serious flaw in Frege's [Fre79]. In the second half of the century several mathematicians like Cantor, Dedekind, Frege and Peano had investigated the proper foundations of the notion of a number and a set. Cantor is one of the founders of set theory and famous for the construction of transfinite cardinal numbers. He first used the diagonal method to prove that the set of the natural numbers is "smaller" than the set of the real numbers [Can91], i.e., $\aleph_0 < 2^{\aleph_0}$, and advanced the continuum hypothesis.[7]
Dedekind wrote two fundamental papers on the foundations of numbers, one on the reals and one on the natural numbers. In his [Ded72] he defined, among other things, the well-known Dedekind cuts. In his [Ded88] he defined finite and infinite sets of natural numbers, provided an axiomatization for arithmetic and included a definition of mathematical induction as a method of proof.[8]
Also Peano worked on the foundations of numbers and gave his famous axiomatization for arithmetic in his [Pea89]. About ten years before Peano's pa-

[5] Yandell [Yan02] has written a good survey of the research on Hilbert's problems.

[6] [Hil05], p. 131

[7] A good historical survey on set theory is [Kan96]. A list of references for further reading on Cantor's work can be found at: http://www-history.mcs.st-andrews.ac.uk/References/Cantor.html. The complete collected works by Cantor [Can32] are available on-line through: gdz.sub.uni-goettingen.de.

[8] At http://www-groups.dcs.st-and.ac.uk/ history/References/Dedekind.html a list of references for Dedekind's work can be found. The collected works by Dedekind [Ded32] are available on-line through http://gdz.sub.uni-goettingen.de.

per, Frege's *Begriffschrift* [Fre79] was published.[9] He regarded this language as a kind of *lingua characterica* for pure thought, that can be used to manipulate symbols through definite rules, avoiding any ambiguities that might come from the natural language. As he states in the introduction to the *Begriffschrift* [Fre79], p. 7:

> If it is one of the tasks of philosophy to break the domination of the word over the human spirit by laying bare the misconceptions that through the use of language often almost unavoidably arise concerning the relations between concepts and by freeing thought from that with which only the means of expression of ordinary language, constituted as they are, saddle it, then my ideography, further developed for these purposes, can become a useful tool for the philosopher.

Cantor, Frege, Peano and Dedekind were major contributors to research on the foundations of mathematics at the end of the 19th century and were an important influence on Hilbert's work. Another important influence here was Russel.[10]

As was said, Russell pointed out a fundamental problem in Frege's [Fre79]. In 1902, June 16 Russell wrote a letter to Frege [Rus02] in which he formulated his famous paradox. Together with Burali Forti's paradox[11] this is one of the famous paradoxes of set theory. These paradoxes made clear that one was in need of a precise statement of the assumptions made in set theory. As a reaction, Russell developed type theory to exclude the paradox he had pointed out. He first introduced type theory in his [Rus03], but the theory was worked out in more details in his [Rus08] and in the famous three-volume work written in collaboration with Whitehead, *Principia Mathematica* [RW13]. This monumental work was by its authors regarded as the logical formalization of the whole body of mathematics.

This was exactly the kind of formalization that could further the investigations on the foundations of mathematics Hilbert was searching for. After *Principia*'s

[9] A bibliography of secondary literature on Frege's work can be found at: http://www.philosophy.ox.ac.uk/reading_lists/mods_prelims/2000_Frege.PDF

[10] A paper by Mancosi [Man03] discusses the Russellian influence on Hilbert and his school.

[11] Formulated in [BF97].

publication (1910–1913), it was studied by several students of Hilbert. In 1917 Hilbert returned to his study on the foundations of mathematics, again emphasizing the significance of consistency proofs for arithmetic and set theory and was at that time convinced that such proofs might be found through reduction to the kind of logical formalism as proposed by Russell and Whitehead. Besides consistency proofs there were several other open foundational problems including the problem of the solvability of certain decision problems like the decision problems for Diophantine equations. Hilbert thus further devoted himself to a study of the foundations of mathematics through logic. In 1917, Paul Bernays became his assistant at Göttingen. In a series of lectures during 1917–1921 Hilbert in collaboration with Bernays and Behmann made significant contributions to the newly developing domain of what is now known as mathematical logic. As is pointed out by Sieg [Sie99] and Zach [Zac99, Zac01] the notes made for this series of lectures contain important material to understand how Hilbert was led to his finitism and laid the basis for Hilbert and Ackerman's textbook *Grundzüge der theoretischen Logik* [AH28] published in 1928. After Hilbert had supported the logicist programme developed in *Principia Mathematica* for some time, he became more and more critical about it. As is stated in his [Hil28], p. 473:

> My theory is opposed on different grounds by the adherents of Russell and Whitehead's theory of foundations, who regard *Principia mathematica* as a definitely satisfying foundation for mathematics. [...] the foundation that it provides for mathematics rests, first, upon the axiom of infinity and, then, upon what is called the axiom of reducibility, and both of these axioms are genuine contentual assumptions that are not supported by a consistency proof; they are assumptions whose validity in fact remains dubious and that, in any case, my theory does not require.

Instead of starting from type theory, Hilbert proposed a new system for studying the foundations of mathematics, its main purpose being the simultaneous development of logic and mathematics since he no longer believed that mathematics could be founded on logic alone [Hil26, Zac99, Zac01, Sie99].

In the meantime, also Brouwer's intuitionism had gained ground in the domain of research on the foundations of mathematics. Brouwer's intuitionism leads to a form of constructive mathematics, rejecting certain principles of mathematics like the law of excluded middle. Although Hilbert's finitary programme cannot be regarded as a mere reaction against the intuitionist programme [Sie99], their critique was not without influence on Hilbert's programme.[12] He now started from an explicit *finitary* point of view to further develop what is now known as Hilbert's proof theory. Furthermore, Hilbert wanted to justify the use of certain principles and modes of reasoning rejected by Brouwer and others, because they presuppose infinite totalities. In the development of his proof theory, mathematical proofs and propositions had to be turned into finite objects, constructed through derivations from axioms according to strict rules. Metamathematics is then the study of these finitary objects and their relations. Hilbert's finitism however did not imply the exclusion of the notion of infinity. On the contrary, as is stated in one of the most famous quotes by Hilbert from his beautiful text *On the infinite* ([Hil26], p. 376):"*No one shall be able to drive us from the paradise that Cantor created for us.*" To Hilbert one is allowed to work with the infinite but only through the kind of finitary framework he proposed ([Hil26], p. 392):

> The final result then is: nowhere is the infinite realized; it is neither present in nature nor admissible as a foundation in our rational thinking – a remarkable harmony between being and thought. We gain a conviction [...] that if scientific knowledge is to be possible, certain intuitive conceptions and insights are indispensable; logic alone does not suffice. The right to operate with the infinite can be secured only be means of the finite. The role that remains to the infinite is, rather, merely that of an idea – if, in accordance with Kant's words, we understand by an idea a concept of reason that transcends all experience and through which the concrete is completed so as to form a totality – an idea, moreover, in which we may have unhesitating confidence within the framework furnished by

[12] In [Man98] one can find a collection of 25 papers translated into English of some of the leading mathematicians from the beginning of the 20th century, focussing on the debate between Brouwer and Hilbert. Each paper is discussed in detail and placed in its proper historical context. Furthermore the book gives a detailed bibliography containing both primary as well as secondary sources.

> the theory [...]

If one wants finitary proofs for any proposition in mathematics a natural question to be asked is of course whether every problem in mathematics can be solved through such finite means.
It is exactly this problem that led to the formulation of the *Entscheidungsproblem* for first-order predicate calculus, formulated by Ackerman and Hilbert in their [AH28]. The first use of the word *Entscheidungsproblem* is most probably due to Behmann, a student of Hilbert, who explained it as follows ([Beh22], p. 166):

> A quite definite generally applicable prescription is required which will allow one to decide in a finite number of steps the truth or falsity of a given purely logical assertion; or at least precise limits should be given within which an effective prescription of this kind can be found.[13]

However, already before Behmann's use of the notion *Entscheidungsproblem* Hilbert had formulated another important decision problem as one of the 23 problems he confronted the mathematical community with at the beginning of the 20th century, i.e. the decision problem for Diophantine equations. This long-standing problem was finally solved in the negative by Yuri Matijasevich [Mat70] in 1970. Hilbert formulated the problem as follows ([Hil01], p. 19):

> Given a diophantine equation with any number of unknown quantities and with rational integral numerical coefficients: to devise a process according to which it can be determined by a finite number of operations whether the equation is solvable in rational integers.[14]

In both the statement of the Entscheidungsproblem as well as the problem now known as Hilbert's tenth problem the use of the word finite is basic. Indeed,

[13] Translated from German in [Gan88], p. 62.

[14] "*Eine Diophanstische Gleichung mit irgend welchen Unbekannten und mit ganzen rationalen Zahlencoefficienten sei vorgelegt: man soll ein Verfahren angeben, nach welchem sich mittelst einer endlichen Anzahl von Operationen entscheiden läßt, ob die Gleichung in ganzen rationalen Zahlen lösbar ist.*". Translated in English by David E. Joyce available through: http://aleph0.clarku.edu/ djoyce/hilbert/toc.html.

what one is asking for is to find a general finite method to solve any instance of the problems involved.

Two other famous decision problems had also already been formulated. In 1911 Max Dehn had formulated the decision problem for groups (and other related problems). It was solved in the negative by Novikov and Boone in the sixties [Boo66]. Dehn defined the problem in terms of finding a solution in a finite number of steps.[15] In 1914 then, Axel Thue first stated the word problem for semi-groups, and also asked for a finite method to solve the problem. He imposed the extra requirement that one should not only be able to prove that something is solvable by proving that there exists a general finite method to solve the problem, but that one should also be able to compute a bound on the number of steps needed to make the calculation ([Thu14], p. 4):

> One can now formulate the general problem: Given any two sequences of symbols A and B, find a method to determine, after a calculable number of operations, if two arbitrary sequences of symbols are or are not equivalent to the sequences A and B.[16]

As is pointed out by Gandy [Gan88], to calculate a bound on the number of steps needed to solve a given instance of a decision problem is not a necessary requirement to prove that a given decision problem is solvable. Although in practice this bound is a basic requirement, one "only" has to prove that a given decision procedure will always end in a finite number of steps resulting in a solution, to establish the theoretical result.

The only way to prove any of these problems unsolvable, was to find a good formalization of certain intuitive notions such as the notion of computability. In the twenties however, many mathematicians, including Hilbert, shared the optimism that it would be possible to find a solution for any mathematical prob-

[15] "*Man soll einde Methode angeben, um mit einder endlichen Anzahl von Schritten zu entscheiden [...]*", [Deh11], p. 117.

[16] "*Man kann sich nun die große allgemeine Aufgabe stellen: Bei beliebiger Wahl der gegebenen Zeichenreihen A und B einde Methode zo finden, durch welche man nach einer berechenbaren Anzahl von Operationen immer entscheiden kann, ob zwei beliebig gegebene Zeichenreihen in Bezug auf die Reihen A und B äquivalent sind oder nicht.*" I am indebted to Maarten Bullynck for his translation of the German quote.

lem through finite means,[17] i.e., they believed that any decision problem could be proven solvable. This optimism is still best expressed in the famous quote by Hilbert ([Hil30], p. 387):

> The true reason why Comte could not find an unsolvable problem, lies in my opinion in the assertion that there exists no unsolvable problem. Instead of the stupid Ignorabimus, our solution should be: We have to know. We will know.[18]

Others however did not share this optimism and understood the possibility of such positive solution as the end of mathematics as it existed at that time. This is e.g. clearly echoed in a quote by Von Neumann ([vN27], 11–12):

> So it appears that there is no way of finding a general criterion for deciding whether or not a well-formed formula is a theorem. (We cannot at the moment prove this. We have no clue as to how such a proof of undecidability would go.) But this ignorance does not prevent us from asserting: As of today we cannot in general decide whether an arbitrary well-formed formula can or cannot be proved from the axiom schemata given below. And the contemporary practice of mathematics, using as it does heuristic methods, only makes sense because of this undecidability. When the undecidability fails then mathematics, as we now understand it, will cease to exist; in its place there will be a mechanical prescription for deciding whether a given sentence is provable or not.[19]

[17] To Gandy this optimism is one of the main reasons why no one belonging to Hilbert's school proved the Entscheidungsproblem unsolvable.

[18] "Der wahre Grund, warum es Comte nicht gelang, ein unlösbares Problem zu finden, besteht meiner Meinung nach darin, daßes ein unlösbares Problem überhaupt nicht gibt. Statt des törichten Ingnorabimus heiße im Gegenteil unsere Losung: Wir müssen wissen, Wir werden wissen." I am indebted to Maarten Bullynck for translating the German quote.

[19] "Es scheint also, daß es keinen Weg gibt, um das allgemeine Entscheidungskriterium dafür, ob eine gegebene Normalform a beweisbar ist, aufzufinden. (Nachweisen können wir freilich gegenwärtig nichts. Es ist auch gar kein Anhaltspunkt dafür vorhanden, wie ein solcher Unentscheidbarkeitsbeweis zu führen wäre.) Diese Ungewißheit hindert uns aber nicht daran, festzustellen: Heute ist es nicht allgemein zu entscheiden, ob irgendeine gegebene Normalform a (bei der in folgenden zu beschreibenden Axiomenregel) beweisbar ist oder nicht. Und die Unentscheidbarkeit ist sogar die Conditio sina qua non dafür, daß es überhaupt einen Sinn habe, mit den heutigen heuristischen Methoden Mathematik zu treiben. An dem Tage, an dem

As was shown here, starting in the 19th century, the foundations of mathematics became the object of study for many mathematicians and logicians. Research on these foundations finally led to the finitary programme of Hilbert's school and resulted in a new discipline lying at the borderline between logic and mathematics, i.e. mathematical logic. In the end, research in this domain would uncover its own limits.

2.2 From solvability to unsolvability: Emil Post's frustrating problem of "Tag"

2.2.1 Introduction

In 1965 Martin Davis, a student of Emil Post, published an important anthology of fundamental papers on unsolvable decision problems and undecidable propositions, including the seminal papers by Church, Turing and Post. This collection of papers has been invaluable in many respects for my research, especially because it includes a paper by Emil Post entitled *Absolutely Unsolvable Problems and Relatively Undecidable Propositions. Account of an Anticipation.* The paper was never published before. It describes the results Post had established during his Procter fellowship from 1920-21 in Princeton. Fifteen years before the publication of Church's and Turing's now classic results on the *Entscheidungsproblem* [Chu36c] [Tur37], Post had already found that certain decision problems closely related to the Entscheidungsproblem are unsolvable and inferred from these results that any finite system of symbolic logic relative to a certain class of systems, must be incomplete. Post had thus anticipated results similar to the fundamental results by Gödel, Church and Turing. At that time, he did not prepare a paper for publication describing these results. Only 20 years later he submitted the paper mentioned above to the *American Journal of*

die Unentscheidbarkeit aufhörte, würde auch die Mathematik im heutigen Sinne aufhören zu existieren; an ihre Stelle würde eine absolut mechanische Vorschrift treten, mit deren Hilfe jedermann von jeder gegebenen Aussage entscheiden könnte, ob diese bewiesen werden kann oder nicht." English translation from [Gan88], p. 67.

Mathematics but it was rejected by the editor Hermann Weyl. Following the referee's recommendation that the normal form theorem included in his [Pos65] is new and important, and might be used to obtain proofs of unsolvability for various mathematical problems such as the word problem of groups [20] Post reworked the paper and finally a very abbreviated version (to about one third) of the paper was accepted [Pos43], the historical account being reduced to a long footnote at the end of the paper.

In his letter of submission of the original version, Post gave several reason why he did not submit these results back in the twenties.[21] First of all, he had already experienced several problems to publish other of his results, including his Ph.D. dissertation which was only accepted after its original length was reduced to one third. Secondly, his efforts to obtain a full, more detailed proof of his results, searching for a more complete analysis supporting the thesis he had to assume, similar to Church's and Turing's, for his results to be valid, were interrupted by a manic-depressive illness he suffered from during his whole career.

In his letter explaining why Post's *Account of an anticipation* was not accepted, Hermann Weyl states:[22]

> [...] I have little doubt that twenty years ago your work, partly because of its then revolutionary character, did not find its due recognition. However, we cannot turn the clock back; in the meantime Gödel, Church and others have done what they have done, and the American Journal is no place for historical accounts;... (Personally, you may be comforted by the certainty that most of the leading logicians, at least in this country, know in a general way of your anticipation.)

Post himself was very well aware of the fact that Gödel, Turing and Church had already published similar results in the thirties. The question of why he

[20] See [Dav94] where part of this referee report is quoted.

[21] The full text of the letter was published in the introduction Martin Davis wrote to Post's collected works [Dav94] as well as in [Dav89].

[22] Herman Weyl, in a letter to Post dated March 2, 1942. Quoted from the introduction of [Dav94].

nonetheless wanted his results to be published is thus significant. Assuming that it was self-absorption that led him to this seems to be wrong given the modesty with which he writes, his care in referring to other sources, and not in the least, the way in which he acknowledged the fact that Gödel fully deserved his credit. In a postcard dated October 29, 1938 written after he had met Gödel, Post writes ([Göd03b], p. 169):

> [...] for fifteen years I had carried around the thought of astounding the mathematical world with my unorthodox ideas, and meeting the man chiefly responsible for the vanishing of that dream rather carried me away. [...] As for any claims I might make perhaps the best I can say is that I would have *proved* Gödel's Theorem in 1921 - had I been Gödel.

One day later, he wrote a letter to Gödel in which he stated the following hard words: "[...] *after all it is not ideas but the execution of ideas that constitute[s] a mark of greatness.*" ([Göd03b], p. 172). In a way, in 1936 he had bad luck again. He formulated and formalized a concept of computability [Pos36] which is almost identical to that formulated and published by Turing at about the same time [Tur37]. Post's paper though did not contain the concept of a universal machine, nor the important theorems on unsolvable decision problems from Turing's paper. These misfortunes seem not to have discouraged Post: in the forties he wrote his important paper on recursion theory [Pos44], he published his proof of the unsolvability of the Correspondence Problem [Pos46], and furthermore proved the unsolvability of the word problem for semi-groups [Pos47]. A further important contribution was not in mathematical logic but in group theory, namely his long paper on polyadic groups [Pos40].[23]

The work of Emil Post has had many influences ranging from mathematical logic to computer science.[24] To give some examples, he is known as one of the

[23] Another part of the research of Post that should be mentioned here is his work on provability and definability. He tried to find an absolute and fundamental explication of these notions comparable to those already offered in [Tur37, Pos36, Chu36c] for the notion of computability. There are only two abstracts published on this research [Pos53b, Pos53a], but as mentioned by Martin Davis in his introduction to the Collected Works of Emil Post [Dav94] his notes in bound notebooks on this subject are still available.

[24] A paper surveying Post's influence on computer science is Davis's [Dav89].

co-founders of recursion theory [Dav94, Sti04], he seems to have influenced John Backus in formulating what came to be called "Backus Normal Form" [Bac80, Bac81, Dav88], he had an impact on the study of NP-complete languages in structural complexity theory [Dav94, Sad98], and influenced Chomsky's construction of context-free grammars [CM58] through his systems in canonical form C [Dav88].[25] Furthermore Post was one of the first to prove the unsolvability of certain decision problems further removed from mathematical logic like the unsolvability of the word problem for semi-groups [Pos47], and the unsolvability of the Post correspondence problem [Pos46], which has become an important tool to obtain unsolvability results in formal language theory [Dav89]. It should also be mentioned here that there are some indications that Post's work on rewriting systems might have had an influence on cryptography [AA93].

So why did Post in the end submit his *Account of an anticipation*? In the introduction to his [Pos65] Post fully acknowledges the fact that there would be little point in publishing his "anticipation [...] merely as a claim to unofficial priority" to the results of Gödel, Turing and Church. But he also remarks:

> [...] with the *Principia Mathematica* of Whitehead and Russell as a common

[25] As for the exact extent to which Chomsky was influenced by Emil Post's work, Chomsky pointed out to me:"In the days when I was following these topics closely – some years ago – Post systems were little known, apart from Martin Davis's book [Dav58], where I learned about them, then checked some of the papers. I was interested at the time in automata theory and possible applications to linguistics. I'd studied standard versions of recursive function theory (Kleene, etc.), but when I came across Post's work (in Davis) it was obvious that this was a good framework for systems of the sub-recursive hierarchy that could be adapted to the study of language, specifically context-sensitive and context free grammars (and as a subcase, finite automata and Markov sources, mostly in order to show that they couldn't work for language – they were all the rage at the time in information sciences, mathematical psychology, and related areas). But that's the limit of the influence. My first paper about this was in 1956, at the IRE (Institute of Radio Engineers), but I also pointed out in that paper that for other reasons even the richest systems of this kind didn't have the right properties for natural language, in my opinion (then, or now)."

> starting point, the roads followed towards our common conclusions are so different that much may be gained from a comparison of these parallel evolutions.

He then mentions three reasons why his work can still be of significance. First, he focuses on the outward form of symbolic expressions and the possible operations thereon – an approach which resulted in a class of very general and, at first sight, simple systems [Pos65], pp. 341–342:

> Perhaps the chief difference in method between the present development and its more complete successors is its preoccupation with the outward forms of symbolic expressions, and possible operations thereon, rather than with logical concepts as clothed in, or reflected by, correspondingly particularized symbolic expressions, and operations thereon. While this in part is perhaps responsible for the fragmentary nature of our development, it also allows greater freedom of method and technique.

Second, he added another equivalent formulation to the list of general recursiveness, λ-definability and Turing computability, viz., normal forms. Also, Post considered not the intuitive idea of a computation (as Turing) or the concept of effective calculability (as Church), but that of a generated set. In this respect Post formulated a thesis similar to that by Turing and Church already in 1921. A third and final reason, for the significance for making available his results from 1920-21 is his conclusion that "*mathematical thinking is, and must be, essentially creative*" from which he concludes that such "*developments will result in a reversal of the entire axiomatic trend of the late 19th and early 20th centuries, with a return to meaning and truth*".[26]
Starting from his Ph.D. dissertation, we will show in this section how Post was led from the optimistic opinion, that there exists a single algorithm for the whole of mathematics, to the idea that no such algorithm can ever be found.[27]

[26] It should be noted here that this statement should not be confused with those made by Lucas and Penrose [Luc61, Pen94] who concluded, on the basis of Gödel's incompleteness result, that mathematical thinking and understanding must be non-computable.

[27] The analysis presented in this section appeared as *Closing the Circle. An analysis of Emil Post's early work* [Mol06a]. It should also be pointed out here that Martin Davis's [Dav82] discusses Post's [Pos21a, Pos65] where Post's thesis is mentioned for the first time.

Focus will be put on the first two reasons pointed out by Post for wanting to publish his *Account of an anticipation.* We will return to all three reasons in our comparison of the respective theses put forward by Post, Church and Turing and their respective interpretations attached to it (See Ch. 3). We will argue how it was Post's focus on outward form rather than on logical concepts that led him to the construction of his form of "tag" – a form that will play a major role in part II of this dissertation – and how it were his "experiences" in working with special cases of this form, that played a basic role in the formulation of his very important systems in normal form. These last systems finally led him to a thesis similar to Church's and Turing's and a proof of the general unsolvability of certain decision problems to these systems in normal form. Through the analysis of Post's earlier work we will be able to show how he arrived at the formalization of the intuitive notion of a generated set. As will become clear, it was not an analysis of the intuitive notion and thus the idea of finding such identification, but rather the formalisms themselves that resulted in Post's thesis. The results from this analysis will avail themselves as important in our recurring discussions on identifying an intuitive notion with a formal (class of) systems throughout this dissertation and will contribute to one of our main questions of trying to understand the link between the proof of the unsolvability of a whole class of systems, and the actual execution of these systems.

In 1954 Post died from a heart-attack, after one of the electro-shock treatments he received for the illness that pursued him throughout his entire career. Post had had a very difficult life, and had to cope with several problems that many people would not be able to combine with an academic career. This enforces even more respect for Emil Post both as a mathematician as well as a person. The following quote by Martin Davis gives an impression of the difficulties Post had to overcome during his life [Dav94]:

> Post's life was a struggle with adversity. He managed well the handicap he suffered in childhood when he lost an arm in an accident. But in his scientific labors, he had to overcome obstacles that would have daunted most. He suffered all his adult life from crippling manic-depressive disease at a time when no drug therapy was available for this malady. Until 1935, he was unable to obtain a regular

> academic position, making his living, for the most part, by teaching in the New York high school system. At City College he worked under conditions that would seem intolerable nowadays. The standard teaching load was 16 contact hours per week. There were no individual faculty offices (everyone shared one large room with a huge table in the center), so Post did his research sitting at a desk in the living room in his small apartment while his young daughter was required to maintain silence. There was no secretarial help, and Post had to type his own letters of recommendation for students unless his wife did it for him. His research in mathematical logic was ahead of its time and very much out of the mainstream of mathematical research in the United States. Post suffered repeated episodes of mania which required institutionalization. Electro-shock therapy was believed by his physicians and his family to be the most efficacious treatment. His tragic death from a sudden heart attack occurred in a mental institution shortly after one of these treatments.

2.2.2 Focus on Form

> It is a familiar fact to the student of algebra or geometry that many a seemingly difficult problem may often become remarkably simple when one makes the right change in variable or the appropriate choice of coordinates. In the same way a suitable system for representing numbers will sometimes facilitate and simplify problems in higher arithmetic. Conversely, he who devises a new numerical notation is sure to discover new properties of numbers and to realize more fully the difference between a symbol for the number and the number itself.
>
> Derrick Lehmer, 1935.[28]

As was explained in the previous section, there was clearly something in the air in the twenties and thirties of the 20th century. Research on the foundations of mathematics gave rise to a new discipline called mathematical logic. Post can be regarded as one of the main and early contributors to this new domain.

[28] [Leh33], p. 460.

Towards a *general* theory of elementary propositions

In 1917 Post got his B.S. at City College, and went to Columbia University where he wrote his Ph.D. dissertation. It was here that he became acquainted with Russel's and Whitehead's massive three volume work in Kassius J. Keyser's seminar on *Principia Mathematica* [RW13]. Another important influence on Post's earlier work was the recently published book *A survey of Symbolic Logic* [Lew18] by C.I. Lewis, where it is shown how any system of symbolic logic working with an infinite set of variables, can be transformed into a logic that deals with finite strings of symbols over a finite alphabet.[29]

These two influences are visible in Post's dissertation, published in 1921 as *Introduction to a general theory of elementary propositions*, where he developed a general form of symbolic logic based on finitary symbol manipulations, to be studied by mathematical methods, abstracting from meaning.[30] At that time Post was convinced that the whole of mathematics could be formalized into a system of finitary symbolic logic and that *Principia* would be shown to be complete, decidable and consistent, thus sharing the optimism of some other mathematicians during that period. In his Ph.D. dissertation he isolated the propositional part of *Principia*, nowadays called propositional calculus, and proved that its axioms are complete and consistent. To prove this, he developed the truth-table method and showed that this method provides a solution to the decision problem for propositional calculus.[31] Basic for Post's disserta-

[29] Even in his [Pos43], the very abbreviated version of Post's [Pos65] that was finally published in the *American Journal of Mathematics* Post starts by referring to Lewis' [Lew18], and makes explicit that he does not work with "the usual form of symbolic logic with its parenthesis notation and infinite set of variables", but with the equivalent forms of symbolic logic for which "the *enunciations*, i.e., formulas of the system [what we now call w.f.f.'s], are finite sequences of letters, the different letters constituting a once-and-for-all given finite set." [Pos43], p. 197

[30] The paper [Dav95] by Martin Davis and his introduction to the Collected Works [Dav94], gives a further analysis of Post's Ph.D. thesis.

[31] Although Post is known as one of the first to prove propositional calculus decidable and complete, it should be pointed out that Zach [Zac99] has shown that similar results were already obtained by Hilbert and Bernays: "*These results include: explicit semantics for propositional logic using truth values, decidability of the set of valid propositional formulas, completeness of the axiom systems considered relative to that semantics, as well as what is now called*

tion is that he enunciated the distinction between the theorems of the system and the theorems about the systems, a differentiation that was at that time far from evident‘.[32] Not making such a distinction is identified by Post as "an incurable defect" in many developments of symbolic logic at that time. As Post points out, *Principia* appeared not to suffer from this defect, but this does not mean that he was completely satisfied with the system presented by Russell and Whitehead.

It is at this point that his dissertation, notwithstanding its affiliations to the ideas of contemporary logicians and mathematicians, marks the beginning of the development of a theoretical framework that would remove him from these common origins. Post wanted to develop the most general *form* of symbolic logic and mathematics. As is already clear from the title of his dissertation, he started this project by doing so for propositional calculus. To Post, *Principia* was not the right format to find the most general form of symbolic logic and ultimately mathematics [Pos21a, pp. 163–164]:

> But owing to the particular purpose the authors [Russell & Whitehead] had in view they decided not to burden their work with more than was absolutely necessary for its achievements, and so gave up the generality of outlook which characterized symbolic logic. [...] we might take cognizance of the fact that the system of 'Principia' is but one particular development of the theory [...] and so [one] might construct a general theory of such developments.

In his dissertation Post went on to develop more general forms and methods for symbolic logic, which he wanted to use as "*instruments of generalization*" to study more general properties of logic and mathematics like, e.g., decidability

Post completeness, consistency and independence results, general three- and four-valued matrices, and rule-based derivation systems. All these results were obtained independently of logicians to whom they are usually credited (notably Pierce, Wittgenstein, Post, and Lukasiewicz). Far be it from me to dispute their priority. After all, Hilbert and Bernays's work remained unpublished, and in some respects the work by those other logicians investigates the questions at hand more deeply or is more precise than Hilbert and Bernays's." [Zac99], p. 332

[32] "*We here wish to emphasize that the theorem of this paper are* about *the logic of propositions but are* not included *therein*" [Pos21a], pp. 163–164

and completeness. One such instrument was his truth-table method of which he states [Pos21a, p. 166]:

> Let us denote the truth-value of any proposition by + if it is true and by – if it is false. This meaning of + and – is convenient to bear in mind as a guide to thought, but in the actual development that follows they are to be considered merely as symbols which we manipulate in a certain way.

As is clear from this quote, to Post, systems and methods of symbolic logic have to rely on finitary symbol manipulation, and should not be bothered with the specific meanings involved. As was shown in the introduction (2.2.1) this focus on outward form, rather than on logical concepts is exactly where Post's work differs from that by Church and Gödel. Making abstraction from specific logical concepts and thus focussing on form is one of the main features of Post's earlier work and makes it mathematical rather than logical in nature.[33] This method of generalization is made explicit in his dissertation not only through the development of the truth-table method but also through the further generalization of two-valued logics to many-valued logic.

A more important generalization in this context is the proposal of a general form for systems of symbolic logic. In his dissertation he identifies this further generalization as *generalization by postulation* and later called systems of this form, systems in canonical form *A*. Within these systems strings are produced through finitary symbol manipulation and they can thus be regarded as combinatorial systems.[34] These were at first developed as generalizations for propositional calculus, but Post later realized that they are far more general. Without going into the details of systems in canonical form *A*, it is important

[33] This method of generalization replacing the more usual semantical approach by logicians in the twenties was also discussed in [Dav82]. Davis states [Dav82], p. 18: "*Whereas Hilbert and his school went on to approach the decision problem for quantification theory semantically, Post evidently felt that was not a promising direction because the combinatorial intricacies of predicate logic were too great to penetrate in that manner, and what he proposed instead was to simplify by generalization.That is, he proposed to abstract from the kind of rules that occur in quantification theory to obtain a class of rules which included them.*"

[34] As Davis remarks in his introduction to [Dav94] the strings produced by such systems can be regarded as being an arbitrary recursively enumerable set of strings on a finite alphabet.

to notice the explicit use of the word "form". Instead of constructing one specific system with a specific interpretation, Post constructs a form, a general set of rules, that includes not one but an infinite number of systems of symbolic logic. Indeed, starting from this form, it is possible to generate infinitely many systems of logic – each with its own axiom(s) and production rules – which all share the same form, i.e., formalization is taken literal here.

Ambitions of a Ph.D. student: Solving the finiteness problem for *Principia*

After the generalizations developed in his Ph.D., Post wanted to go ahead with his programme, as he already announced in the introduction of his [Pos21a]. He wanted to extend his results for the propositional calculus to the whole of *Principia* and find a positive solution for the decision problem for the entire *Principia* and thus also solve the *Entscheidungsproblem* that had hardly been formulated as a problem at the time Post considered it. Post used the term *finiteness problem* to talk about decision problems, and we will use his terminology in the remainder of this section. This programme of proving the solvability of the whole *Principia* can at least be called ambitious. As Martin Davis described it ([Dav94]):

> Since *Principia* was intended to formalize all of existing mathematics, Post was proposing no less than to find a single algorithm for all of mathematics.

Post did not want to use the formalism of *Principia* since he believed that the particular processes used in this framework would hardly allow for such more general results and thus started from his systems in canonical form *A*.[35]

In the abstract [Pos21b] Post presented a solution of the finiteness problem for a certain subclass of systems in canonical form *A*. In his *Account of an anticipation* he points out the difference between this solution and the solution for the finiteness problem for propositional calculus [Pos65, pp. 345-346]:

[35] E.g., in his dissertation he states that he is convinced that due to these particular processes it would have been hardly possible to develop a general theory of propositions as well as the results from his dissertation, if he would have used the formalism presented in *Principia* (See [Pos21a], p. 164).

> We shall say that we thus solved the finiteness problem for the (∼, ∨) system. While this solution was purely formal, nevertheless it was suggested by the intuitive interpretation of "∼" and "∨". For the above generalizations such interpretation are not at hand. Nevertheless (...) the writer solved the finiteness problem for those of the above systems in which the primitive functions are all functions of one variable, the resulting relative simplicity of the systems allowing a direct analysis of the formal processes involved.

As is clear from this quote, Post indeed saw certain advantages in the more abstract class of systems in canonical form *A*. Indeed, after having noted that his solution for the finiteness problem for propositional calculus was suggested by the interpretation of "∼" and "∨", he immediately adds that such interpretation is lacking for the subclass of systems in canonical form *A* he considered, thus allowing for a more direct analysis of the formal processes involved. He could study these systems as being pure symbol manipulating systems that generate logical propositions without having to take notice of the content of what precisely is deduced.

Being convinced that it would be more straightforward to find a positive solution to the finiteness problem for systems in canonical form *A*, he wanted to prove the solvability of the finiteness problem for restricted functional calculus (first-order predicate logic) contained in *Principia* by reducing it to a system in canonical form *A* and then solve the finiteness problem for these formally simpler systems in canonical form *A*. He proved this through reduction to what he called a system in canonical form *B* and then reduced systems in canonical form *A* to *B*.[36]

By the time he got these results (October 1920), he was already awarded the prestigious Procter fellowship at Princeton. He now reoriented his research towards a problem that is closely connected to the finiteness problem, namely the problem of determining for any two expressions of a given system, what sub-

[36] As he remarks, some slight adjustments of his canonical form *A* made such a reduction proof for the functional calculus easier. It are systems in this adjusted canonical form he called systems in canonical form *B*. It should be further noticed that about one year later he also proved the reducibility of canonical form *B* to *A* so the equivalence of these two forms was established.

stitutions would make those expressions identical. This problem would nowadays be called the *unification problem for the ω order predicate calculus* (See [Dav94]).[37] However a solution for this problem for the whole of *Principia* was not immediately at hand – "[this] *general problem proving intractable*". In trying to solve it, Post then applied a technique which was at that time already rather familiar to him, namely simplifying and abstracting from the original problem. This abstraction process resulted in the problem of "tag".

2.2.3 The Problem of "Tag".

Post's method of generalization, abstracting away from meaning, led him to a class of symbol manipulating systems for which he formulated a problem closely connected to the finiteness problem. He arrived at this problem by performing several successive simplifications on the above mentioned unification problem for *Principia.* Despite the expected simplicity of the problem in this reduced form, a solution was considered to be fundamental for the further development of his research. It was not only considered relevant for the above mentioned unification problem, but also for a solution of the finiteness problem for those systems in canonical form *A*, which go but a little beyond those which involve only primitive functions with one argument (and were already proven to be solvable) [Pos65], p. 361:

> The general problem proving intractable, successive simplifications thereof were considered, one of the last being this problem of "tag". Again, after the finiteness problem for systems in canonical form *A* involving primitive functions of only one argument was solved, an attempt to solve the problem for systems going, it seemed, but a little beyond this one argument case, led once more essentially to the selfsame problem of "tag". The solution of this problem thus appeared as a vital stepping stone in any further progress to be made.

Although a solution for this problem of "tag" was considered to be a vital stepping stone, Post believed that finding a solution for his problem of "tag" would

[37] A very trivial example of unification is to make f(x) identical to f(y) by substituting x for y.

be rather straightforward. However, after about nine months of work he realized that this judgement was seriously in error and he was finally led to a reversal of the direction of his entire program.

A form of "tag", i.e. a tag system, is defined as follows.[38] Given a positive integer ν, and an alphabet $\Sigma = \{0, 1, ..., \mu - 1\}$ consisting of μ symbols. With each of these symbols one associates a word over the alphabet:

$$\begin{array}{ccc} 0 & \rightarrow & a_{0,1}\, a_{0,2} ..... a_{0,\nu_0} \\ 1 & \rightarrow & a_{1,1}\, a_{1,2} ..... a_{1,\nu_1} \\ ... & .... & .... \\ \mu - 1 & \rightarrow & a_{\mu-1,1}\, a_{\mu-1,2} ..... a_{\mu-1,\nu_{\mu-1}} \end{array}$$

Now, given an initial string A over the alphabet, "tag" at the right end of the string the word associated with the leftmost symbol of A, and remove at the left end the first ν symbols. Apply this tagging and removing operations on the new resulting string A', which results in a new string A'',... Post gives the following seemingly very simple example: $A = \{1, 0\}$; 1 → 1101; 0 → 00; $\nu = 3$. If the initial string is "101010011101011111001", applying the rules of this tag system results in the following productions:

101010011101011111001

010011101011110011101

01110101111001110100

1010111100111010000

[38] I am indebted to Martin Davis for calling my attention to the origin of the word "tag". As a noun, a "tag" designates a physical object to label something – like the price of an item in a shop. The respective verb "to tag" then designates this operation of putting such a tag on something. A derived meaning can be found in the children's game in which a pursuer tries to catch the pursued and "tag" her (or him) with a touch. It is this meaning of the verb that gave the problem of tag its name: in an earlier formulation of the problem, every ν-th letter of the sequence was merely checked off (and not removed) while at the same time the respective sequence was added to the right end of the sequence. Starting on the left the mark then moves more and more to the right, "pursuing" the right end of the string.

011110011101000000

....

Post formulated two forms of the problem of "Tag". In its first form the problem is to obtain for a given basis, a general method to decide for any initial string *I* whether the process of tagging and removing letters will ever produce the empty string and thus come to an end. In its second form – where the initial sequence *I* is considered as being part of the basis – the problem for a given basis is then to find a general method for determining for an arbitrary string and this basis whether it will ever be generated by the given basis. It is the problem posed in its second form – as Post remarks – that arose in connection with the finiteness problem.

At first, Post was very optimistic about finding a solution for solving the problem of "tag", given the deceiving simplicity of the general form. Soon, however, he understood that he was misled. During his research on tag systems, Post was obliged to apply the more "experimental" practice of working out specific cases and varying several parameters in order to infer certain properties of tag systems and identify the different classes of behaviour.[39] Despite their seeming simplicity tag systems are well-known for their intractability. In order to get a

[39] The notion "experimental" is placed between double quotes here, because it is in no way whatsoever intended as a kind of special method intervening on the usual methods of mathematics. It merely indicates the fact that Post had to test several tag systems to gain more information about these systems, a method that seems unavoidable for any mathematician when intractable systems are involved. I would like to thank Martin Davis here, because it was through his comments that I became more aware of the fact that one should be very careful in using such terminology. In the first draft for my paper [Mol06a] I all too easy used the word "experimental" in the context of Post's work on tag systems without having thought about it enough. It was through Davis' comments that I understood that it can be a tricky business to differentiate an experimental approach from some other kind of method used by the practicing mathematicians. During my own research on tag systems, presented in part II of this dissertation I for myself began to better understand that in a way it is sometimes hardly possible to differentiate an "experimental" method from some other method when actually doing mathematics, searching for results, and I often remembered the words by Davis. Getting a better understanding of mathematics, by actually doing it, is probably the most important and valuable lesson I learned in doing my research.

mathematically rigorous grip on these systems, if possible, you first have to find out how these systems behave under several different conditions. For certain of these conditions, you don't need to 'test' anything on paper. It is, for example, trivial to see why the class of tag systems with $v = 1$ is solvable, you don't even have to write anything down to understand this. But what about the class of tag systems with $v = 2, \mu = 2$? As Post himself remarks, this case already demanded considerable effort and he considered the proof of the solvability of this class as the major result of his work as a Procter fellow.[40] But how would one start with such a proof? More generally, how can one start with any mathematical proof for these systems without having any information about what kind of behaviour several initial conditions can lead to, without e.g. knowing about the link between the length of the words and v, without having a clue about what the effect is of varying v and μ,...? Or, as Minsky put it in considering the problems involved when studying the example of a tag system Post gave: "*one cannot expect much help from a computer [...] except for clerical aid in studying examples; but if the reader tries to study the behavior of 100100100100100100 without such aid, he will be sorry*" ([Min67], pp. 267–268). Some of the problems related to tag systems can be solved by "pure reasoning", but most of them can only be answered – or even posed – theoretically by first having gone through several "tests".[41] Tag systems simply don't allow for a "direct theoretical intuition" due to their abstractness.

Post indeed tested several cases, varying the parameters, and found three classes

[40] [...] *the problem of "tag" was made the major project of the writer's tenure of a Procter fellowship in mathematics at Princeton during the academic year 1920-1921.* [...] *And the major success of that project was the complete solution of the problem for all bases in which μ and v were both 2.* [...] *this special case $\mu = v = 2$ involved considerable labor.*" ([Pos65], p. 362.). The proof by Post was never published. In Section 9.4.2 we will give the proof, that consists of several classes of cases, and one part of the proof is based on observations of the behaviour of tag systems, using a computer.

[41] Since the notion "experimental" was put between double quotes, the same must be done with the notion "pure reason". To our mind, it seems sometimes as difficult to differentiate an "experimental" approach from some other approach as it is difficult to separate a "pure reasoning approach" from a so-called "experimental" approach.

of behaviour: termination, periodicity and divergence.[42] Divergent behaviour was further divided into two subclasses: tag systems that grow in a predictable way and tag systems that don't ([Pos65], p. 362):

> Where the process does not terminate, it is readily seen that according as the lengths of the resulting sequences are bounded, or unbounded, the resulting infinite sequence of the sequences will, from some point on, become periodic, or the length of the n-th sequence will increase indefinitely with n. In the first case the second form of the problem is again immediately solvable, while in the second case the solution would follow if a method were also found for determining of any given length of sequence a point in the process beyond which all derived sequences were of length greater than that given length.[43]

This last possibility of divergent behaviour caused major difficulties Post was unable to resolve. He furthermore classified classes of cases which are solvable and which might not be solvable. The classes with $\mu = 1$ or $v = 1$ or $v = \mu = 2$ were proven solvable. The case with $\mu = 2$, $v > 2$ he calls intractable, while he terms the cases $\mu > 2, v = 2$ as being of "*bewildering complexity*".

Post had not expected this. Simple though as they may seem, tag systems indeed give rise to intractable and complex behaviour. Even the simple example given above is still not known to be decidable nor universal (despite the availability of the computer). About this example Post remarks in a footnote ([Pos65], p. 363):

> Numerous initial sequences actually tried led in each case to termination or periodicity, usually the latter. It might be noted that an easily derived "probability" prognostication suggested that in this case periodicity was to be expected.[44]

[42] In Post's example the string "010001011" will result in a NILL, while the string "10111011101000000" will lead to periodic behaviour – period 6.

[43] After this description Post added the following footnote: "In this analysis we may have gone somewhat further than is justified by the notes." ([Pos65], p. 362), a comment that shows how much time Post spent on the problem.

[44] It should be remarked here that for small initial conditions, the tag system mentioned by Post indeed always terminates or becomes periodic. Of course he was not able to test larger initial conditions, since then one might have to go through millions of iteration steps before

From this quote it is not only clear that Post indeed tested several cases, e.g., by trying out "numerous initial conditions", but that he even developed a certain probabilistic method to predict the behaviour of the system.
It were his experiences with tag systems that laid the ground for the reversal of Post's program: his goal of finding a positive solution for the finiteness problem for *Principia* seemed hopeless at that time. It is noteworthy that Post's search for the most general form to capture systems of symbolic logic ended up with a form as simple as that of tag systems and that exactly this led him to the mathematical practice of working out special cases, hoping to infer more general properties from these. It were not the theoretical considerations preceding his work on tag systems but his struggle with what appeared to be easy problems that led him to the idea of the finiteness problem possibly being unsolvable [Pos65, p. 363]:[45]

> While considerable effort was expanded on the case $\mu = 2$, $\nu > 2$, but little progress resulted, such a simple basis as $0 \to 00, 1 \to 1101$, $\nu = 3$, proving intractable. For a while the case $\nu = 2$, $\mu > 2$, seemed to be more promising, since it seemed to offer a greater chance of a finely graded series of problems. But when this possibility was explored in the early summer of 1921, it rather led to an overwhelming confusion of classes of cases, with the solution of the corresponding problem depending more and more on problems in ordinary number theory. Since it had been our hope that the known difficulties of number theory would, as it were, be dissolved in the particularities of this more primitive form of mathematics, the solution of the general problem of "tag" appeared hopeless, and with it our entire program of the solution of finiteness problems. This *frustration* [my emphasis], however, was largely based on the assumption that "tag" was but a minor, if essential, stepping stone in this wider program.

As is clear from this quote, these observations really bothered Post – he had not expected difficulties for such "*primitive forms of mathematics*", and it was only when he was able to prove that canonical form A is reducible to a form which is

the system becomes periodic or terminates (if it ever does) – a task which is hardly possible with pencil and paper.

[45] For further arguments concerning this statement, Cfr. Sec. 2.2.4

closely connected to these tag systems, that these difficulties no longer seemed surprising.[46]

2.2.4 Further reductions: From tag systems to Post's thesis

Before Post started his research on tag systems, he had already shown that canonical form *A* can be reduced to canonical form *B* and further proved that the restricted functional calculus is reducible to a system in canonical form *B*. At that time he still believed that the finiteness problem for *Principia* had a positive solution – it was the motivation behind these first reductions ([Pos65], p. 346):

> [...] impetus was lent to the work by our formally reducing the subsystem of Principia Mathematica treated in *10 and *11 thereof to a system of the above type [canonical form *B*]. For thereby a solution of the finiteness problem for all of the above systems would immediately lead to a solution for this important subsystem of Principia Mathematica.

Having understood after his work on tag systems that very simple production systems can give rise to complex behaviour, Post was now led to a whole series of reductions to systems in forms, that are formally far simpler than those in canonical form *A* or *B*, let alone *Principia*.

Systems in canonical form *C*, normal form and the normal form theorem

The first transformation step he made was the reduction of systems in canonical form *B* to a canonical form *C*. Systems in this last form are nowadays known as *Post production systems.* Given a system of which the enunciations, i.e. w.f.f.'s are all expressed in terms of "*finite sequences of letters, the different letters constituting a once-and-for-all given finite set.*" ([Pos65], p. 197) such a system is said to be in canonical form *C*, when the primitive assertions (i.e. the axioms)

[46] As was pointed out to me by Martin Davis, Post hoped that someone else would prove the recursive unsolvability of tag systems. Post himself did not want to work on them again given his frustrating experience. Finally the conjecture that tag systems are unsolvable, was proven by Marvin Minsky in his [Min61], after the problem was suggested to him by Martin Davis.

of the system are a finite set of enunciations in the above form, and the operations performable in the system are specified by a finite set of production rules all of the following form:

$$\begin{gathered}
g_{11}P_{i_1^1}g_{12}P_{i_2^1}\dots g_{1m_1}P_{i_{m_1}^1}g_{1(m_1+1)}\\
g_{21}P_{i_1^2}g_{22}P_{i_2^2}\dots g_{2m_2}P_{i_{m_2}^2}g_{2(m_2+1)}\\
\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\\
g_{k1}P_{i_1^k}g_{k2}P_{i_2^k}\dots g_{km_k}P_{i_{m_k}^k}g_{k(m_k+1)}\\
\textbf{\textit{produce}}\\
g_1P_{i_1}g_2P_{i_2}\dots g_mP_{i_m}g_{(m+1)}
\end{gathered}$$

The g's are given sequences of letters of the once-and-for-all given finite set of letters or alphabet and the P's the operational variables – they can be any combination of letters from this same alphabet. The further restriction is added that each P in the conclusion of the production is present in at least one premise of that production. To understand how such a system in canonical form C works, it may be helpful to give an example. Suppose that the alphabet is given by the set of letters $\{a, b\}$, and for the sake of simplicity, further suppose that there is only one initial assertion, namely:

$$ababbabbabaaaaababbababbbababbbabbb$$

Furthermore, suppose there are two production rules:

$$\begin{gathered}
abaP_1abP_2b\\
aP_3bbP_4b\\
\textbf{\textit{produces}}\\
bbP_1aabbaab
\end{gathered}$$

and

$$\begin{gathered}
bP_{1'}baP_{2'}b\\
bbP_{3'}ab\\
\textbf{\textit{produces}}\\
abaP_{3'}b
\end{gathered}$$

Thus the basis of a system in canonical form *C* has been defined. To be able to produce a new string, one has to check whether the initial assertion “fits” into one of the forms given by the production rules. In this case the first production rule can be applied, but not the second, resulting in the following production:

aba**bb**ab**babaaaaababbababbbbababbbabb**b
a**ba**bb**abbabaaaaababbababbbbababbbabb**b
produces
*bb**bb**aabbaab*

From this new string, another string can be produced by applying the second production rule:

b**bb**ba**abbaa**b
bb**bbaabba**ab
produces
*aba**bbaabba**b*

Constructing systems in this way, makes it possible to generate a variety of systems, which, depending on the production rules and the initial assertion(s), each have their own properties: systems which are monogenic or not[47], systems which produce an infinite or a finite number of strings, systems whose results are periodic or not,… To finish this example, the first strings produced by the basis given here, are shown:

bbbbaabbaab
ababbaabbab
bbbbaaabbaab
ababbaaabbab
bbbbaaaabbaab
ababbaaaabbab
bbbbaaaaabbab
…………………

[47] A monogenic system is a system for which there is for each assertion in the system one and only one applicable production.

Although systems in canonical form C can be interpreted as functional systems, it is more straightforward to regard them as pure string rewriting systems. This is partly due to the fact that "*the boxes within a box symbolic form of the parenthesis notation is replaced merely by finite sequences of letters* [...]"[48]. This results in a more flexible mechanism that becomes even clearer when actually constructing such a system and making it work: the arbitrariness with which such systems can be constructed – manually or programmed – is significant.
A special class of systems in canonical form C consists of systems in normal form. Systems in normal form have only one primitive assertion (axiom) and the finite set of production rules are all of the following form:

$$gP$$

produces

$$Pg'$$

Except for his tag systems, these systems are the most simple and general systems Post ever developed and he used them in several influential papers he wrote in the forties, among them his foundational paper on recursion theory [Pos44]. Tag systems are in fact a special class of systems in normal form. Indeed, the production rules of any tag system can be rewritten in normal form as:

$$a_i wP$$

produces

$$Pw_{a_i}$$

with the length of $w = v - 1$ and w_i being the word corresponding to a_i.
Fundamental to Post's further research was his important normal form theorem, first published in his [Pos43].[49] From this theorem it follows that despite their formal simplicity, systems in normal form are as powerful as systems in

[48] [Pos65], p. 363.

[49] The normal form theorem was first proved in the summer of 1921 and published in his [Pos65]. Marvin Minsky gave a simpler version of the normal form theorem [Min62a, Min67]

canonical forms A, B and C. The theorem states that for every set of assertions generated by a system in canonical form C over a given alphabet Σ, a system in normal form over an alphabet including Σ can be set up, such that the assertions from the system in canonical form are exactly those assertions of the system in normal form which consist of no other letters than those contained in Σ. Of this theorem, Marvin Minsky stated [Min62a], p. 1:

> The theorem proved in this note is the Normal Form Theorem proved in Post's 1943 paper [...] We have long felt that this result is one of the most beautiful in mathematics. The fact that any formal system can be reduced to Post canonical systems with a single axiom and productions of the restricted form $\sigma\alpha \to \alpha\tau$ is in itself a remarkable discovery, and even more so when we learn that this was found in 1921, long before the formalization of metamathematics became so popular.

Given his normal form theorem, Post was led to the actual reversal of his entire program: he formulated a thesis similar to that by Church and Turing stated only 15 years later and proved on the basis of this assumption that the finiteness problem for systems in normal form is unsolvable. But before further discussing these results it is important to argue that it was Post's research on tag systems that actually prepared the ground for this reversal.

The problem of "tag" and the reversal of Post's program

The construction of systems in canonical form C and normal form and the respective reduction proofs, were preceded by Post's work on tag systems.[50] In the limited existing literature discussing Emil Post's early work [Dav82, Dav94, Sti04, Mur98] the fundamental role these tag systems have had in the further development of this work is underestimated if not neglected. There are several arguments that show that tag systems were an essential step towards Post's results on systems in normal form.

First of all, it was through his work on tag systems that he realized that such primitive forms of mathematics can give rise to such bewildering complexity.

[50] As is clear from the chronology of the sequence of events that led to his results Post sketches in the introduction of [Pos65] (p. 341).

On the one hand, this made the positive solution of the finiteness problem appear hopeless at that time (See Sec. 2.2.3). On the other hand the insight that simple forms can be as powerful as *Principia* has been basic to his later results. The fact that Post discusses his tag systems in [Pos43] of which the main purpose is the proof of his normal form theorem, adds further strength to the significance of his work on tag systems for these later reductions.

More convincing are some quotes in which he literally states the dependence of normal form on tag systems. In the last footnote of his [Pos43] Post writes:

> In the summer of 1921, the intervening work on the problem of tag suggested the reduction of canonical form C to the canonical form of the present paper, and this reduction was followed by the successive reductions to normal form essentially as given in section 2.

Although this quote seems ambiguous in that it suggests that there was an intermediary canonical form between canonical form C and normal form, the 'canonical form of the present paper' most probably refers to his normal form. This interpretation relies on the fact that the only two forms described in the paper besides the form of "tag" are canonical form C and normal form.[51] If this interpretation is correct, then this quote indeed illustrates that tag systems led Post to his normal form. The only other sensible interpretation here could be that Post made a typing error: it is possible that it should not be canonical form C, but B. Although this seems less obvious given the fact that the reduction of canonical form B to C is mentioned in this same footnote preceding this quote, the quote interpreted in this way also gives full support to the above mentioned statement: given the fact that canonical form C differs significantly from canonical form B, the fact that tag systems suggested the possibility of this reduction is then fundamental to the formulation of normal form.

A second less ambiguous quote is from Post's [Pos65], p. 382:

> In fact, at one point later in the work on "tag", it seemed that the regularity induced by always removing μ elements from the beginning of a sequence was responsible for the intrusion of number theory in the development, so that it

[51] Normal form being a special case of systems in canonical form C.

> was tentatively suggested that "tag" be generalized to a form, which, indeed, is exactly that of the later derived normal form.

Here Post states that the normal form was suggested due to a property of tag systems which links them to number theory. This is a clear illustration of the significance of tag systems for Post's construction of systems in normal form. Moreover, it was a property that showed the intrusion of number theory in his development. This not only adds support to the significance of tag systems for normal form, but furthermore to their relevance for Post's results on unsolvability.[52] A last quote that almost literally states the fact that tag systems were fundamental to the further reductions from canonical form *B* to *C* to normal form is the following ([Pos65], pp. 202–203):

> Before turning to the proof of our basic theorem given in the next section [i.e. Post's normal form theorem] we wish to [...] state a problem which largely determined the direction taken by the reductions of the next section, and may offer further opportunities for unsolvability proofs. [...] The problem referred to above takes two related forms. Both forms employ the following "tag" operations as we shall call them.

Although being less direct, maybe the most remarkable thing Post notices in this context, is the fact that once he had constructed the reduction from canonical form *A* to normal form, these reductions functioned as a kind of explanation for the difficulties he had experienced with tag systems [Pos65, p. 386]:

> We have observed [...] how the seemingly simple problem of "tag" in fact proved intractable for $\mu = 2, \nu > 2$, of bewildering complexity for $\mu > 2, \nu = 2$. In view of our reduction of canonical form *A* to a form as close to that of "tag" as the normal form, the difficulty of "tag" is no longer surprising.

His work on tag systems was probably the most frustrating research Post did – struggling on the edge of unsolvability. However, they showed him that the

[52] The significance of tag systems for the genesis of the idea that the finiteness problem does have a negative solution is also explicitly stated in the last part of the second quote of section 2.2.3.

simplicity of the basis of a system does not necessarily imply simplicity of the behaviour of the system once it is "run". In the end they were basic for his at that time revolutionary, but unpublished, results.

Post's thesis

Once Post had proven his normal form theorem, he completed his work by "closing the circle" – he showed that systems in normal form are reducible to canonical form *A*. All the forms discussed here (except for the form of "tag") were thus shown to be equivalent to each other. The fact that he had shown that systems in a simple form could produce the same assertions deducible from a more complicated system was now clearly understood. Post had already proven that the part of *Principia* corresponding to first-order predicate calculus could be reduced to a system in canonical form *B*, and that systems in canonical form *B* can be reduced to systems in canonical form *C*. It was the possibility of reducing a seemingly more complicated form to a simpler form that was used as a further argument for the possibility of reducing not only first order predicate calculus but the entire *Principia* to a system in canonical form *B* and thus, through the later reductions, to a normal form [Pos65, p. 384]:

> The power of canonical form *B* was demonstrated [...] by the reduction of *10 *Principia Mathematica* to a single system in that canonical form. From this experience, and the knowledge of the kind of forms and the kind of operations appearing in the whole of *Principia Mathematica*, or could be made to appear if a complete symbolic development thereof were given, it becomes reasonably certain that all of *Principia Mathematica* can in similar fashion be reduced to a system in canonical form *B*. In the absence of the forbidding amount of work needed to actually carry out this reduction, added strength is lent to the above conclusion by the further reductions carried though (...); for if the meager formal apparatus of our final normal systems can wipe out all of the additional vastly greater complexities of canonical form *B*, the more complicated machinery of the latter should clearly be able to handle formulations correspondingly more complicated than itself.

Together with the normal form theorem it followed from these considerations that the whole of *Principia* could be reduced to the simple normal form, indeed a result that can be called remarkable given the time at which it was established. It was this realization that finally led Post to the actual reversal of his entire program.

Notwithstanding the fact that his tag systems had shown him the possibility of certain decision problems being unsolvable it is significant that he still must have had some slight hope for proving the solvability of the finiteness problem for systems in normal form. In reducing such systems to still another form, a solution again seemed within reach ([Pos65], p. 382):

> While [...] special cases of "tag" might well be worth consideration as major problems in themselves, the [...] further reduction of the normal form seemed more promising.

Despite first successes Post soon realized that he wouldn't find a solution to the finiteness problem for systems in normal form. In using this new form for solving the finiteness problem, he was only able to find solutions for a subclass of this form, namely for those systems of which the operations consisted of three of the four operations allowed for in this form. He concluded [Pos65, p. 283]:

> The resulting methods held out the possibility of an attack on the finiteness problem for systems having all four of the above types of operations, though certain of the difficulties of "tag" even then seemed glimmering in the distance. And just when hope was thus renewed for a solution of the general finiteness problem, a fuller realization of the significance of the previous reductions led to a reversal of our entire program.

This "fuller realization" points at what Martin Davis has called Post's thesis [Dav82]. Given the generality of *Principia* it seemed that any method one could think of to generate a set of strings, could be generated through *Principia*. Given the reduction of first order predicate calculus to a system in normal form, by using the normal form theorem, and the above mentioned considerations that convinced Post of the reducibility of the whole of *Principia* to normal form led Post to the conclusion that whatever set we would consider generated can be generated by a system in normal form. This is Post's thesis, and clearly differs from that proposed by Church and Turing. It is significant to note that Post realized that the validity of the thesis did not depend on a mere definition of generated set, but depends on the possibility of a real implementation of an algorithm for generating such a set. In this sense no diagonalization can be done effectively. After having defined a certain set through diagonalization, Post remarks ([Pos65], p. 386):

> [...] in our example we have *merely* defined a set of *a*-sequences [i.e. a certain set defined through diagonalizing out of the class of normal form systems], whereas to yield a true counter-example, we must show how to *generate* that set, i.e., set up a system of "combinatory iteration"

where "combinatory iteration" is described as follows [Pos65, p. 386]:

> [...] the method of combinatory iteration [...] eschews all interpretation, and studies the system merely as a formal system. The operations of the system are then described as "combinatory" since they largely involve but a reshuffling of symbols; and it is through the "iteration", i.e. continued reapplication, of these combinatory operations that the entire system is obtained.

In understanding the computable character of the notion of generated set, identifying it with every possible system of symbolic logic, *Principia* included, the fuller realization of the power of his normal form finally made him, after application of Cantor's diagonal method, conclude that the finiteness problem for the entire class of systems in normal form is unsolvable ([Pos65], p. 386):[53]

> We [...] conclude that the finiteness problem for the class of all normal systems is unsolvable, that is that there is no finite method which would uniformly enable us to tell of an arbitrary normal system and arbitrary sequence on the letters thereof whether that sequence is or is not generated by the operations of the system from the primitive sequence of the system.

[53] For a more detailed explanation of the diagonal argument Post uses see [Dav94, Sti04].

Of course the validity of this assertion really depended on his thesis: it was only the assumption that every generated set of sequences can be obtained from a normal system that enabled this negative conclusion. To Post, his analysis of the notion of generated set however was not yet completed ([Pos65], p. 387):

> The correctness of this result is clearly entirely dependent on the trustworthiness of the analysis leading to the above generalization [...] it is fundamentally weak in its reliance on the logic of Principia Mathematica [...] for full generality a complete analysis would have to be given of all the possible ways in which the human mind could set up finite processes for generating sequences.[...] assuming the correctness of our characterization of generated set of sequences, a mathematical derivation of the unsolvability of the finiteness problem for normal systems as a consequent theorem should be feasible.

The fact that Post understood that a more complete analysis was needed, was one of the reasons why he delayed publication of his work.
Post further realized that his results implied the incompleteness of all systems of finitary symbolic logic reducible to a system in normal form [Pos65, p. 216]:[54]

> Having noted the identity of canonical systems and normal sets [...] our last conclusion was transformed into the generalization that every generated set of sequences on a finite set of letters was a normal set. [...] In the early fall of 1921, the formal proof of this unsolvability [...] was outlined, and led to the further conclusion that not only was every (finitary) symbolic logic incomplete relative to a certain fixed class of propositions (those stating that a given sequence was or was not an assertion in a given normal system) but that every such logic was extendible relative to that class of propositions.

Post had thus anticipated basic results found only 10 years later by Gödel, Church and Turing. Although he had proven the unsolvability of the finiteness problem for systems in normal form, he never proved the Entscheidungsproblem unsolvable. While he had reduced first-order predicate calculus to a system in canonical form *B* and thus indirectly to a system in normal form, he did not

[54] For a more detailed exposition of how Post proved incompleteness on the basis of these results – in comparing this approach to Gödel's – see [Sti04].

prove the reverse reduction. Indeed, at the time he reduced first order calculus to canonical form *B* he was still convinced that its decision problem would be solvable and there was thus no need for a reduction in the other direction (from canonical form *B* to predicate calculus.) In footnote 79 of [Pos65] he explains:

> As to *10 being merely attached to this circle, [an unpublished note] categorically states that a proof of the reducibility of canonical form *A* to *10 is "nearly completed," and as a result even suggests that the solution of the finiteness problem for *10 would yield the solution of the finiteness problem for all of *Principia Mathematica*.

In footnote 90 he further added:

> Less certain, however, is our having paused at the time to realize that the completion of the proof of the reducibility of canonical form *A* to *10 [...] would yield the unsolvability of the latter's finiteness problem. It remains uncertain, therefore, to what extent the writer participated [sic] Church's result on the unsolvability of the deducibility problem for the restricted functional calculus.

2.2.5 Conclusion

In sketching the evolution of Post's ideas starting from his Ph.D. and ending in the fall of 1921, it was shown how the direction of this development was motivated by a method of generalization, reduction to simplified forms and focus on outward form. As was shown, Post was not interested in the development of one particular system of symbolic logic, but instead searched for the most general form and methods to capture any system of symbolic logic. In doing so he wanted to investigate the more general properties of logic, like decidability and completeness, and ultimately mathematics.

His method of generalization and abstraction led him to a form called "Tag" that forced him to a conclusion he had not foreseen: given the complexity and intractability of the several cases of tag systems he considered, his entire program of proving the whole of mathematics solvable, seemed hopeless. His form of "tag" led the way for his at that time revolutionary results playing a significant role in his further reductions to normal form. In the end, his normal form theorem convinced him that it is possible to reduce the seeming more complicated *Principia* to the formally simple normal form and he thus formulated his thesis leading to his results.

As he states in the introduction to his *Account of an Anticipation* [Pos65], it was exactly his focus on the outward forms of symbolic logic rather than on the logical concepts involved that marks the difference between his work and that of his "more complete successors", and allowed for a greater freedom of method and technique. This is clearly exemplified in his reduction proofs. There one sees how step-by-step all meaningfulness is removed. To abstract from the specific meaning of an assertion he first eliminates the meaningful symbols like "$\vee$", and constructs a form which is already then characterized by its abstractness. This became even more explicit once he began his research on tag systems. For example, there is no syntax or order for a specific assertion because of the removal of the delimiters (brackets), all symbols being put on the same level. In that way the concept of a well-formed formula becomes far less important since there are only sequences of letters without any syntax. Every arbitrary combination of letters or symbols from the predefined set of letters or symbols, is well-formed. Furthermore, the concept of an axiom becomes empty since it was only in varying these "axioms" as parameters, together with the parameters of the bases, that Post identified different classes of behaviour and different classes of cases for his tag systems.

Significant in this development, is that through further abstraction, motivated by a difficult theoretical problem, the only "meaningful" method that could be implemented to proceed with these abstractions is a typical mathematical one. In Post's work the technique of studying systems by working out special cases seems to be a consequence of precisely this abstraction process. Since the result of this process were systems in a very simple and "meaningless" form, the methods and results in a system one normally deduces through the interpretation of its axioms, production rules and syntax was not available. There is no specific system, but an infinite group of systems sharing the same form. The only way left to understand the differing properties of such a class of systems is to test them, by checking for different kinds of initial conditions, different values of v keeping for example other parameters constant, testing tag systems with a different number of symbols,.... Only then was it possible to continue the theoretical efforts. This approach then pushed Post's research in a direction he would most probably not have taken without it.

To summarize, although Post was, from the very beginning, searching for the most general form of symbolic logic, it was only in effectively constructing such forms that he was able to conclude for his at that time "unorthodox ideas". In the end, his theoretical assumptions were contradicted by his struggle with what appeared to be easy problems together with the consequent development of his research towards the beautiful systems in normal form. Only then he was able to come to the fuller realization that his systems in normal form are able to generate any set we consider intuitively as being generated. In other words, Post's results on unsolvability were not rooted in a search for a right formalization of a given intuitive notion such as generated set. This idea only emerged *after* he understood that the primitive forms of mathematics he constructed are in fact far from primitive and powerful enough to formalize very general intuitive notions.

2.3 "To deny what seems intuitively natural": Church and the λ-calculus

2.3.1 Introduction

At about the same time Post found his revolutionary results, Church was just starting his career.[55] He arrived at Princeton as an 18-year old boy after having graduated at a preparatory school in Connecticut. In 1924 he graduated with a B.A. in mathematics. After three years he finished his Ph.D. under the supervision of Oswald Veblen. Although, as Martin Davis pointed out in the introduction to Post's *Account of an anticipation* [Dav65a] the field of symbolic logic "*suffered from virtually total neglect in the United States*" Church had not been working as isolated as Post. This "virtually total neglect" of logic in the twenties in the U.S. is affirmed by Church, in an interview with Aspray.[56] When Aspray asked him what kind of textbooks on logic there were around in the 20's, Church answered [Asp84a]:

> There were none that I liked. Lewis and Langford's Symbolic Logic was around. No, that may have been later, but certainly the book by C.I. Lewis was available. But there was nothing about the sort of thing I wanted to teach, logic directed towards math rather than the philosophical aspects of logic. Well, I am not sure; there may have been a book of that sort. Of course [David] Hilbert and Wilhelm Ackermann's Grundzuege der theoretischen Logik was in existence at that time, but it was in German. While the grad students were supposed to learn German, as a practical matter I could not have used it as a textbook. So I used written notes of my own and things like that.

After he had finished his Ph.D. he was awarded a two year National Research Fellowship and spent two years visiting first Harvard, then Göttingen and Amsterdam. In recounting these visits, Church states in the interview with Aspray [Asp84a]:

[55] The biographical information from this introduction mainly comes from [End05] and [Asp84a].

[56] It should be noted that Martin Davis wrote an interesting paper [Dav95] on logic in the twenties in the U.S., focussing on the pioneering work by both Post and Church.

> I had two years on a National Research Fellowship. I spent a year at Harvard and a year in Europe, half the year at Goettingen, because [David] Hilbert was there at the time, and half the year in Amsterdam, because I was interested in [L.E.J.] Brouwer's work, as were some of those advising me.

In the same interview Church remembers taking the train to Brouwer's residence out in the country on several occasions.[57] Afterwards, Church returned to Princeton where he would stay until 1967. In 1967 he left Princeton and went to UCLA where he was Flint professor of Philosophy and Mathematics until 1990, when he retired at the age of 87.

Once the thirties started, Princeton became "the place to be" for many logicians. Church was now surrounded by his two famous Ph.D. students Barkley Rosser and Stephen Kleene. John von Neumann was there, and furthermore Gödel crossed the ocean. From 1933 to 1934 Gödel visited the Institute for Advanced Study, where he gave his important lectures[58] attended by Church, Kleene and Rosser. In other words, the situation Church was working in before he published his important 1936 results can hardly be compared to that of Post. Not only did he have a rather luxurious position as compared to Post's but he was surrounded by several other logicians, so he was able to exchange ideas with his colleagues.

Church's contributions to logic and the foundations of mathematics cannot be underestimated. Besides his research results, he was one of the principal founders of the Association for Symbolic Logic, the publisher of *The Journal of Symbolic Logic* (*JSL*). Starting from the publication of the first volume of this well-known journal, Church was not only an editor for contributed papers from 1936 to 1950 but was also the editor of the review section (1936–1979). He wrote many, often harsh and severe, reviews. His goal of the review section was twofold. In 1936 in the last number of the first volume of the JSL, Church had published an almost 100 pages long *Bibliography of Symbolic Logic* [Chu36a] covering the period 1666–1935. One of the goals of the review section was to

[57] In answering who he met in Amsterdam, he says that he didn't meet with Heyting then. [End05] mentions that Church met amongst others Bernays and Heyting during these years, but this happend rather in Göttingen.

[58] Published as [Göd34]

further extend and update this bibliography, for books and journals published after 1935. Furthermore, its purpose was to provide commentary to the literature where necessary ([Chu36b], p. 42:[59]

> It is intended that this section of the Journal shall serve as a complete bibliography of current literature in the field of symbolic logic, from January 1, 1936. To this end an effort will be made to include in it, at least by title, all publications in this field, both books and articles in journals, and as far as possible these will be accompanied by signed reviews.

Church's work not only influenced the domain of logic and mathematics, but it inspired important contributions to computer science as well. Of course, Church is best-known because of Church's thesis, providing a formal definition of the intuitive notion of effective calculability. In collaboration with Kleene and Rosser, one of Church's most important achievements has been the development of λ-calculus. This calculus has influenced the domain of computer science in many different respects. First of all, it lies at the basis of the oldest functional programming language LISP, developed by John McCarthy and has, as a theoretical functional programming language, a general significance for the theory of programming.[60] Together with Curry's [Cur30] and Schönfinkel's [Sch24] systems, the λ-calculus is furthermore one of the famous examples of combinatory logic.[61] The λ-calculus nowadays also plays an important role in the implementation on computers of systems to do mathematics, called computer mathematics. These systems are used to formalize and verify proofs by

[59] The paper [End98] discusses Church's role in the review section of the JSL.

[60] LISP was first described in [McC60]. A paper by McCarthy on the history of Lisp is [McC81]. In recounting the innovative character of LISP, McCarthy says about the influence of the λ-calculus: "*To use functions as arguments, one needs a notation for functions, and it seemed natural to use the λ-notation of Church [Chu41]. I didn't understand the rest of the book, so I wasn't tempted to try to implement his more general mechanism for defining functions.*" ([McC81], p. 176.) A very interesting paper discussing the influence of λ-calculus on functional programming languages is [Tra88]. Rosser's [Ros82, Ros84] further discusses the role of λ-calculus and in general of combinatory logic, for computer languages.

[61] Rosser's Ph.D. thesis, published as [Ros35] made clear the connection between Curry's and Schönfinkel's combinatory logics and the λ-calculus.

computers.[62]

Alonzo Church died at the age of 93 on August 11, 1995.

In this section we will discuss Church's work preceding the first announcement of his famous results on 19 April, 1935 to the American Mathematical Society [Chu35]. We will show how he was led to the formulation of his thesis and the proofs of certain unsolvable decision problems, starting from the first published paper by Church in 1924 on the Lorentz transformation. Since the period between the publication of this paper and Church's famous results is well-documented and marked by a continuity of published work, we are able to sketch the evolution of Church's ideas from 1924 till April 1935 without large interruptions, contrary to our analysis of Post's earlier work.

2.3.2 Towards variant systems of logic.

As Church tells in an interview with Aspray [Asp84a], he was already interested in foundational issues as an undergraduate. His first published paper was on the Lorentz transformation [Chu24], which is at the foundations of (special) relativity. The object of this paper was to find a set of logically independent postulates that uniquely determine the Lorentz transformation for one dimension. One year later he published a more general paper [Chu25] further exploring the concept of independent sets, in relation to irredundant sets of postulates.[63]
After graduating, he started his Ph.D. at Princeton in 1924 under Oswald Veblen, who was interested in the foundations of mathematics and thus sharpened Church's general interest in the subject. He even urged him to read some of Hilbert's work:

[62] A paper surveying the influence of λ-calculus on logic and computer science, including its significance for computer mathematics is [Bar97].

[63] It is interesting to note that in describing a method by which any set of independent postulates can be made irredundant, Church identifies it as a "mechanical method" ("*There is a mechanical method by which any set of postulates can be made irredundant.*"([Chu25], p. 321).

> It was Veblen who urged me to study Hilbert's work on the plea, which may or may not have been fully correct, that he himself did not understand it and he wished me to explain it to him. At any rate, I tried reading Hilbert. Only his papers published in mathematical periodicals were available at the time. Anybody who has tried those knows they are very hard reading. I did not read as much of them as I should have, but at least I got started that way.

Veblen was also interested in the question if the axiom of choice was independent of other axioms, as Church remarks in the same interview, and it became the subject of his dissertation, published as [Chu27]. As is clear from its title, *Alternatives to Zermelo's assumption*, p. 178:

> The object of this paper is to consider the possibility of setting up a logic in which the axiom of choice is false.

In his Ph.D., Church indeed started from the 'hypothesis' that the axiom of choice could be considered independent from Zermelo-Fraenkel set theory. In making this assumption, he wanted to investigate the possible consequences of several alternatives to Zermelo's 'assumption'. Church was well aware of the fact that replacing the axiom of choice by contradictory assumptions was not evident at that time.[64] He even had to convince his supervisor Veblen, due to the fact that he wanted to contradict that which seemed intuitively more natural. After having explained in the interview with Aspray that he regarded his dissertation more as research in mathematics rather than in logic, Church explains ([Asp84a]):

> The only thing that might have annoyed some mathematicians was the presumption of assuming that maybe the axiom of choice could fail, and that we should look into contrary assumptions.[...] [Veblen] was really the only man supervising it. I sort of had to convince him about some aspects of the axiom of choice. To deny what seems intuitively natural is rather difficult. You tend to slip back into what informally seems more reasonable. I remember from time to time having to explain things to him, but I convinced him that my arguments were sound.

[64] In [Dav95] it is discussed in more detail that at the time, Church's general approach of exploring variant systems of symbolic logic was indeed far from evident.

As is noted by Martin Davis ([Dav95], p. 275):

> [Church] was acutely aware of set theory together with logic as a foundation of mathematics [...] [while] [n]oteworthy contributions to logic and foundations of mathematics were few and far between during the twenties.

His research was indeed explicitly embedded in the context of the foundations of mathematics. Before actually starting with his investigation into the alternatives to Zermelo's assumption, he discusses the problem of completeness "*to prepare the way for the suggestion that there may be one or more additional independent postulates which can be added to the set of postulates 1-5* [...]".[65]
Church considers three main postulates A, B and C wanting to "*inquire into their character, and to derive as many of their consequences*"[66] in order to find reasonable alternatives for the axiom of choice.
Significant is the fact that Church considers the derivation of as many consequences as possible a valuable way to argue for the independence of the axiom of choice. If one of the postulates would involve a contradiction, this process of deriving as many consequences as possible, should reveal it at a given time.[67] If not, this fact can be regarded as "presumptive evidence" for the independence of the axiom of choice ([Chu27], p. 187):

> If any one of these involve a contradiction it is reasonable to expect that a systematic examination of its properties will ultimately reveal this contradiction. But if a considerable body of theory can be developed on the basis of one of these postulates without obtaining inconsistent results, then this body of theory, when developed, could be used as presumptive evidence that no contradiction exists. If there be two of these postulates neither of which leads to contradiction, then there are corresponding to them two distinct self-consistent second ordinal classes, just as Euclidian and Lobachevskian geometry are distinct self-consistent geometries [...]

[65] [Chu27], p.186

[66] [Chu27], p.178

[67] This approach is very similar to, e.g., J.H. Lambert's work (published posthumously 1786) on the parallel postulate that precedes later work by Gauss, Bolyai and Lobachevski (See [Lam86, SE95]).

Starting from the idea that if a set of postulates is inconsistent, a systematic examination of its properties should ultimately reveal a contradiction, Church concludes that if one is able to develop a *considerable* amount of theory starting from the assumption, without finding a contradiction, one can *presumptively* conclude that the theory developed might be consistent. This evidence in its turn then adds strength to the hypothesis of the independence of the axiom of choice.

Later on in his dissertation, Church identifies this attitude as an experimental one. After having deduced many of the consequences of the three postulates, Church proposes two more postulates F and G, inconsistent with each other, but "apparently consistent" with postulates 1-5 and C. After the statement of the postulates, Church announces how he wants to proceed ([Chu27], p. 205):

> We shall examine briefly the consequences of each of the postulates just stated when taken in conjunction with Postulates 1-5 and C, taking the same experimental attitude as that which we took in the case of Postulates A, B and C.

One year later another paper by Church was published *On the law of the excluded middle* [Chu28] of which the purpose is clearly in line with the ideas sketched in his Ph.D.:

> [The purpose of this paper is] to discuss the possibility of a system of logic in which the law of the excluded middle is not assumed [...]

Again it is clear that Church was not interested in the study of one ultimate system of logic. On the contrary, he wanted to consider variant systems of symbolic logic, the underlying idea being that there is not one absolute system of logic.

If one no longer holds on to the idea of preferring one set of axioms over another one, because it is closer to intuition, i.e. if one wants to deny what seems intuitively natural, one has to find other ways to evaluate the different systems one is considering. In Sec. 2.1 we already discussed that for Hilbert this new criterium was the system's consistency. Also for Church, already in these earlier papers, consistency is the main criterium to judge a given system of symbolic logic. Proving the consistency of a given system can be very hard. It can take years before a proof is found, if ever. Church must have been aware of the problems that might be involved in proving a system consistent since he was familiar with Hilbert's work and the German language. His "empirical" attitude towards a systems' consistency is thus very reasonable and nowadays shared by several other mathematicians. As, e.g., Martin Davis remarked in discussing the problem of consistency proofs [Dav90]:

> [...] great logicians (Frege, Curry, Church, Quine, Rosser) have managed to propose quite serious systems of logic which later have turned out to be inconsistent. "Insight" didn't help. New axioms are just as problematical as new physical theories, and their eventual acceptance is on not dissimilar grounds.

This point of view became only more explicit in the results to follows.
Four years after the publication of [Chu28] Church published the first of two major papers in which the ideas and methods already present in his earlier work become even more apparent. It were these papers which finally led to λ-calculus and ultimately Church's thesis.

2.3.3 An Inconsistent Set of Postulates

In [Chu32] Church developed a system of postulates to serve as a foundation for logic and mathematics – a system of logic adequate for the development of mathematics in which the notion of a function plays a fundamental role. This set of postulates however had to be "*free of some of the complications entailed by Bertrand Russell's theory of types, and* [at the same time had to avoid] *the well known paradoxes* [...]"[68].
Basic to Church's set of postulates and in fact to the subsystem included therein now known as the λ-calculus, is the notion of a function. As he states in the introduction of his nice little orange book published in 1941 and still known as a good introduction to λ-calculus, called *The calculi of lambda-conversion* [Chu41], p. 1:

> Underlying the formal calculi which we shall develop is the concept of a function, as it appears in various branches of mathematics [...]. The study of the gen-

[68] [Chu33], p. 839

> eral properties of functions, independently of their appearance in any particular mathematical (or other) domain, belongs to formal logic, or lies on the boundary line between logic and mathematics. This study is the original motivation for the calculi [...]

Church's set of postulates was thus developed to study the properties of functions, independently of their appearance in a specific domain. As is also pointed out in Kleene's excellent paper on the history of recursive functions and the λ-calculus [Kle81a], one of the main ingredients of Church's set of postulates is that he got rid of the ambiguous use of expressions that can either denote a function or an expression, containing a variable, that ambiguously denotes a value of the function. Church gives the following example of such an expression: $(x^2+x)^2$. He proceeds [Chu41], p. 6:

> If we say "$(x^2+x)^2$ is greater than 1,000, " we make a statement which depends on x and actually has no meaning unless x is determined as some particular natural number. On the other hand, if we say "$(x^2+x)^2$ is a primitive recursive function," we make a definite statement whose meaning in no way depends on a determination of the variable x (so that in this case x plays the role of an apparent, or bound, variable).

This ambiguity was resolved by using the so-called *abstraction operator* λ. In the example, the ambiguity is banished by using $\lambda x[(x^2+x)^2]$ as notation, x now being bound by λ. In this respect, Church's set of postulates abandoned ambiguous uses of the free variable, the reason being that he required ([Chu32], p. 346):

> [...] that every combination of symbols belonging to our system, if it represents a proposition at all, shall represent a particular proposition, unambiguously, and without the addition of verbal explanations.

Church's set of postulates thus had the ambition to provide foundations for logic and mathematics in which the notion of a function plays a basic role.

Unlike the authors of *Principia*, Church did not claim any absoluteness for his proposed set of postulates, an attitude clearly inspired by his former work ([Chu32], p. 348):

> We do not attach any character of uniqueness or absolute truth to any particular system of logic.

While he did not give explicit reasons for this kind of attitude towards logic in his former work, Church now adds strength to his approach by making statements about the connections between an abstract theory and the reasons why it is developed – its 'application'. In this context he links up, by analogy, the existence of alternative geometries with the existence of alternative systems of symbolic logic ([Chu32], 348–349):

> The entities of formal logic are abstractions, invented because of their use in describing and systematizing facts of experience or observation, and their properties, determined in rough outline by this intended use, depend for their exact character on the arbitrary choice of the inventor. We may draw the analogy of a three dimensional geometry used in describing physical space [...] In building the geometry, the proposed application to physical space serves as a rough guide in determining what properties the abstract entities shall have, but does not assign these properties completely. Consequently there may be, and actually are, more than one geometry whose use is feasible in describing physical space. Similarly, there exist, undoubtedly, more than one formal system whose use as a logic is feasible, and of these systems one may be more pleasing or more convenient than another, but it cannot be said that one is right and the other wrong.

Indeed, the fact that any system of formal logic is determined by its use in order to describe certain experiences and observations, implies that there cannot be one ultimate system of logic. This does not mean that the logic is completely determined by its application. On the contrary ([Chu32], p. 349):

> In consequence of this abstract character of the system which we are about to formulate, it is not admissible, in proving theorems of the system, to make use of the meaning of any of the symbols, although in the application which is intended the symbols do acquire meanings. The initial set of postulates must of themselves define the system as a formal structure, and in developing this formal structure *reference to the proposed application must be held irrelevant.* [m.i.] There may, indeed, be other applications of the system than its use as a logic.

As was said, given this attitude towards variant systems of logic, there remained for Church only one criterion to reject or accept (be it on a presumptive basis) a given system of logic: its consistency. Given the non-existence of a general method to prove consistency the only reasonable attitude left to apply this criterion to a given system of logic, is an 'empirical' one, for as long as no consistency proof is found ([Chu32], p. 348):

> Whether the system of logic which results from our postulates is adequate for the development of mathematics, and whether it is wholly free from contradiction, are questions which we cannot answer except by conjecture. Our proposal is to seek at least an empirical answer to these questions by carrying out in some detail a derivation of the consequences of our postulates, and it is hoped either that the system will turn out to satisfy the conditions of adequacy and freedom from contradiction or that it can be made to do so by modifications or additions.

This attitude is repeated by Church in a reply to a letter to Gödel, dated july 27, 1932.[69] In answering the question posed by Gödel of whether there is any other way to prove the consistency of Church's set of postulates besides proving it consistent relative to type or set theory, Church answers:[70]

> In fact, the only evidence for the freedom from contradiction of *Principia Mathematica* is the empirical evidence arising from the fact that the system has been in use for some time, many of its consequences have been drawn, and no one has found a contradiction. If my system be really free from contradiction, then an equal amount of work in deriving its consequences should provide an equal

[69] It should be noted here that Church later admitted that he was among those at that time who believed that "*Gödel's incompleteness theorem might be found to depend on peculiarities of type theory [...] in a way that would show this results to have less universal significance than he was claiming for them.*" (Church in a letter to John Dawson, dated July 25, 1983, reprinted in [Sie97]). This is already clear from this reply to Gödel, in which he states amongst other things, that he "*has been unable to see, however, that your conclusions in* §*4 [Gödel's second incompleteness theorem] of this paper apply to my system.*", [Göd03a], p. 369

[70] The exact question posed by Gödel is: "*In case the system is consistent, won't it then be possible to interpret the fundamental concepts in a system with type theory, or in the axiom system of set theory, and can one make the consistency plausible at all in any other way than through such an interpretation?*" , 17 June, 1932, [Göd03a], p. 367

> weight of empirical evidence for its freedom from contradiction.([Göd03a], p. 368)

This 'empirical' attitude was further pursued in [Chu33]. Having learned in the meantime that some of his postulates lead to a contradiction, the list was revised. Furthermore 42 new theorems were proven to follow from this new set of postulates and a basis to develop a theory of positive integers in the set of postulates was added. In this paper Church beautifully summarizes his 'empirical' approach to logic and mathematics ([Chu33], p. 842):

> Our present project is to develop the consequences of the foregoing set of postulates, until a contradiction is obtained from them, or until the development has been carried so far consistently as to make it empirically probable that no contradiction can be obtained from them. And in this connection it is to be remembered that just such empirical evidence, although admittedly inconclusive, is the only existing evidence of the freedom from contradiction of any system of mathematical logic which has a claim to adequacy.

However, soon after the publication of this paper it would be shown by Kleene and Rosser – Church's Ph.D. students – that he had not inferred enough consequences out of the system: they showed that Church's set of postulates is inconsistent [KR35][71] – a result that clearly illustrates the problematic character of Church's, or any other, 'empirical' attitude, and "*not exactly what one dreams of having one's graduate students accomplish*" as Martin Davis stated ([Dav82], p. 4).

2.3.4 λ - The Ultimate Operator

In the meantime Kleene's attention had shifted to a subpart of Church's set of postulates, now known as the λ-calculus. He was working on his Ph.D. replying to the program Church proposed at the end of [Chu33] , p. 864:

> Our program is to develop the theory of positive integers on the basis which we have just been describing, and then, by known methods or appropriate modifi-

[71] The proof itself is from early 1934

> cations of them, to proceed to a theory of rational numbers and a theory of real numbers.

Kleene's original Ph.D. topic was indeed to develop a theory of positive integers in Church's set of postulates.[72] It was published in two parts in 1935 ([Kle35a] [Kle35b]) and contained the development of such a theory in the λ-calculus.[73]

A quick introduction to λ-calculus.[74] In λ-calculus there are two types of symbols. The three primitive symbols λ, (,) also called the improper symbols by Church, and an infinite list of variables. There are three rules to define the well-formed formulas of λ-calculus, called λ-formulas.

1. The λ-formulas are first of all the variables themselves.
2. If P is a λ-formula already constructed, containing x as a free variable then $\lambda x[P]$ is also a λ-formula. The λ-operator is used to bind variables and it thus converts an expression containing free variables into one that denotes a function (cfr. supra, Sec. 2.3.3).[75]
3. If M and N are λ-formulas then so is {M}(N), where {M}(N) is to be understood as the application of the function M to N.

The λ-formulas, or well-formed formulas of λ-calculus are all and only those formulas that results by (repeated) application of these three rules. There are three operations or rules of conversion. Let us define $\mathrm{S}_{\mathrm{N}}^{x}\mathrm{M}|$ as standing for the formula that results by substitution of N for x in M. For each of the rules we will give one explanatory example. the expression we will use are not in the pure language of λ-calculus and are merely added to make clear the rules.

[72] In [Asp84b], Kleene says: "*Church, in the last paragraph or the last page of his second paper on the foundation of logic, proposed the problem of developing the theory of positive integers on the basis of his system. There was a ready-made Ph.D. thesis problem. With my very limited knowledge of the area at that time, I don't think I could have dreamed up a problem for myself. It proved to be a challenging problem, and I did it.*"

[73] After Kleene and Rosser had shown that Church's set of postulates was inconsistent, Kleene rewrote his dissertation taking into account this result, although his Ph.D. had already been accepted in September 1933.

[74] This exposition is based on [Chu41] and [Kle81a].

[75] "*We think of $\lambda x[P]$ as denoting that function of x whose value (if defined), for each value taken by x, is the value then taken by P.*" [Kle81a], p. 54

1. *Reduction.* To replace any part $((\lambda x \text{ M}) \text{ N})$ of a formula by $\text{S}^x_\text{N}\text{M}|$ provided that the bound variables of M are distinct both from x and from the free variables of N. For example to change $\{\lambda x[x^2]\}(2)$ reduces to 2^2
2. *Expansion* To replace any part $\text{S}^x_\text{N}\text{M}|$ of a formula by $((\lambda x \text{ M}) \text{ N})$ provided that $((\lambda x \text{ M}) \text{ N})$ is well-formed and the bound variables of M are distinct both from x and from the free variables in N. For example, 2^2 can be expanded to $\{\lambda x[x^2]\}(2)$
3. *Change of bound variable* To replace any part M of a formula by $\text{S}^x_\text{y}\text{M}|$ provided that x is not a free variable of M and y does not occur in M. For example changing $\{\lambda x[x^2]\}$ to $\{\lambda y[y^2]\}$

Church then introduced an encoding for the natural numbers, where he considered the numeral in Arabic notation as abbreviations for an infinite set of λ-formulas. I.e., he gave the following definitions:

$$\begin{aligned}
&1 \rightarrow \lambda yx.yx,\\
&2 \rightarrow \lambda yx.y(yx),\\
&3 \rightarrow \lambda yx.y(y(yx)),\\
&\ldots
\end{aligned}$$

where it should be noted that the λ-definition of the natural numbers uses not the original notation of λ-calculus, but an abbreviated notation using, amongst others dots as brackets as a kind of shorthand. Using these definitions of the natural numbers it is possible to *λ-define* functions over the positive integers. A function F of one positive integer is λ-definable if we can find a λ-formula F, such that if $F(m) = n$ and m and n are λ-formulas encoding the integers m and n (according to the above given encoding scheme), then the λ-formula {F} (m) can be *converted* to r by applying the conversion rules of λ-calculus. Thus, for example the successor function S, first introduced by Church, can be λ-defined as follows:

$$S \rightarrow \lambda abc.b(abc)$$

To give an example, applying S to the λ-formula standing for 2, we get:

$$\begin{aligned}
&\big(\lambda abc.b(abc)\big)\big(\lambda yx.y(yx)\big) \rightarrow \lambda bc.b\big(\big(\lambda yx.y(yx)\big)bc\big)\\
&\rightarrow \lambda bc.b\big(\big(\lambda x.b(bx)\big)c\big) \rightarrow \lambda bc.b(b(bc))
\end{aligned}$$

It is also important here to explain the normal form in λ-calculus. A formula is said to be in *normal form* if it is well-formed and contains no part of the form $\{\lambda x[\text{M}]\}(\text{N})$. A formula is said to be in *principal normal form* if it is in normal form and no variable occurs in it both as a free and as a bound variable, and the

> variables which occur in it immediately following the symbol λ are, when taken in the order in which they occur in the formula, in natural order, without repetitions, beginning with a and omitting only such variables as occur in the formula as free variables. For example the formula $\lambda ab.b(a)$ is in principal normal form, $\lambda ac.c(a)$ is in normal form but not in principal normal form.

Already from the first paragraph of his dissertation it is clear what Kleene had learned, or at least inherited, from Church ([Kle35a], p. 153):

> Our object is to demonstrate empirically that the system is adequate for the theory of positive integers, by exhibiting a construction of a significant portion of the theory within the system. By carrying out the construction on the basis of a certain subset of Church's formal axioms, we show that this portion at least of the theory of positive integers can be deduced from logic without the use of the notions of *negation*, *class*, and *description*.

While Church's empirical approach *might* have been disappointing when his set of postulates turned out to be inconsistent, it would show very fruitful during further research on the λ-calculus.

As is stated by Rosser in his [Ros84], Church first mentioned (or even had) the idea that every effectively calculable function from positive integers is λ-definable in a conversation in late 1933, after Rosser had told him about his latest function in λ-calculus ([Ros84], p. 345):[76]

> One time, in late 1933, I was telling him [Church] about my latest function in the LC. He remarked that perhaps every effectively calculable function from positive integers to positive integers is definable in LC. He did not say it with any firm conviction. Indeed, I had the impression that it had just come into his mind from hearing about my latest function. With the results of Kleene's thesis and the investigations I had been making that fall, I did not see how Church's suggestion could possibly fail to be true. in fact, I immediately berated myself (silently) for

[76] A more detailed account of the events preceding the first official statement of Church's thesis can be found in [Kle81a, Ros84, Dav82, Sie97]

> not having seen the obvious a month or two before, so that I would have made that proposition to Church before he made it to me.

According to both Rosser and Kleene, Church was convinced about the equivalence between λ-definability and effective calculability in early 1934.[77] This idea however was far from evident given the, at first, counterintuitive way one can compute functions in λ-calculus ([Kle81a], p. 54):

> Before research was done, no one guessed the richness of this subsystem. Who would have guessed that this formulation, generated as I have described to clarify the notation for functions, has implicit in it the notion (not known in mathematics in 1931 in a precise version) of all functions on the positive integers (or on the natural numbers) for which there are algorithms?

Indeed, Kleene himself had not expected that λ-calculus would have been so powerful, and it was not he nor Rosser but Church who first came up with this idea of identifying λ-definability with calculability. As is told by Barendregt ([Bar97], p. 186):

> Many years later – it was at the occasion of Robin Gandy's 70-th birthday, I believe – I heard Kleene say: "I would like to be able to say that, at the moment of discovering how to lambda define the predecessor function, I got the idea of Church's thesis. But I did not, Church did."

A very important trigger for Church's idea was indeed Kleene's definition of the predecessor function in λ-calculus ([Kle81a] p. 57):[78]

> When I brought this result to Church, he told me that he had just about convinced himself that there is no λ-definition of the predecessor function. The discovery that the predecessor function is after all λ-definable excited our interest in what functions are not just definable in the full system but actually λ-definable. The exploration of this became a major subproject for my Ph.D. thesis. Of course,

[77] The exact time at which Church made a more definite proposal of his thesis should be situated between February 7, 1934 and March 1934. See [Dav82], p. 8

[78] In [Kle81a] he explains that he got the idea of how to λ-define the predecessor function at the dentist in late January or early in February 1932

> I did develop a great deal of theory of positive integers in Church's formalism, using many λ-definitions in the process.

From that moment on, the search for effectively calculable functions which are λ-definable became a more explicit research goal. Kleene gradually unravelled the amazing computational power of the λ-calculus, in being able to show that each example of an effective calculable function he and Church could think of, was indeed λ-definable ([Kle81a] p. 57):

> We [Church and Kleene] kept thinking of specific such functions, and of specific operations for proceeding from such functions to others. I kept establishing the functions to be λ-definable and the operations to preserve λ-definability.

However, as was stated before, it was not Kleene but Church who first thought about an explicit identification between λ-calculus and effective calculability. In fact, when Church first proposed his 'thesis' to Kleene ([Kle81a], p. 59):

> [I, Kleene] sat down to disprove it by diagonalizing out of the class of the λ-definable functions. But, quickly realizing that the diagonalization cannot be done effectively, I became overnight a supporter of the thesis.

Significant is the fact that it was not "*the concept [of λ-definability] itself but rather [the] results established about it*" ([Kle81b], p. 49) that led Church to his 'conjecture'. As is pointed out by Sieg [Sie97], the main reason for proposing the identification was, what Sieg calls, the 'quasi-empirical' fact expressed by Church in a letter to Bernays, dated January 23, 1935 (Quoted in [Sie97], p. 155):

> The most important results of Kleene's thesis concern the problem of finding a formula to represent a given intuitively defined function of positive integers (it is required that the formula shall contain no other symbol than λ, variables, and parentheses). The results of Kleene are so general and the possibilities of extending them apparently so unlimited that one is led to the conjecture that a formula can be found to represent any particular constructively defined function of positive integers whatever.

Although several sources report that Church had already formulated his thesis in terms of λ-definability in early 1934 he only communicated the result in April

1935. Not in terms of λ-definability however, but in terms of Herbrand-Gödel *general recursiveness.*

Indeed, in the meantime, Gödel had already published his seminal 1931-paper containing the two incompleteness theorems [Göd31] and, after a suggestion by Herbrand,[79] extended the notion of primitive recursiveness to general recursiveness. From February to May 1934 Gödel gave a series of lectures attended by Kleene, Rosser and Church.[80] As is discussed by Davis [Dav82], given the definition of general recursiveness and a footnote added in [Göd34], the notes from these lectures suggest that Gödel formulated a thesis similar to Church's. [81] When Davis was preparing his [Dav65b] he submitted his first draft of the introduction to the lecture notes to Gödel for his comments, suggesting that Gödel had indeed stated such a thesis similar to Church's during his lectures. As Davis writes ([Dav82], p. 8), "Gödel took strong exception to my suggestion", as is clear from his reply (Quoted in [Dav82], p. 8):

> [...] it is *not true* that footnote 3 is a statement of Church's Thesis. The conjecture stated there only refers to the equivalence of "finite (computation) procedure" and "recursive procedure." However, I was, at the time of these lectures, not at all convinced that my concept of recursion comprises all possible recursions [...]

In a letter to Kleene dated November 29, 1935 Church gave an account of a discussion on effective calculability with Gödel, presumably to be situated in early 1934. Kleene supplied a copy of the letter to Martin Davis who quoted it in his [Dav82], p. 9:

> In regard to Gödel and the notions of recursiveness and effective calculability, the history is the following. In discussion with him the notion of lambda-definability,

[79] See [Sie05] for a detailed account of the correspondence between Gödel and Herbrand.

[80] A series of lecture notes taken by Rosser and Kleene have been preserved and published in a corrected and amplified version in [Dav65b].

[81] After having noted in the main text that primitive recursive functions "*have the important property that, for each given set of values of the arguments, the value of the function can be computed by a finite procedure*" the following footnote was added: "*The converse seems to be true, if, besides [primitive] recursions [...] recursions of other forms (e.g., with respect to two variables simultaneously) are admitted. This cannot be proved, since the notion of finite computation is not defined, but it serves as a heuristic principle.*" [Göd34], p. 44

> it developed that there was no good definition of effective calculability. My proposal that lambda-definability be taken as a definition of it he regarded as thoroughly unsatisfactory. I replied that if he would propose any definition of effective calculability which seemed even partially satisfactory I would undertake to prove that it was included in lambda-definability. His only idea at the time was that it might be possible, in terms of effective calculability as an undefined notion, to state a set of axioms which would embody the generally accepted properties of this notion, and to do something on that basis. Evidently, it occurred to him later that Herbrand's definition of recursiveness [...] could be modified in the direction of effective calculability, and he made this proposal in his lectures. At that time he did specifically raise the question of the connection between recursiveness in this new sense and effective calculability, but said he did not think that the two ideas could be satisfactorily identified "except heuristically".[82]

Gödel thus regarded Church's proposal as "thoroughly unsatisfactory" and was convinced that recursiveness cannot be identified with computability, "except heuristically".[83]

Despite Gödel's criticism, who was at that time already a respected authority given his [Göd31], Church publicly announced his thesis in a talk to the American Mathematical Society, 19 April, 1935. Not in terms of λ-definability though, but in terms of general recursiveness. As he writes in the abstract of the talk [Chu35], submitted 22 March, 1935:

> [...] it is maintained that the notion of an effectively calculable function of positive integers should be identified with that of a recursive function, since other plausible definitions of effective calculability turn out to yield notions which are either equivalent to or weaker than recursiveness.

As is clear from this quote, λ-definability has been completely replaced by recursiveness, so one wonders why Church made this substitution and why he waited about one year to publicly announce this variant of the thesis he had

[82] A shorter excerpt of this letter was also published by Kleene [Kle81a].

[83] Davis' [Dav82, Dav05] gives a more detailed account of Gödel's opinion in this context, and its evolution over time.

talked about with Kleene, Rosser and Gödel already in early 1934.
According to Davis, the fact that Church only implicitly refers to λ-definability is due to his being uncertain at that time about the equivalence between λ-definability and general recursiveness.[84] But Davis does not provide a real explanation for the fact that Church waited so long before publicly announcing his thesis, and, especially, the fact that λ-definability was now replaced by recursiveness. Still, given Gödel's reluctance to accept Church's thesis, while Church already proposed his thesis informally in 1934 and publicly (in its variant version) in 1935, Davis concludes for a clear contrast between both logicians [Dav82], p. 12–13:

> [...] Gödel was not convinced by the available evidence, and remained unwilling to endorse the equivalence of effective calculability, either with recursiveness or with λ-definability. [...] Thus while Gödel hung back because of his reluctance to accept the *evidence* for Church's thesis available in 1935 as decisive, Church (who after all was right) was willing to go ahead, and thereby to launch the field of recursive function theory.

In [Sie97], Sieg provides his interpretation of the fact that Church waited so long before publicly announcing his thesis, now stated in terms of recursiveness instead of λ-definability, and concludes that Davis's interpretation is not completely correct. Basic in the argumentation supporting Sieg's interpretation, is the fact that, according to him, the equivalence between general recursiveness and λ-definability had already been established before March 1935 when Church submitted his abstract. Sieg uses this to argue that "*Church's and Gödel's developed views actually turn out to be much closer than [their] early opposition might lead one to suspect.*" ([Sie97], p. 157) and thus criticizes Davis's account. Sieg then explains Church's so-called reluctance by the fact that Church himself was not completely convinced of his λ-calculus as being a good identification for calculability. In arguing that the equivalence between

[84] "*It is interesting that λ-definability occurs only by implication in the reference to "other plausible definitions of effective calculability ... either equivalent to or weaker than recursiveness." The wording leaves the impression that in the early spring of 1935 Church was not yet certain that λ-definability and Herbrand-Gödel general recursiveness were equivalent.*" ([Dav82], p. 10)

λ-definability and recursiveness was already established before Church submitted his abstract, Sieg claims ([Sie97], p. 157):

> I claim, and will support through the subsequent considerations, that Church was reluctant to put forward the thesis in writing – until the equivalence of λ-definability and general recursiveness had been established. The fact that the thesis was formulated in terms of recursiveness indicates also that λ-definability was at first, even by Church, not viewed as one among equally natural definitions of effective calculability: *the notion just did not arise from an analysis of the intuitive understanding of effective calculability*[m.i.]. I conclude that Church was cautious in a similar way as Gödel.

Later on in his [Sie97], Sieg goes on to argue not only that Church did not view λ-definability as one among equally natural definitions of effective calculability, but that recursiveness itself was in fact regarded as a more natural definition of calculability ([Sie97], p. 157):

> That the thesis was formulated for general recursiveness is not surprising when Rosser remark in his [Ros84] about this period is *seriously* taken into account: “Church, Kleene, and I each thought that general recursiveness seemed to embody the idea of effective calculability, and so each wished to show it equivalent to λ-definability”.[85] (p. 345) There was no independent motivation for λ-definability to serve as a concept to capture effective calculability, as the historical record seems to show: consider the surprise that the predecessor function is actually λ-definable and the continued work in 1933/4 by Kleene and Rosser to establish the λ-definability of more and more constructive functions. In addition, Church argued for the correctness of the thesis when completing the 1936 paper (before July 15, 1935); his argument took the form of an *explanation* of effective calculability with a central appeal to “recursivity”.

While it is indeed a fact that λ-definability does not have a direct appeal to our intuition of calculability – in the end, Church did not start from the intuitive notion itself, but only came to the conclusion of his thesis through a serious study of λ-calculus – there are some very clear arguments which show that

[85] It should be noted that despite this remark by Rosser, he expresses Church’s thesis in terms of λ-definability, not in terms of recursiveness in the same paper Sieg refers to in this quote.

Sieg's interpretation here is not that well-argued. Indeed, as we will show, it can be seriously doubted that Church's reason to use recursiveness instead of λ-definability is rooted in the fact that, on the one hand, Church himself was not completely convinced of λ-calculus's computational power, and, on the other hand, Church believed recursiveness to be a more suitable formalization of calculability than λ-definability.

From footnotes 3 and 16 from Church's [Chu36c], it is clear that the proof that any recursive function is λ-definable is due to Kleene and Rosser. A proof of the theorem can be found, as is stated by Church, by applying the methods presented in Kleene's [Kle35a, Kle35b]. The result that every λ-definable function is recursive, "*was obtained independently by the present author [Church] and S.C. Kleene at about the same time.*"[86] and published as [Kle36b]. However, no mention is made of the exact date at which these results were established. The paper by Kleene proving the equivalence, as well as Church's [Chu36c] containing the footnotes, were submitted only some months *after* Church publicly announced the thesis. Despite this lack of the exact dates, Sieg has provided arguments on the basis of which he concludes that the equivalence between λ-definability and recursiveness had already been established before Church submitted his abstract, and he uses this result as an argument for his explanation of why Church waited some time before publicly announcing the thesis, and replaced recursiveness by λ-definability. The arguments however given by Sieg to show that this equivalence was already established before March 22, 1935 are not convincing. We will not discuss this in the main text, but the interested reader is referred to the long footnote.[87] Notwithstanding the fact that,

[86] [Chu36c], footnote 17

[87] The arguments Sieg gives for the equivalence between λ-definability and general recursiveness being proven before March 1935 are based on two letters from Church to Bernays, the first dated January 23, 1935 the second dated July 15, 1935 as well as the fact that Church used recursiveness instead of λ-definability in the talk from April 19 1935: "*if the inclusion of λ-definability in recursiveness had not also been known by then, the thesis could not have been formulated coherently in terms of recursiveness*". Now, from [Kle35b, Chu36c] and the letter Church wrote to Bernays dated January 23, 1935 it is clear that the reducibility of general recursiveness to λ-definability had indeed already been established before March 1935. There is however no definite support given by Sieg that the converse direction, that every λ-definable function is re-

for now, we can only guess whether this equivalence was proven before or after Church submitted his abstract in March, 1935 it is important to further discuss Sieg's conclusions in this context.

As was said, the fact that Church only submitted his abstract after this equiv-

cursive, had already been proven by then. Sieg uses the letter Church wrote to Bernays but does not make clear what letter is meant: the letter dated July 15, 1935 or that dated July 23, 1935. In this last letter Church explicitly refers to his abstract from March and mentions his 1936 paper [Chu36c] as "in the process of being typewritten" (quoted from [Sie97], p. 163) He furthermore mentions that Kleene's paper on the equivalence was forthcoming. Sieg then concludes: "*[...] neither from Kleene's or Rosser's historical accounts nor from Church's remarks it is clear,* when *the equivalence was actually established. In view of the letter to Bernays and the submission date for the abstract, March 22, 1935, the proof of the converse must have been found after January 23, 1935, but before March 22, 1935. So one can assume with good reason that this result provided to Church the additional bit of evidence for actually publishing the thesis.*" ([Sie97], p. 163). As was said, either Sieg is pointing at the letter to Bernays from January 1935 or that from July. If he refers to the earlier letter, this does not add any strength to the conclusion, since Sieg does not give any quote or annotation from this letter supporting this conclusion. If the second letter is intended this neither supports the conclusion since it was dated in July, about 4 months after Church had submitted his abstract and presented his thesis. Now, Kleene only submitted an abstract of his equivalence proof at the end of June 1935 (the abstract was received July 1, 1935 by the American Mathematical Society). The question then of course is, supposing that Kleene and Church had established this result before March 1935, why Kleene waited 4 months to submit an abstract of this result. Furthermore, Church's [Chu36c] mentioning this equivalence result, was at that time in the process of being typewritten, again 4 months after the abstract was submitted. The argument given by Sieg that Church used recursiveness instead of λ-definability in the talk neither adds strength to this argument. As was already pointed out by Davis, the fact that λ-definability occurs only by implication in the reference to "*other plausible definitions of effective calculability ... either equivalent to or weaker than recursiveness.*" rather leaves the impression that Church, at that time, was still uncertain about the equivalence. It should also be mentioned here that Rosser [Ros82] in his account of the history of the λ-calculus states Church's thesis not in terms of recursiveness but in terms of λ-definability, and understood the equivalence proof as a support for this form of the thesis. To summarize, until now there is no definite evidence for the fact that the reduction from λ-definability to recursiveness had been completed before or after Church submitted his abstract in March. And, as will become clear from the remaining discussion, even if this result would have been established before March 1935, this still does not imply that Church was almost as reluctant as Gödel to formulate his thesis, nor the idea that Church, Rosser and Kleene believed recursiveness to be more well-suited and intuitive than λ-definability.

alence was proven (although we can doubt this) and, in the abstract, defined effective calculability in terms of recursiveness instead of λ-definability, shows, according to Sieg, that Church was reluctant to put forward his thesis in terms of λ-definability and that Church did not count λ-definability as a natural definition for effective calculability since the notion "*just did not arise from an analysis of the intuitive understanding of effective calculability*".

We are not specialists as far as the λ-calculus is concerned, but we are familiar with its basic mechanism and how to define and compute a computable function in λ-calculus. Now, when you first start working with λ-calculus, e.g. performing an addition, the least one can say is that it is rather counterintuitive to perform computations in λ-calculus. Following the definition of addition as given in [Chu41] – the nice little orange book – performing the rules of conversion of the calculus, one is almost surprised to see that after some conversions one has performed an addition. This was my own experience, and I had the occasion to check this with a group of other people. During a small colloquium called *Mathematik für Künstler* (mathematics for artists) at the Kunsthochschule in Hamburg, I gave a kind of strange workshop on λ-calculus. Some people came to me at the blackboard, after I had performed some calculations in the calculus saying that they didn't see how e.g. an addition was performed, although the result was there on the blackboard. I gave them a chalk and let them do the operations by themselves. Each of them was as surprised as I was at first, to see that after some conversions, they had indeed performed an addition, looking back at the several steps to understand at what moment exactly the addition "happened".

This small story illustrates how counterintuitive it is, *at first* to compute with λ-calculus and in this respect Sieg is certainly right in stating that, *if one starts from an analysis of the notion of effective calculability*, one will most probably not end up with something like the λ-calculus. In the end, the calculus was never intended to be the result of such an analysis when it was first conceived. It was only after Church, Kleene and Rosser understood what λ-calculus is capable of that Church proposed his thesis. As was said before, it was not the concept of λ-definability itself that led to the thesis, but rather the results established about it, to use Kleene's words [Kle81b].

Although we completely agree with Sieg that λ-definability does not naturally arise from an analysis from our intuition of effective calculability, we cannot neglect that it was λ-calculus and not general recursiveness that led Church to his thesis (and was convincing enough at least for Rosser and Kleene). Even though he was already familiar with general recursiveness during Gödel's lectures he did not use the notion of recursiveness but hung on to his own λ-definability in his discussions with Gödel. As he states in footnote 18 of [Chu36c] it was Gödel who first brought up the question of the relationship between effective calculability and general recursiveness, not Church.[88] To Sieg, the fact that Church waited to submit his abstract, although he had already formulated the thesis, indicates that he understood recursiveness as a more natural definition for effective calculability as compared to λ-definability, adding strength to his point of view by mentioning the surprise of the possibility to λ-define the predecessor function. However, other explanations can be given here.

First of all, Church must have been impressed by Gödel's criticism. Given his very negative reaction towards λ-definability as a definition for effective calculability, while he seems to have been slightly less negative about general recursiveness, Gödel's reaction might have been one of the reasons for Church to be a bit hesitant and to use recursiveness instead of λ-definability in his first public announcement of the thesis.

Another explanation is indirectly given by Kleene([Kle81a], p. 62):

> The earliest notion, λ-definability, has [...] the remarkable feature that it is all contained in a very simple and almost inevitable formulation, arising in a natural connection with no prethought of the result. And a given λ-formula engenders the computation procedure for the function it defines. Of course, the λ-formula may be complicated. Under Herbrand-Gödel general recursiveness,

[88] As to the extent Gödel's work influenced Church's, Kleene has noted ([Kle87], p. 491): *One sometimes encounters statements asserting that Gödel's work laid the foundation for Church's and Turing's results [...]. It seems to me that the truth is that Church's approach through λ-definability and Turing's through his machine concept had quite independent roots (motivations), and would have led them to their main results even if Gödel's paper [Göd31] had not already appeared.*" This claim is supported even more by Emil Post's early work, which was done at a time that Gödel was only 15 years old.

> and my partial recursiveness adapted from it, one works with systems E of equations that can be very unwieldy. Under Turing computability one may have very long machine tables. Indeed Turing [...] spoke of the λ-definitions as "more convenient".[89] (As I see it, convenience for one or another purpose requires testing in practice.) I myself, perhaps unduly influenced by rather chilly receptions from audiences around 1933–1935 to disquisitions on λ-definability, chose, after general recursiveness had appeared, to put my work in that format. [...] I thought general recursiveness came the closest to traditional mathematics. It spoke a language familiar to mathematicians, extending the theory of special recursiveness, which derived from formulations of Dedekind and Peano in the mainstream of mathematics. I cannot complain about my audiences after 1935, although whether the improvement came from switching I do not know. In retrospect, I now feel it was too bad I did not keep active in λ-definability as well.

As is clear from this quote, after his experience with his audience during lectures or courses on λ-definability during 1933–1935, Kleene shifted to general recursiveness because it is closer to "traditional mathematics". To convince an audience at that time that the λ-definable functions – without reference to intuition of course – are exactly those that are effectively computable must have been very hard, given the non-familiarity of the public with λ-calculus. To Kleene, it must thus have been more "convenient" to use the more accessible general recursive functions instead of the difficult λ-calculus taking into account the public. We are inclined to apply this same reasoning to Church's "reluctance" to put forward his thesis in terms of λ-definability.

We are not convinced by Sieg's conclusion that Church used recursiveness instead of λ-definability in his first announcement of the thesis, because he considered recursiveness as a more natural definition for computability. Rather we believe that Church used recursiveness because the mathematical public would be more open to this identification, since they were more familiar with

[89] "*The identification of effective calculable functions with computable functions is possibly more convincing than an identification with the λ-definable or general recursive functions. For those who take this view the formal proof of equivalence provides a justification for Church's calculus, and allows the 'machines' which generate computable functions to be replaced by the more convenient λ-definitions.*" ([Tur37], p. 153)

the idea of recursion, Gödel’s reaction only adding strength to this consideration.[90]

There is indeed no definite reason to suppose that Church (or Kleene, or Rosser) understood recursiveness as a more natural definition. In this context, it should be emphasized that Gödel had made clear to Church that he regarded neither λ-definability nor general recursion, as good formalizations for effective calculability. It was only after he had read Turing’s paper [Tur37] – which starts from an analysis of the intuitive notion of computability to find a suitable formalization – that Gödel became convinced.[91] This illustrates that there is no reason to suppose that recursiveness would better serve its goal. Furthermore Church was well aware of the fact that, as he mentions in footnote 3 of his [Chu36c], his analysis of effective calculability could be “*carried through entirely in terms of λ-definability, without making use of the notion of recursiveness*”. Following this quote, and this is the most convincing argument, Church also explicitly stated that as far as his opinion is concerned, recursiveness and λ-definability are to be considered as equally natural definitions of effective calculability, a remark not mentioned by Sieg.[92]

In discussing the possible influence of Gödel’s results on Church’s and Turing’s, Kleene ([Kle87], p. 491) states:

> One sometimes encounters statements asserting that Gödel’s work laid the foundation for Church’s and Turing’s results [...] It seems to me that the truth is that Church’s approach through λ-definability and Turing’s through his machine concept had quite independent roots (motivations) and would have led them to their main results even if Gödel’s paper [Göd31] had not already appeared.

According to Kleene, Gödel’s impact on Church’s results should not be overestimated. The fact that he sees the originality of Church’s approach in his use of λ-calculus (and not his later use of general recursiveness) again emphasizes

[90] One could maybe put this a bit stronger and state that the mathematician’s and logician’s intuition of computability was more open to recursiveness at that time, because the latter was already much more integrated into the general knowledge of the mathematicians.

[91] Cfr. the postscript added to [Göd34] in Davis’s [Dav65b] where Gödel makes this explicit.

[92] “*The fact, however, that two such widely different and (in the opinion of the author) equally natural definitions of effective calculability turn out to be equivalent [...]*” [Chu36c], p. 346.

the significance of λ-definability for Church's thesis. Furthermore, after having acknowledged that one important influence from Gödel might have been his encoding system, Kleene discusses in a footnote Church's use of general recursiveness, but immediately mentions Church's footnote quoted above (i.e. [Chu36c], p. 346, footnote 3). This suggests that Kleene did not regard general recursiveness as a fundamental influence on Church's results. Also, in his review of Turing's 1936 paper [Tur37], Church says [Chu37b]:

> [computability by a Turing machine] has the advantage of making the identification with effectiveness in the ordinary (not explicitly defined) sense evident immediately – i.e. without the necessity of proving preliminary theorems. [General recursiveness and λ-definability] have the advantage of suitability for embodiment in a system of symbolic logic.

From this quote it is clear that Church understood Turing computability as a more intuitively appealing definition of effective calculability, as compared to both general recursiveness as well as λ-definability, but makes no qualitative differentiation between recursiveness and λ-definability.

In the end, we cannot come to a definite conclusion concerning Church's use of recursiveness instead of λ-definability in first announcing his thesis in public. Still, it remains a fact that Church did not start from an analysis of the intuitive concept of effective calculability. It was the non-expected power of λ-calculus that made him state the thesis. In this sense, Sieg's emphasize on the significance of λ-definability being not intuitively appealing, is slightly anachronistic, since the significance of the "direct appeal to intuition" argument only became clear to Church, after he had read Turing's paper. The fact that it were the results established about the formalism of λ-calculus itself, rather than the idea of finding an adequate formalization of the intuition, is not only important as a historical fact, but also, from a more philosophical point of view: the fact that a formalism that is further removed from intuition, is capable to capture the intuition, at least if one accepts Church's thesis, shows that our intuition is very much restricted, i.e., in confronting it with other ways of computing, like e.g. doing an addition in λ-calculus, one can only learn the rich variety of processes covered by computability.

2.4 From typewriters to universal computing machines

2.4.1 Introduction

While Post and Church were already in their thirties when they wrote their 1936 papers, Turing was still very young. He was only 24 years old when he submitted his paper to the London Mathematical Society. As a consequence it is impossible to give the kind of analyses of Turing's earlier work here as we did for Church and Post, since there is hardly any earlier work.

Turing was born on 23 June 1912 in an upper-middle class family. The best book we believe ever written on Turing's life and work is Andrew Hodges' wonderful biography of Turing [Hod83]. Most of the information used here comes from or is inspired by this book. To give a summary of Turing's influence on computer science, philosophy of computer science, mathematical logic, mathematics and possibly even world history is very difficult and we will only give an impression here.[93] Turing did research in a large variety of domains. As we will discuss immediately, he started his career with a dissertation in probability theory. The influence of Turing's machines can hardly be underestimated. His analysis of computability that resulted in his Turing machines is still regarded as being the most convincing one, compared to those given by Church and Post, it is also the best-known of these three. The Turing machine concept is still a paradigm n many theoretical branches of computer science. For example, it is still the framework to define the complexity of certain algorithms in the context of computational complexity theory.

Turing constructed the theoretical foundations of a universal computing machine, but also actually contributed to the design of one of the first computers, called the ACE [Tur47]. He wrote several more philosophical papers on intelligent machinery, that laid the basis for the so-called Turing test (See e.g. [Tur69, Tur50]). Together with Post, he is one of the founders of recursion theory through his dissertation *Systems of logic based on ordinals* [Tur39] written

[93] The volume edited by Herken, *The universal Turing machine* [Her88] as well as the recently published *Alan Turing: Life and Legacy of a Great Thinker* [Teu04] give a clear overview of the impact of Turing's work.

under the supervision of Church. He was one of the first to do computer experiments (See Sec. 4.2), and made contributions to the theory of morphogenesis with his *The chemical basis for Morphogenesis* [Tur52b] as well as some other papers.[94]

One of his most valuable contributions not to mathematics but to world history is his involvement in the work at Bletchley Park, where he made significant contributions to breaking the Enigma code used by the Germans.[95] He also contributed to breaking the Fish material,[96] In this function, Turing went to the U.S. for highest-level communications. Later he became the "all-purpose consultant" at Bletchley park. After the war, he continued working for GCHQ, the post-war successor to Bletchley Park. Turing knew many high-level secrets and this might have played a role in his later conviction for homosexuality and the consequent punishment of chemical castration with injections of oestrogen, since homosexuality was not only forbidden by law in conservative Britain, but also considered a security risk (a potential source of blackmail). Turing died on 7 June, 1954. The coroner's verdict was suicide from eating an apple laced with cyanide.

2.4.2 Typewriters and "Little wonders"

As is pointed by Hodges ([Hod83], pp. 7–8), Turing

> [...] was one of those many people without a natural sense of left and right, and he made a little red spot on his left thumb, which he called the 'knowing spot. [...]

[94] In the recently published volume of Turing's life and legacy [Teu04], there is one paper discussing this later work of Turing [Swi04].

[95] He generalized the Bombe developed by Polish crypto-analysts, into a powerful device that in fact mechanized certain logical deductions, searching for as many conclusions as possible until a contradiction was found. Hodges makes a nice link here with an extended argument Turing had some years earlier with Wittgenstein. Wittgenstein was doubting the significance of contradictions, and Turing reacted by saying that as long as one does not have a consistency proof one cannot completely trust the system one is working in and if there is a hidden contradiction in a given system this might lead to disastrous consequences when applying the system. (See [Hod83], pp. 153–154, pp. 183–185)

[96] Messages encyphered on a different system, used for Hitler's strategic communications.

> he had great difficulty in writing. His brain seemed barely coordinated with his hand. A whole decade of fighting with scratchy nibs and leaking fountain-pens was to begin, in which nothing he wrote was free from crossing-outs, blots and irregular script which veered from stilted to depraved.

Given his problems with writing Turing began to invent his own "machines" to improve his writing abilities. There are two letters from 1923 mentioned by Hodges ([Hod83], p. 14) in which the young Turing describes two such machines, the one being a 'fountain pen', the other describing a crude idea for a typewriter. Turing never stopped inventing machines, later he even designed a special-purpose machine to study the Riemann-Zeta function. As is described in Hodges's biography, Turing has a fascination with automatization throughout his whole life and this probably played an important role in Turing's analysis of computability, resulting in his machines that share certain features with typewriters ([Hod83], p. 96–97).
Several fascinations, besides those for machines, influenced Turing's description of Turing machines. One such fascination was the idea to regard a mechanical procedure as a concrete physical process in nature (i.e. the human brain) and culture (i.e. machines). This interest that was kindled by the book *Little wonders every child should know*, that Turing receivedd in 1922 from some unknown benefactor. Regarding this book, Hodges states (p. 11): "*If anything at all can be said to have influenced [Turing], it was this book [...]*". The book gives a naive mechanistic picture of life and the mind and must have made a strong impression on the young Turing.

2.4.3 The central limit theorem

After he graduated from Sherborne school in Dorset, Turing was awarded a scholarship at King's College where he started upon the mathematics degree courses, as a schedule B candidate in 1931.[97] In the autumn of 1933, Turing attended a course of lectures on the methodology of science by Arthur Edding-

[97] A schedule B candidate would offer for examination the Schedule A courses together with an additional number of more advanced courses.

ton. One of the subjects Eddington discussed was the observation that scientific measurements, when plotted on a graph, tend to be distributed on a Gaussian or normal curve. However, Eddington merely outlined why this was to be expected, instead of giving a rigorous mathematical explanation.[98] Turing was not satisfied with this sketch, so he set himself the goal to find an exact proof. By the end of February 1934, he had succeeded to prove what is known as the Central Limit Theorem. Only afterwards he was told that this theorem had already been proved by Jarl Waldemar Lindeberg in 1922. Despite this, he was advised that his work might still be acceptable as original work for a King's fellowship dissertation. In November 1934 he completed and submitted his dissertation and in the spring of 1935 he was elected, as the first of his year, as one of the forty-six Fellows. After his paper had entered the Cambridge mathematical essay competition, he was one year later awarded the prestigious Smith's prize for this work.[99]

Although the subject of his dissertation clearly differs from the paper that would be published only two years later in 1936, there are two features of his dissertation that would also characterize his 1936 paper and in fact most of his later work. First of all, as is acknowledged by several authors [Goo80, Hod83, Zab95], Turing often worked in a self-contained way, with limited knowledge of the existing literature on the subject and thus starting from first principles. In the interesting paper [Zab95], in which Turing's work on the central limit theorem is analyzed, Zabel remarks (p. 490):

> Coming to the subject as an undergraduate, his knowledge of mathematical probability was apparently limited to some of the older textbooks (...) it is clear that Turing had penetrated almost immediately to the heart of a problem whose solution had long eluded many mathematicians far better versed in the subject than he.

Indeed, when Turing started working on his dissertation he had hardly any knowledge of the field at that time, he simply got triggered by the problem and

[98] See [Hod83], p. 87.

[99] Turing never published his dissertation, since its major result had already been anticipated. However as is argued in [Zab95], it contained other results that were interesting and novel at that time. It can still be found in the archive of King's college library, see [Tur34].

started to work on it in his own way. This was also the case for his 1936 paper. Again he started from scratch, working in his own way. As Hodges (p. 96) remarks:

> [...] he attacked the problem in a peculiarly naive way, undaunted by the immensity and complexity of mathematics. He started from nothing, and tried to envisage a machine that could tackle Hilbert's problem, that of deciding the provability of any mathematical assertion presented to it.

As a result of this isolated way of working, he was again not the first to arrive at his main result. When Turing developed his ideas for the paper,[100] Alonzo Church had already officially announced some of his main results and it was only when he had already written the paper that he first heard of Church's result. Still, Church was very positive about Turing's paper as we will discuss in Ch. 3.

Besides this 'isolated' way of working, using his own symbolism and terminology, another feature already present in his dissertation reappears in his 1936 paper, connecting certain physical processes with abstract mathematical thinking. As was already pointed out, Turing had a sheer fascination for the connection between the seeming erratic natural processes and the (possibly) deterministic processes underlying it. With the central limit theorem he had proven how one can obtain order out of the most basic kind of erratic processes observed during scientific measurements. In his 1936 paper, he would again make such a connection, by showing how abstract logic can be connected to the processes of computing.[101]

[100] According to Turing himself, it was during an afternoon in the early summer of 1935, lying in the meadow at Grantchester, that he first understood how to answer the Entscheidungsproblem

[101] It should be noted here that besides his proof of the central limit theorem Turing had already made another "small-scale discovery" as he called it ([Hod83], p. 94) and led to his first publication [Tur35]. The results was a small improvement on a paper by John von Neumann, developing the theory of almost periodic functions, defining them in connection to group theory. Since we are not specialists in the domain neither of group theory nor of almost periodic functions and no paper has been published that discusses this paper by Turing, we have excluded it here.

2.4.4 Newmann's course on the foundations of mathematics

In the spring of 1935 Turing attended Newmann's part III course on the foundations of mathematics, of which the last part was the proof of Gödel's incompleteness theorems. It was here that he first heard of the Entscheidungsproblem. According to [Hod83] it was the notion of a "mechanical procedure" that must have catched Turing's ear. Newman used the words 'mechanical procedure' when explaining that one needed a definite method to decide for every well-formed formula defined over the language of first-order predicate calculus, whether or not it can be derived within this calculus. It was the idea of a mechanical procedure that had to be formalized properly before one could start with a proof of the Entscheidungsproblem and it was Turing who gave a very physical version of such a formalism, based on an analysis of how we humans calculate. This analysis led to Turing's description of Turing machines, finite-state machines operating on an infinite tape that, contrary to Post's normal form, Church's λ-calculus and the Herbrand-Gödel definition of recursive functions, resulted from a direct analysis of the intuitive notion of computability itself. In studying the 'general' features of human computing, he showed how such properties lead to a definite class of functions. As is pointed out in [Dav82] this was exactly what Church expressed in his letter to Kleene to be Gödel's idea of how one might proceed to find a satisfactory identification between the intuitive concept of computability and a given formalism, i.e. to construct a set of axioms that embody the generally accepted properties of the notion. Although Turing did not use axioms, it thus does not come as a surprise that it was only after having read Turing's paper that Gödel became convinced of a thesis stating such identifications.

2.5 Conclusion

In this chapter we have shown through an analysis of Church's, Post's and, to a lesser extent, Turing's earlier work, how each of these mathematicians/ logicians arrived at their respective theses. An important result of our argumentations is the fact that neither Church nor Post started with the explicit goal of proving certain decision problems unsolvable, nor from the idea of finding a proper formalization of the intuitive notion of computability, when they first formulated their respective theses.

Post started from exactly the opposite of proving certain decision problems unsolvable. His main method was to develop more general forms of logic, instead of one specific system of logic, resulting in simpler and more abstract forms of logic. By working with these forms, he hoped that it would be more straightforward to prove the Entscheidungsproblem solvable. It was only after his experience with tag systems that he first considered the possibility that there might exist unsolvable decision problems and they laid the ground for his important normal systems. After this research on tag systems, Post constructed systems in canonical form C and systems in normal form, and proved the important normal form theorem. *On the basis of these results* he then concluded for his thesis, identifying the intuitive notion of generated set with sets of assertions that can be produced through systems in normal form. Given this assumption, he proved the unsolvability of the decision problem for normal systems. To Post's mind however, "*a complete analysis would have to be made of all the possible ways in which the human mind could set up finite processes for generating sequences*" in order for his thesis to be more general.

It was thus only after he had already formulated his thesis and proven certain decision problems unsolvable, that Post set himself the goal of giving an analysis of what we humans understand under "generated set". His thesis and the resulting unsolvability of the finiteness problem for normal systems, however, are rooted in Post's study of the formalisms themselves, rather than in an analysis of the intuitive notion considered formalized in normal systems.

Church on the other hand started from a study of alternative systems of logic and considered consistency as the basic criterium for evaluating systems of logic. Given the difficulties of proving a system consistent, he understood that a more empirical approach is the best one available to build up confidence in a given system of symbolic logic, as long as a consistency proof is missing. After his Ph.D. students Kleene and Rosser had proven that the set of postulates Church considered adequate for the development of mathematics where functions play an important role, attention shifted to λ-calculus. It was the fact that

Church, Kleene and Rosser could λ-define any function over the integers they could think of, that led Church to the first formulation of his thesis, identifying effective calculability with λ-definability. On the basis of this thesis, and its reformulation in terms of recursive functions, Church was able to prove certain decision problems unsolvable. Thus, also in Church's case, one cannot but conclude that it was not an analysis of the intuitive notion of effective calculability that led to his important results. Rather it was a study of λ-calculus that led him to his first formulation of his thesis. Contrary to Post however, Church clearly did not come to the conclusion that he first had to provide an analysis of all the processes the human mind can set up to effectively calculate a function to make his thesis more general, since he announced and published it before having read Turing's paper, or having gone through such an analysis.

As far as Turing is concerned, it is important to take into account that he was still very young in 1936. As a consequence his thesis, identifying the notion computability with Turing machines, can hardly be traced back to his limited amount of work preceding his *On computable numbers* [Tur37]. Basic here is that, contrary to Church and Post, Turing's thesis did result from a direct analysis of the vague notion of computability itself, considering the general properties of the process of human computing. Turing machines then resulted from this analysis.

Chapter 3

1936

In the previous chapter we showed that there are important differences but also, to a certain extent, similarities, between the way Church, Post and Turing each arrived at their theses and the related unsolvability results. In this chapter we will further explore some of the basic differences and similarities between their work, starting from their 1936 papers, focussing on the theses they each proposed.

In a first more descriptive section (Sec. 3.1), we will take a closer look at the exact statements of the theses starting from the 1936 papers [Chu36c, Tur37, Pos36], and, in Post's case, also his *Account of an anticipation* [Pos65]. Focus here, will be put on the actual formulation of the several theses, as well as the supporting arguments present in the work by Church, Post and Turing. We will also provide arguments showing that one can deduce a thesis from Post's 1936 paper, different from his original thesis.

On the basis of this analysis, we will then discuss the problem of identifying the intuitive notion of "computability" with a given formalism, starting from the two reviews written by Church on Post's and Turing's 1936 papers (Sec. 3.2). It will be shown that already at the time of the original formulations, there were important discussions on the actual status of such theses: should they be regarded as theorems, as (hypo)theses or as definitions? It will be argued that one's preference with respect to the status of these "theses", has a close connection with the kind of arguments one considers as the most important. In this respect, it will become clear why Turing's thesis is very often considered as the most adequate one, and by some, even as the only really convincing thesis. In discussing several possible interpretations of the theses we would like to relativize the "dominance" of Turing's thesis.

In a last, shorter, section (Sec. 3.3) we will offer our own more philosophical thoughts on the subject, drawing from our historical results of this and the previous chapter. We will argue here that although it seems that one has very quickly come to the consensus that Turing's thesis is, in a way, the most convincing, one should be very careful in coming too quickly to any conclusion in this respect. In fact, we will argue that it is not only of historical but also of philosophical significance to not restrict one's attention to that which is the most intuitively appealing.

It is important to note here, that we will not take into account (yet) the ongoing debate on the physical Church-Turing thesis and the idea of beating it, at least not in any detail. This will be done in Sec. 4.3. It should also be mentioned that, although the title of the chapter is 1936, we cannot but include a further discussion of Post's earlier work, since he already formulated his thesis and concluded for the unsolvability of certain decision problems in 1921.

It is also important to point out that several other mathematicians and logicians have formulated theses comparable to those by Church, Post and Turing and made important contributions in this context, but these will not be discussed here in much detail. For example, Kleene's work should not be underestimated in this context, and we will discuss it to some extent.[1] Although Gödel's work on incompleteness and his more philosophical thoughts on the subject are also very important here, we do not have the space to discuss them in depth.[2]

[1] The paper [Sho96] discusses Kleene's work, and its invaluable role for the constitution of recursion or computability theory. Webb's book [Web80], gives a central role to some of Kleene's results as support for the validity of the Church-Turing thesis.

[2] Several papers and books have been written on Gödel's work and his more philosophical ideas and it is impossible to give an exhaustive overview here. Biographical information can be found in [Daw97, Wan87, Wan96]. The last two books contain lots of material on Gödel's more philosophical thoughts. We should also mention the special issue on Gödel of the *Bulletin of symbolic Logic*, vol. 11, nr. 2, 2005, as well as a special issue of *Philosophia Mathematica*, vol.

3.1 Different questions, different answers.

In this section we will discuss the general content of each of the 1936 papers, some of the methods used, as well as the exact formulation of the respective theses and the arguments – if provided – supporting them. It should be pointed out to the reader that part of this section is descriptive. I.e., some paragraphs will contain material that is merely summarizing the results as originally described by Church, Post and Turing. In order to minimize these descriptive parts, we have used as many intermezzo's as possible, which might be skipped by the reader who is familiar with the results summarized.

3.1.1 "An Unsolvable Problem of Elementary Number Theory"

As is clear from its title, Church's paper [Chu36c] does not start from the notion of effective calculability but from unsolvable decision problems. This is also clear from the introduction of this paper. Its first sentences are ([Chu36c], p. 345):

> There is class of problems of elementary number theory which can be stated in the form that it is required to find an effectively calculable function f of n positive integers, such that $f(x_1, x_2, ..., x_n) = 2$ is a necessary and sufficient condition for the truth of a certain proposition of elementary number theory involving $x_1, x_2, ..., x_n$ as free variables.

After having given one of two examples of such problems – involving Fermat's last theorem – Church writes:

> Clearly, the condition that the function f be effective calculable is an essential part of the problem, since without it the problem becomes trivial.

The purpose of the paper thus becomes:

> [...] to propose a definition of effective calculability which is thought to correspond satisfactorily to the somewhat vague intuitive notion in terms of which

14, nr. 2, 2006.

> problems of this class are often stated, and to show that not every problem of this class is solvable.

From 2.3, we already know the history preceding the publication of this paper and it was neither an analysis of the notion effective calculability nor the idea of proving certain problems unsolvable, but the computational power of λ-calculus that lay the basis for this paper. Still we think it important to note that, notwithstanding our knowledge of the events preceding this paper, Church starts from the problem of proving specific problems solvable or unsolvable to tackle the problem of formally *defining* the vague notion of effective calculability.

Of more significance here is the fact that Church interpreted the identification of effective calculability with general recursiveness and λ-definability as a definition. As we will see in 3.1.3 and discuss in 3.2, this interpretation stands in sharp contrast with Post's ideas in this context.

In the sections following the introduction, Church introduces the λ-calculus, the Gödel representation of formulae and recursive functions. Church proves or mentions several theorems with respect to λ-definability and recursiveness, including the two theorems stating he equivalence between λ-definability and recursive functions.

Church's statement of the thesis

After this exposition Church again considers the problem of identifying the intuitive notion of effective calculability with a certain formalism and proposes the following "definition" ([Chu36c], p. 356):

> We now define the notion [...] of an *effectively calculable* function of positive integers by identifying it with the notion of a recursive function of positive integers (or of a λ-definable function of positive integers.)

He immediately adds (p. 356):

> This definition is thought to be justified by the considerations which follow, *so far as positive justification can ever be obtained for the selection of a formal definition to correspond to an intuitive notion.* [m.i.]

As is clear from these two quotes, Church indeed regarded the identification he made, as a definition, not as a thesis. Despite his calling the identification a definition, he understood that it is far from unproblematic to offer a positive justification for his definition, i.e. it cannot be proven to be absolutely true. Church's thesis can be stated as:

> **Church's Thesis.** *Every effectively calculable function is general recursive (λ-definable) and conversely.*

We will now discuss the justification provided by Church in his paper, relying on Gandy's analysis of and critique on Church's arguments [Gan88] in this context. In discussing Church's paper [Chu36c], Gandy points out four different arguments supporting Church's thesis, relying on Kleene's [Kle52]. These arguments are:

(A1) *The argument by Example.* This is the argument that led Church to the first formulation of his thesis in terms of λ-definability, i.e., the fact that one can represent any function one can think of in the formalism one is working with, as was the case for λ-calculus in Church's work. In his 1936 paper however [Chu36c], no mention is made of this argument. This argument was also used by Turing in his 1936 paper.

(A2) *The Step-by-Step Argument.* Church considered this argument as the main justification for his thesis in [Chu36c]. He announces this arumengt in the quote given above. He considers two possible methods that can be used to evaluate or compute a function $f(x)$, two methods one might identify on an intuitive level with the notion of effective calculability. These are: the application of an algorithm that computes the value of $f(x)$ *and* the derivation of $f(x) = y$ from a set of axioms, after application of a certain number of operations or rules of procedure. For each of these two methods, the computation is done in a series of steps. Church then interprets this step-by-step procedure as a recursive process, i.e. each step performed is a recursive step. Then, since each step is recursive, f must also be recursive. For Church there is no more general definition of effective calculability than the one he proposed, that can be obtained by analyzing either of the two methods (computations through algorithms or in a logic). I.e. there is no more general way to describe the step-by-step processes underlying these two methods than to describe them in terms of recursive steps.

(A3) *The Argument by Confluence.* The argument by confluence concerns the fact that very different formalisms that are each considered capable to capture the intuitive notion of effective calculability, are proven to be equivalent. This argument is used by Church in his 1936 paper [Chu36c], in a footnote (p. 346, footnote 3):

> The fact, however, that two such widely different and (in the opinion of the author) equally natural definitions of effective calculability turn out to be equivalent adds to the strength of the reasons adduced below for believing that they constitute as general a characterization of this notion as is consistent with the usual intuitive understanding of it.

This argument was also used by Post, Kleene and Turing.

(A4) *The Criterion of the Failure of the Diagonal Argument.* Although this argument was not mentioned by Church in the paper proposing the thesis [Chu36c], it is clear that it played a role. This argument was also used by Post in his [Pos65], as was shown in Sec. 2.2.4. He used the fact that the diagonalization cannot be done effectively to argue that the sequence defined through the diagonalization does not contradict his thesis. Also for Kleene this argument played a basic role in his acceptance of the thesis.[3] Turing also considered this argument, as we will see in Sec. 3.1.2.

Gandy gave several objections to **(A1)**–**(A3)**. Sieg [Sie97] also criticized **(A2)**, and has called Church's interpretation of the steps of any effective procedure as recursive steps, *Church's central thesis.*

As far as **(A1)** is concerned, Gandy notes that although this argument can be

[3] The reader is referred to Sec.2.3.4, for the quote by Kleene in which he states that he became an overnight supporter of the thesis, in having realized that the diagonalization cannot be done effectively.

used to justify the heuristic value of the thesis, it cannot be used to settle the philosophical or foundational question, in the sense that it does not exclude the possibility that some day someone might establish an entirely new kind of calculation that is not covered by Church's thesis. This is in fact a general problem of any argument supporting a thesis equivalent to Church's, in that no argument can be found that excludes this possibility. Indeed, one cannot give a proof for the thesis, since one is working with intuitive concepts.

Another objection with respect to **(A1)** might be to give an example of something we would consider effectively calculable by intuition, but we cannot "encode" it in any formalism equivalent to λ-calculus like Turing machines or tag systems. I.e., even if one has e.g. λ-defined thousands of functions, one can never be sure whether there will not be some special kind of function left, we consider as effectively computable, that cannot be λ-defined. But again, this problem is inherent to the problem the thesis wants to tackle, i.e., the formalization of an intuition, rather than to the argument.

Similar objections can be made with respect to **(A2)**, **(A3)** and **(A4)**, and we will thus not discuss them here. As far as **(A2)** is concerned, personally I find it hard to accept this argument as a real argument, since the only way I can understand it, is that Church himself simply did not see a more general way than to interpret any "step-by-step" procedure as a "step-by-recursive-step" procedure. Although I for myself neither see any other more general way than to understand computability in terms of recursion or any other equivalent formalism, it is not an argument one can use with respect to the sceptical person who wants to beat the so-called Turing limit. Also Gandy [Gan88], Sieg [Sie94, Sie97] and Soare [Soa96] pointed out the problems related to **(A2)**.[4]

Unsolvable decision problems in λ-calculus

After Church discussed his definition of effective calculability in terms of general recursive functions, he could pass on to the proofs of the existence of cer-

[4] Soare for examples notes: "*The fatal weakness in Church's argument was the core assumption that the atomic steps were stepwise recursive, something he did not justify.*" ([Soa96], p. 290).

tain unsolvable decision problems. These proofs depend on the following theorem:

Theorem 3.1.1 *There is no recursive function of a λ-formula* **C**, *whose value is 2 or 1 according as* **C** *has a normal form or not.*

In other words, the property of a λ-formula having a normal form is not recursively solvable and thus not-computable.[5] We will not enter into the details of the proof of the theorem, since it involves explaining the λ-definition of several different functions. However, it is important to at least point out that Church λ-defined a rather involved function, composed out of several different λ-defined functions of which many were defined by Kleene [Kle35a, Kle35b], that can be interpreted as a kind of procedure that should be able to evaluate whether a given formula is convertible to formulas 1, 2, 3,... (which are all in principle normal form).[6] This λ-formula $\mathfrak{e}$ is defined as follows:

$$\mathfrak{e} \rightarrow \lambda n.\mathfrak{d}(\mathfrak{h}(\mathfrak{b}(\mathfrak{a}(n),\mathfrak{z}(n))),\mathfrak{b}(\mathfrak{a}(n),\mathfrak{z}(n)))$$

As was said, we will not give a detailed explanation of this function, but merely indicate how $\mathfrak{e}$ works. If n is one of the formulas 1, 2, 3,... then $\mathfrak{e}(n)$ is convertible into one of the formulas 1, 2, 3,....as follows:

1. if $(\mathfrak{a}(n),\mathfrak{z}(n))$ can be converted to a formula that stands for the Gödel representation of a formula which has no normal form, then $\mathfrak{e}(n)$ conv 1.

2. If $(\mathfrak{a}(n),\mathfrak{z}(n))$ converts to a formula that stands for the Gödel representation of a formula having a (principal) normal form which is not one of the formulas 1, 2, 3,.... then also $\mathfrak{e}(n)$ conv 1.

3. If $(\mathfrak{a}(n),\mathfrak{z}(n))$ is converted to the Gödel representation g of a formula that has a (principal) normal form which is one of the formulas 1, 2, 3,... then $\mathfrak{e}(n)$ is converted to the next formula $g+1$ in the list 1, 2, 3,....

[5] For the definition of normal form with respect to λ-definability, the reader is referred to Sec. 2.3.4.

[6] Remember that Church used the integers as abbreviations for certain λ-formula. See Sec. 2.3.4.

I will not give the proof based on $\mathfrak{e}$, but will merely point out some of the essential ideas behind the proof. Since we follow Church's exposition, we decided to add this sketch of the proof in an intermezzo.

> Starting from the assumption that it can be determined for every λ-formula that it has a normal form, Church deduced a contradiction. If the assumption holds, then clearly one should be able to determine whether every λ-formula is convertible into one of the formulas 1, 2, 3,...Indeed, given a formula **R** one can first determine whether it has a normal form, and if it has, one can obtain its principal normal form by enumerating all the formulas into which **R** is convertible, picking out the first formula in principal normal form, determining whether it is in the form 1, 2, 3,...[7] Then, let $\mathbf{A}_1, \mathbf{A}_2, \mathbf{A}_3, ...$ be an effective enumeration of the formulas that have a normal form [8], and let E be a function of one positive integer such that $E(n) = 1$ if $\{\mathbf{A}_n\}(\mathbf{n})$ is not convertible into one of the formulas 1, 2, 3,... and $E(n) = m + 1$ if $\{\mathbf{A}_n\}(\mathbf{n})$ is convertible to **m** and **m** is one of the formulas 1, 2, 3,...[9] The function is effectively calculable and is therefore λ-definable by a formula $\mathfrak{e}$. This formula has a normal form since $\mathfrak{e}(1)$ has a normal form. However, $\mathfrak{e}$ cannot be any of the formulas $\mathbf{A}_1, \mathbf{A}_2, \mathbf{A}_3, ...$ because for every n, $\mathfrak{e}(n)$ is a formula not convertible to $\{\mathbf{A}_n\}(\mathbf{n})$. This contradicts the property of the enumeration $\mathbf{A}_1, \mathbf{A}_2, \mathbf{A}_3, ...$ containing every λ-formula that has a normal form, since $\mathfrak{e}$ cannot be part of it, and we have thus deduced a contradiction.

Basic to the proof is the assumption that a function ($\mathfrak{e}$) can be λ-defined determining for any given λ-formula whether it is convertible to one of the formulae

[7] This result follows from the following theorem: *It is possible to associate simultaneously with every λ-formula an enumeration of the formulas obtainable from it by conversion, in such a way that the function of two variables, whose value, when taken of a λ-formula **A** and a positive integer n, is the n-th formula in the enumeration of the formulas obtainable from **A** by conversion, is recursive.*

[8] It was proven that the set of λ-formula that have a normal form is recursively enumerable.

[9] Note that $\{\mathbf{A}_n\}(\mathbf{n})$ is the λ-formula, corresponding to the recursive function that is used to determine for every formula A_n, the n-th formula in the enumeration of the formulas obtainable from A_n.

1, 2, 3,... On the basis of this assumption a contradiction can be deduced by applying a kind of diagonalization (the use of formulas of the form $\{\mathbf{A}_n\}(\mathbf{n})$). One can thus conclude that one cannot λ-define a function that is able to determine whether a given λ-formula has a normal form.
On the basis of this theorem, Church furthermore proved that there is no recursive function of two formulas **A** and **B** whose value is 2 or 1 according as **A** conv **B**.

3.1.2 "On computable numbers, with an application to the Entscheidungsproblem"

As is clear from its title, Turing's focus is on computable numbers. To be more exact, Turing's paper wants to deal with ([Tur37], p. 230):

> the real numbers whose expressions as a decimal are calculable by finite means.

On the first page of the paper, Turing announces how he will *define* the computable number (p.230):

> According to my definition, a number is computable if its decimal expansion can be written down by a machine.

As was shown in Sec. 2.4, contrary to both Church and Post, who first formulated their thesis after having convinced themselves of certain properties of the formal systems they were studying, Turing did not start from a given formalism, but deduced one on the basis of his analysis of the process of human computing. Indeed, one could say he constructed his Turing machines by taking together some of the generally accepted properties he deduced from his analysis of such processes. In this respect, it is important to emphasize that for Turing the real question at stake is:

> What are the possible processes that can be carried out in computing a number?

As we already know from Sec. 2.2.4 it was a similar question Post wanted to solve in order for his thesis to be generally valid, i.e. "*for full generality a complete analysis would have to be given of all the possible ways in which the human*

mind could set up finite processes for generating sequences." ([Pos65], p. 387). In the next section we will see that the kind of analysis Post is pointing at in the quote, in the end resulted in a formalism almost identical to Turing's.
In the remainder of this text we will use the notion *computor* to indicate a human computer, and the usual term *computer*, when machines are concerned, following Gandy [Gan80].

Turing's machines: Universal computing and the halting problem.

In the beginning of his paper, Turing starts to describe some properties of his machines, by making the comparison with a man in the process of computing a real number. As he points out, the more detailed justification for the definition of computability in terms of machines will follow after he has proven some of his basic results. The only justification he already mentions is that since man's memory is limited, the machine's should also be limited in certain ways. First of all, the machine is supplied with a *finite* number of conditions $q_1, q_2, ..., q_R$, called m-configurations. The machine is supplied with a tape which Turing compares with the paper a computor uses. The tape is divided into squares, each capable of bearing a symbol. At any moment i, there is just one square bearing a symbol a_j that is in the machine. Turing calls this square the scanned square, and the symbol on it, the scanned symbol. This scanned symbol "*is the only one of which the machine is, so to speak, "directly aware"*" ([Tur37], p. 231). If an m-configuration is altered the machine is considered to be capable of remembering the symbol it has "seen" previously. The possible behaviour of the machine at any moment is completely determined by the m-configuration it is in, as well as the symbol scanned. The symbol scanned and the m-configuration the machine is in at a given time is called the configuration of the machine. The machine is capable of several operations: if the square it is scanning is blank, it can print a symbol and if it is not blank, it can erase it. The machine can also move one square to the left or to the right. As should be clear to the reader, these features are basic to the description of what we now know as a Turing machine.

In the following intermezzo, we will give a standardized description of Turing machines. The kind of description we are using here is in no way fundamental, but it is important to pick one out. We choose this description, because it is the one most often used in some of the papers we will discuss in part II.[10]

A Turing machine is considered here to consist of a two-way infinite *tape*, subdivided into squares. Each square can contain one and only one symbol. The machine is capable only of a *finite number of states* (m-configurations). It can perform the following kind of operations: move one square to the *left* (indicated as L), move one square to the *right* (indicated as R), *print* a symbol S_i from a finite alphabet $\Sigma = \{S_1, S_2, ..., S_\mu\}$. It should be noted that we do not use an erasure operation, but rather an overwriting operation, i.e. if a square contains a given symbol, it is overwritten by the new symbol printed. In this respect an empty square is from now on identified as a square containing the symbol 0 (S_0). The machine is also capable of recognizing the symbol in the square it is scanning, and to change its state. A quintuple is an expression of the form $q_i S_j : S_k M_l q_m$, where M_l can be equal to L or R. A quintuple completely determines what the machine should do in state q_i scanning the symbol S_j. A Turing machine can then be defined by a finite set of quintuples that contains no two quintuples for which the first two symbols are identical.[11] Later on, we will represent a Turing machines, defined though a finite set of quintuples, by transition tables. For example the following table:

	q_1
0	$1Rq_1$
1	$0Rq_1$

describes a Turing machine defined through a set of two quintuples $\{q_1 0 : 1Rq_1, q_1 0 R q_1\}$. The state of a Turing machine at a given time, is given by its *instantaneous description* (I.D.) at that time, an expression of the form $Pq_i s_j Q$, where q_i is the state the machine is in, s_j the symbol it is scanning, P the content of the tape to the left of s_j and Q the content of the tape to the right of s_j. Using I.D.'s one can describe the dynamics of a Turing machines.

[10] For this description, see e.g. [Min61].

[11] It should be pointed out that Post [Pos47] introduced the use of quadruples instead of quintuples.

Despite the rather simple mechanism behind Turing machines, Turing considered them as being capable to carry out any process to compute a number we humans can carry out. But before further discussing Turing's thesis, it is important to point out some of the other results contained in the paper.

First of all, we must mention the differentiation between *circular* and *circle-free* machines, which is basic to Turing's proof of the *halting problem*. It should be noted though that Turing never used this last term. This is due to Martin Davis [Dav58].

A machine is considered *circular* if it never writes down more than a finite number of symbols, i.e. if it reaches a configuration from which there is no move possible, or gets into a loop. In all other cases, when the machine is actually computing a real number or an infinite sequence of symbols, it is said to be *circle-free*. Contrary to Church, Turing did not use Gödel coding but used his own coding system which is far more efficient if one actually implements it. We will not give the details of this coding, since the readers are probably already familiar with it, but it is important to point out that Turing differentiates between the description number (D.N.) and the standard description (S.D.) of a Turing machine. The D.N. can be constructed from the S.D. by replacing letters by numbers.

Since Turing's coding system is far more efficient than Gödel coding, it is easier to really implement it on a machine. Although Turing, at that time, did not build or design a real physical computer, he did provide a construction of a Turing machine capable to compute anything computable by any other Turing machine, that later influenced his design of a real computer, as well as, most probably, von Neumann's (See Sec. 4.1). The encoding of this machine heavily relies on the S.D. of Turing machines.

I will not give the description of Turing's original universal machine here.[12] It is rather intricate and contains some mistakes (See Post's [Pos47]). Still, I want to at least notice here that, having gone through the operations of this universal machine, noticing the mistakes its instruction table contains, trying to under-

[12] In part II we will deal with other universal Turing machines, which are far simpler in their description.

stand how such a machine might work, has been an experience for me I will never forget. The insight that a universal machine is in a way nothing more than a kind of complicated cut-copy-paste machine, that is nonetheless capable to interpret and execute the operations of any other Turing machine, has been rather important for me. It has resulted, at least for me, in a first change of my own intuitive notion of computations and computers. In a way the processes that can be used to translate machine language to a user-friendlier language, and vice versa, is very much related to this kind of theoretical construction. Fundamental here is that instructions and data are put on one and the same level, and the instructions can thus be manipulated as data.
This universal machine was used by Turing in his famous proof of the unsolvability of the halting problem. Before giving the proof, Turing emphasizes that it should be understood that the diagonalization cannot be done effectively, thus pointing out the same kind of fallacy Post describes, that might be involved in searching for counter examples through the diagonalization process contradicting the thesis. I.e., the fact that one can define a given sequence through diagonalization that cannot be computed by a normal form, does not imply that one has given a real counter example. This would only be valid if the sequence could be effectively computed. Turing summarized the argument and the fallacy underlying it as follows ([Tur37], p. 246):

> It may be thought that arguments which prove that the real numbers are not enumerable would also prove that the computable numbers and sequences cannot be enumerable. It might, for instance, be thought that the limit of a sequence of computable numbers must be computable. This is clearly only true if the sequence of computable numbers is defined by some rule. Or we might apply the diagonal process. "If the computable sequences are enumerable, let a_n be the n-th computable sequence, and let $\phi_n(m)$ be the m-the figure in a_n. Let β be the sequence with $1 - \phi_n(n)$ as its n-th figure. Since β is computable, there exists a number K such that $1 - \phi_n(n) = \phi_K(n)$ for all n. Putting $n = K$, we have $1 = 2\phi_k(K)$, i.e. 1 is even. This is impossible. The computable sequences are therefore not enumerable". *The fallacy in this argument lies in the assumption that β is computable.*[m.i.] It would be true if we would enumerate the com-

> putable sequences by finite means, but the problem of enumerating computable sequences is equivalent to the problem of finding out whether a given number is the D.N. of a circle-free machine, and we have no general process for doing this in a finite number of steps. In fact, we can show that there cannot be any such general process.

This argument is indeed similar to Post's, i.e. the diagonalization cannot be done effectively! This is *only* possible if one would be able to solve by finite means the problem mentioned at the end of the quote, now reformulated as the halting problem.
As is pointed out by Turing, there is a very direct proof of the unsolvability of this problem based on the assumption of the correctness of his identification between computability and Turing machines: if such general process would exist, than there should indeed exist a machine that computes β. But such proof might leave the reader with a feeling that something is missing, and Turing provided another proof. Let us now turn to a description of the original proof by Turing. The proof depends not on the computability of β but on β' whose n-the figure is $\phi_n(n)$. We give the proof in an intermezzo since, as was the case for Church, we merely follow Turing's description of the proof.

> Turing starts from the assumption that there exists a Turing machine that decides for any given number whether it is the D.N. of a circle-free machine, and deduces a contradiction from the assumption. Let us suppose we could invent a machine D that, when supplied with the D.N. of another machine M, tests this machine through its *D.N.*. If D concludes that M is circular it prints the symbol u, and if it is circle-free it prints s. We can then combine D with the universal Turing machine U to construct a new machine H to compute β'.
> H's tape can be subdivided into several sections. Let us suppose that in the first $N-1$ sections, among other things, the integers $1, 2, ..., N-1$ have been printed and are already tested by H. A certain number of these integers, $R(N-1)$ have been found to be the D.N. of circle-free machines. In its N-th section H now has to test the N. If N is the D.N. of a circle-free machine (N is satisfactory) then $R(N) = R(N-1)+1$ and the first $R(N)$ figures of the sequence calculated by the

machine with its D.N. = N are computed. The $R(N)$-th figure is then written down as the $R(N)$-the figure of the sequence β' which can thus in this way be computed by H. If N is the D.N. of a circular machine, (N is not satisfactory) then $R(N) = R(N-1)$ and the machine goes to the $(N+1)$-th section.

Now, from its construction it is clear that H itself should be circle-free. Each section of the motion comes to an end in a finite number of steps, given the assumption that the machine D, part of H, is capable to decide in a finite number of steps whether a given number N is the D.N. of a circle-free machine. If N is satisfactory, the machine M_N, whose D.N. is N, is circle-free so we can use the universal machine U, part of H, to compute its $R(N)$-th figure in a finite number of steps. When this figure is calculated as the $R(N)$-th figure of β', the machines moves to $N+1$. If N is the D.N. of a circular machine, H also finished in a finite number of steps, and moves to $N+1$. Thus, H is circle-free.

Now suppose N is the D.N. of H itself. H must now test whether its own D.N. is satisfactory. It is at this point that a contradiction arises. Indeed, since N is the D.N. of a circle-free machine, H's verdict cannot be that N is not satisfactory. However, neither can H's verdict be that N is satisfactory. If this would be the case, then H should compute in its N-the section, the first $R(N-1)+1 = R(N)$ figures of the sequence computed by H, and write down the $R(N)$-th figure as a figure of the sequence β' computed by H. There are no problems as far as the computation of its first $R(K)-1$ figures is concerned. However, computing the $R(N)$-th figure would amount to "calculate the first $R(N)$ figures computed by H and write down the $R(N)$-th.", this of course is impossible, since, in a way, the machine should be ahead of its own computations to do this. Thus the $R(N)$-th figure could never be computed and H must thus be circular. In other words, if H is applied to itself, it can never give rise to the right verdict, it cannot decide for itself whether it is circular or circle-free. We can thus conclude that no machine H can be constructed, of course, on the assumption of Turing's thesis.

Basic to the proof is that, on the assumption that one can construct a Turing machine H that decides for any Turing machine whether it is circle-free, Turing is able to deduce a contradiction through diagonalization.

After this proof, Turing proved that there can be no machine P which, when supplied with the D.N. of an arbitrary machine M, determines whether M will ever print a given symbol, i.e., he proved that the *printing problem* is unsolvable [Dav58].

Turing's statement of his thesis

After the proof of the unsolvability of the printing problem, Turing starts his discussion of the identification he had to assume to be valid in order to prove the halting problem unsolvable ([Tur37], p. 348):

> The expression "there is a general process for determining..." has been used throughout this section as equivalent to "there is a machine which will determine...". This usage can be justified if and only if we can justify our definition of "computable". For each of these "general process" problems can be expressed as a problem concerning a general process for determining whether a given integer n has a property $G(n)$ [e.g. $G(n)$ might mean "n is satisfactory" or "n is the Gödel representation of a provable formula"], and this is equivalent to computing a number whose n-th figure is 1 if $G(n)$ is true and 0 if it is false.

As was the case for Church, also Turing was very clearly aware of the fact that no argument supporting the thesis can be used as a mathematical proof of the thesis ([Tur37], p. 349):

> All arguments which can be given are bound to be, fundamentally, appeals to intuition, and for this reason rather unsatisfactory mathematically. The real question at issue is "What are the possible processes which can be carried out in computing a number?"

Turing gives three different kinds of arguments. The first two are **(A1)**, the argument by example and **(A3)**, the argument by confluence. As far as **(A1)** is concerned, Turing gives several examples of classes of numbers and functions that can be computed by Turing machines. The argument **(A3)** is the proof of the equivalence between Turing machines and restricted predicate calculus. It is this proof that leads to the unsolvability of the Entscheidungsproblem for this calculus. To be more specific, Turing showed that the printing problem can be reduced to the Entscheidungsproblem.

After he had already submitted the manuscript of the paper, Turing received an offprint of Church's [Chu36c] via Newmann. After having made himself more familiar with λ-definability, he proved the equivalence between his formalization of computatability and λ-definability, and added the proof as an appendix to his [Tur37]. He was thus able to add more strength to the argument by confluence. A more detailed proof was published as [Tur37].

The significance of a third argument (**A2**), described by Turing as *a direct appeal to intuition*, can hardly be overestimated. It is exactly this argument that convinced many people, including Gödel, of the validity of Turing's thesis, and is nowadays still considered by many as the fundamental argument supporting the thesis. Some even regard it is a proof of "Turing's theorem" (instead of Turing's thesis). As a consequence Turing's identification is often regarded as the best or most convincing one available, since the argument only works for Turing machines or similar kinds of formalizations, i.e., formalizations that start from an analysis of the vague intuitive notion. Before further discussing the significance of this argument, we will now summarize the main ideas behind it, based on Gandy's [Gan88] and Sieg's [Sie94, SB96, Sie97] analyses.

The argumentation that is used as a direct appeal to intuition, is in fact Turing's analysis of *a man in the process of calculating something*, a computor, and his deduction of certain properties that are inherent to this process. It is this kind of analysis (See Sec. 2.4) that Gödel thought to be the best way to find a satisfactory identification between the intuitive concept of computability and a given formalism, i.e. to determine a formalism based on generally accepted properties of the intuitive notion. I still consider this part of Turing's paper as a very strong and beautiful philosophical analysis, making clear how one can proceed to formalize certain non-mathematical notions.

Turing starts from the idea that "*[c]omputing is normally done by writing certain symbols on paper. We may suppose this paper is divided into squares like a child's arithmetic book*" ([Tur37], p. 349). Since the two-dimensional character is not essential to computation according to Turing, he assumes that the computor works on a 1-dimensional tape divided into squares. By considering the

limitations of us humans, with respect both to perceptional as well as mental abilities, while in the process of computing something, Turing deduces several restrictions on the actions of a computor and describes on the basis of this deduction an abstract computor. The actions of this abstract computor are then considered replaceable by the actions of a computer, i.e., the actions of a computor can be formalized in a kind of computers which are in fact reducible to Turing machines.
Both Gandy [Gan88] and Sieg [Sie94, SB96, Sie97] correctly deduced the several restrictions Turing concluded for on the basis of his analysis. We will here summarize these restrictions and indicate the reasons Turing mentioned for adding them.

B.1 *Boundedness condition on the number of symbols that can be printed, i.e. finiteness of the alphabet.* There is a fixed upper bound to the number of distinct symbols that can be printed. As Turing remarks: "*If we were to allow an infinity of symbols, then there would be symbols differing to an arbitrarily small extent.*" ([Tur37], p. 249). Furthermore, if we do not add this restriction, it becomes impossible to recognize a symbols at one glance.

B.2. *Boundedness condition on the number of cells or symbols scanned.* There is a fixed bound on the number of contiguous cells (or their contents) the computor can take in when he is deciding what to do. This restriction is added, since there is a limit for us humans for directly recognizing a given sequence of symbols. Turing gives the example that we cannot determine at a glance whether 9999999999999999 and 9999999999999999 are identical. There is also a further reason for this restriction not explicitly mentioned by Turing: we humans can only perceive a finite space at one and the same moment. If we are reading a text, we have to move our eyes to reach certain points of the text.

B.3. *Boundedness condition on the number of states.* There is a fixed bound on the number of "states of mind" of the computor. Turing's reason for adding this restriction is: "*If we admitted an infinity of states of mind,* *some of them will be "arbitrarily close" and will be confused."* This argument is thus similar to that for **B.1.**.[13]

L.1. *Locality condition on the number of symbols that can be changed.* Only symbols of observed configurations can be changed, and only one at a time. Turing does not provide a real reason here, but it seems only normal that we only change one thing at a time when calculating on a piece of paper.

L.2. *Locality condition on the size of the move to left or right.* Each of the observed squares must be within a bounded distance of an immediately previously observed square, i.e. there is a bound on the number of squares you can move over in one step. Turing provides no argument, but it is a very reasonable condition, since one will most probably never jump from page 50 to page 100 in making a calculation. Furthermore, if this locality condition would not be made, one could make a move to infinity.

For Turing it was very important that the operations of the computor "*are so elementary that it is not easy to imagine them further divided.*" ([Tur37], p. 250). In this respect the operations of the computor can be considered as *atomic acts.* On the basis of this analysis of a human computor, Turing deduces an abstract computor, which should be capable of two basic operations: (1) it must be able to change a symbol in one of the observed squares, and (2) must be able to move from one of the squares observed to another square, within a certain number of squares of the previously observed squares. Since this machine must also be able to change its state of mind, the most general simple operations Turing concludes for is the combination of (1) rsp. (2) with the operation of changing the state of mind. Finally, Turing adds that every such operation performed is completely determined by the the state of mind and the observed symbols. Sieg [Sie94, SB96, Sie97] calls this the *determinacy condition* (**D**).
Given the restrictions one must take into account in observing the process of a man calculating, and the abstract computor deduced on the basis of these restrictions, the idea of identifying human computing with machine computing

[13] Discussions on the "finite-states" hypothesis can e.g. be found in [Web80] and [Kle87].

almost naturally follows. As Turing writes ([Tur37], p. 251):

> We may now construct a machine to do the work of this computer [the abstract computor]. To each state of mind of the computer corresponds an "m-configuration" of the machine. The machine scans B squares corresponding to the B squares observed by the computer. In any move the machine can change a symbol on a scanned square or can change any one of the scanned squares to another square distant not more than L squares from one of the other scanned squares. The move which is done, and the succeeding configuration, are determined by the scanned symbol and the m-configuration. The machines just described do not differ very essentially from computing machines as defined [earlier in the paper, i.e., Turing machines], and corresponding to any machine of this type a computing machine can be constructed to compute the same sequence, that is to say the sequence computed by the computer.

In having constructed an abstract computor on the basis of the several restrictions deduced, and having argued how this abstract computor can in fact be replaced by a computer, which is basically identical to a Turing machine,[14] Turing has thus provided a very direct and convincing argumentation for the following thesis:

> **Turing's Thesis.** *Anything that can be calculated by a human being, can be computed by a Turing machine (and conversely)*

Turing's thesis has been posed in several different forms in the literature, by separating between the several steps in Turing's reasoning. For example, both Sieg and Soare give a different form of Turing's thesis. To Sieg [SB96], Turing's central thesis is:

> **Turing's Central Thesis [Sieg].** *Any mechanical procedure can be carried out by a computor satisfying conditions* ***B.2., B.3., L.1., L.2., D***.

[14] Unlike the simplification from computor to abstract computor, and from abstract computor to computer, this simplification (from computer to Turing machine) can be proven.

On the basis of this thesis, Sieg then concludes for what he calls Turing's theorem:

> **Turing's Theorem [Sieg].** *Any number theoretic function that can be calculated by a mechanical computor can be computed by a Turing machine.*

Soare [Soa96] accepts Sieg's statement of "Turing's theorem'" and concluded that Turing's thesis can be reduced to the thesis Sieg called "Turing's central thesis, although he slightly reformulated it by replacing "mechanical procedure" by "functions that are considered intuitively to be calculable".
In his [Gan88], Gandy gives three different forms of Turing's thesis, but does not identify them as theses, but as theorems. The first statement of the "theorem" is very similar to our statement of Turing's thesis:

> **Turing's theorem I [Gandy].** *Any function that can be calculated by a human being can be computed by a Turing machine.*
>
> **Turing's theorem II [Gandy].** *Any function which can be calculated by a human being following a fixed routine is computable.*[15]
>
> **Turing's theorem III [Gandy].** *Any function which is effectively calculable by an abstract human being following a fixed routine is effectively calculable by a Turing machine – or equivalently, effectively calculable in the sense defined by Church – and conversely.*

It is remarkable that, although Turing machines were shown to be equivalent to general recursive functions and λ-definability, it is Turing's thesis, not Church's (nor any thesis based on a formalism shown to be equivalent to one of these formalisms) that has given rise to a variety of different statements.[16]
As we will see in Sec. 4.3 it has been Turing's thesis that lies at the basis of what

[15] It is important to point out that when Gandy uses the word "computable" it refers to any of the notions equivalent to Turing computability, like e.g. λ-definability.

[16] As for the several formulations of Turing's thesis in terms of Turing's theorem given here, they will be discussed in more detail in Sec. 3.2

is now known as the physical Church-Turing thesis, and forms the main target of many a scientist trying to go beyond the Turing limit. In a way, it is not surprising that one usually talks about going beyond the Turing limit instead of say the Post or Church limit. Neither Post's nor Church's thesis, makes a connection with something as "worldly" as a machine. This is rather logical since neither of them started from a direct analysis of the idea of computing, but only formulated their theses on the basis of formalisms that already existed and used in a different context, i.e., it were the formalisms themselves that convinced them. Turing's machines on the other hand, were constructed by starting from an analysis of the process of a man calculating. In this sense, there is a direct link between Turing machines and the physical world we compute in. But this is not the only connection. The fact that Turing uses the word "machine", instead of calculus or form, makes this connection even stronger. Although at that time there were no computers as we know them now, the notion of a machine was very well known. Even if Turing machines are a very special kind of abstract machines, their description is very physical, using a *tape*, a head that *moves* over the tape, *writes* and *recognizes* symbols, and furthermore works with *states of mind* describing what the machine should do. In this respect, Turing machines are much more connected to the notion of *effectiveness* interpreted in the more physical sense of *mechanizable*.

Given its close connection to our everyday life, especially now in the era of computers, it should thus not come as a surprise that Turing's thesis is regarded as the most convincing one, and thus also as the one that should be beaten. Turing was right in describing his analysis as an argument making a direct appeal to intuition. This is only affirmed by the hundreds of philosophical papers and books that make Turing machines the central concept. However, now that we have a formalism considered to capture the intuition, it might be fruitful to turn the argumentation upside down and to ask how formalisms further removed from intuition can help to alter our intuitions. We will develop this idea in Sec. 3.3.

3.1.3 “Finite Combinatory processes. formulation 1”

In Sec. 2.2.5 it was shown that Post had already formulated his thesis and proven certain decision problems unsolvable relative to this thesis in 1921. We will first discuss these results in more detail. Starting from Post’s critique with respect to his own thesis, we will then finally be ready to discuss Post’s 1936 paper [Pos36]. We already know how Post came to the realization of the generality of his normal form and we will not repeat the significance of, on the one hand, tag systems and, on the other hand, the normal form theorem, for his at that time “unorthodox” ideas.

Contrary to Church and Turing, Post does not refer to the intuitive notion of effective calculability or computability in his thesis, but that of a generated set. Clearly, this notion is further removed from everyday life and thus, in itself, far less intuitively appealing than the notion of a computation. Indeed, while everybody is used to the idea of a calculation – it was and still is part of elementary education – far from everybody has a concrete notion of sets, let alone, generated sets.[17] Still, graduate students in mathematics at that time (up to today) and, especially, the logicians who were (and are) acquainted with formalisms like *Principia* or Cantor’s work, must have had a more concrete intuition of the notion of a generated set.

Post’s statement of his thesis

In Post’s case, the fact that he stated his thesis in terms of generated sets rather than calculability, is not surprising since, basically, he had been working with systems generating sets of sequences of letters. Given the mathematical generality of *Principia*, it was the realization that *Principia* might be reducible to a system in normal form that formed the decisive step for him to formulate his thesis ([Pos65], p. 385):

> In view of the generality of *Principia Mathematica*, and its seeming inability to

[17] The generation I belong to has a more concrete notion of set, since introductory courses on sets were included in the elementary and secondary school curriculum, under the influence of Bourbaki. In the meantime it is no longer a standard part of most curricula (at least as far as I know, here in Europe).

> lead to any other generated sets of sequences on a given set of letters than those given by our normal systems, we are led to the following generalization.

The generalization pointed at in the quote, was first called Post's thesis by Martin Davis [Dav82]. Post has given the following statement of his thesis:

> **Post's Thesis.** *Every generated set of sequences on a given set of letters $a_1, a_2, ..., a_\mu$ is a subset of the set of assertions of a system in normal form with primitive letters $a_1, a_2, ..., a_\mu, a'_1, a'_2, ..., a'_\mu$, i.e., the subset consisting of those assertions of the normal system involving the letters $a_1, a_2, ..., a_\mu$.*

Fundamental for the formulation of this thesis in terms of normal form was Post's normal form theorem, through which any canonical form could be reduced to a special canonical form, i.e., a system in normal form.

Unsolvable decision problems for normal systems

Having started from the idea to develop the most general form of logic and ultimately mathematics, in order to prove that the whole body of mathematics is solvable, Post had now found such general form of logic and mathematics. However, having realized, after his experience with tag systems, that such decision procedure might not exist, together with the realization of the generality of his normal form, which was very closely connected to his form of tag, and the insight that he could apply a diagonalization procedure, Post now decided for the reversal of his entire program and concluded that the decision problem for the class of normal system is unsolvable, in that there exists "*no finite method which would uniformly enable us to tell of an arbitrary normal system and arbitrary sequence on the letters thereof whether that sequence is or is not generated by the operations of the system from the primitive sequence of the system.*" ([Pos65], pp. 386–387).

In his *Account of an anticipation* Post added an *Outline of a minimum mathematical development,* in which he proves the unsolvability of the finiteness problem for systems in normal form through diagonalization. In this outline, Post proved through diagonalization on an enumeration of all normal systems, that there exists a set of "a-sequences", strings on the alphabet {a}, called the N-set, that cannot be generated by a normal system, and is thus a set that cannot be generated in general. To put it in different terms, it is a non-computable set. The following intermezzo explains the proof.

Let us first recall the definition of a system in normal form. Let $\Sigma = \{a_1, a_2, ...a_\mu\}$. A system in normal form is then defined by one initial sequence $A = a_{i_1} a_{i_2} ... a_{i_\lambda}$ (the axiom) over the alphabet Σ, and a finite set of production rules of the form:

$$g_i P \text{ produces } P g_i'$$

where g_i and g_i' are finite sequences over Σ. A normal system is the set of sequences resulting from the iterated application of these operations starting with the initial sequence. Post makes no differentiation between normal systems which only differ from each other in the letters used. In this respect, these letters can always be the first μ symbols of an infinite sequence of letters, like e.g. the first μ positive integers.

Post then goes on to show that the set of all normal systems can itself be ordered in an infinite sequence, i.e., it can be enumerated. This is done by ordering the set of all possible bases, the initial sequence and the production rules, of normal systems. Post then defines a kind of coding, comparable to Gödel coding or Turing's coding through the D.N. and the S.D. Essential to Post's coding here is the *complexity* of a basis of a normal system. The complexity of a given basis is the total number of symbols appearing in the alphabet Σ, the initial sequence and the production rules, where each P is counted as a separate symbol. The different bases are then divided into classes according to their complexities, in order of increasing complexity. Each class is then further divided into subclasses according to the number of symbols of the alphabet, and correspondingly ordered. In the same way, each of these subclasses is divided into subclasses and ordered, according to the number of operations. Each of these subclasses is then divided again and ordered according to the ranks of $C = Ag_1 g_1' ... g_k g_k'$, the rank of a sequence being its length, with the resulting classes ordered according to the rank of the first of the sequences that differ in rank for two classes. In each of the resulting classes two bases are identical iff. their respective sequences C are

> identical.
> C is then interpreted as a number, by setting each letter to an integer, i.e. $a_1 = 1, a_2 = 2, ..., a_\mu = \mu$. Since the number of letters in the alphabet is the same for all bases in the same class, the bases within a given class can finally be ordered within each class according tot the number C represents. Using this ordering, it is possible to order the whole set of bases. Post called this ordering the *σ-ordering*. Given the convention that two normal systems that differ from each other only in the specific alphabet used, are considered identical, so that for any base, the letters from the alphabet are the first μ letters, it is clear that all normal systems have the same letter a_1 which is replaced by a. Then, consider the following set of sequences, involving only the letter a: The string $a...a$, with n a's (of rank n), *is* or **is not** in the set depending on whether it **is not** or *is* in the n-th normal system in the σ-ordering. This set is called the N-set. Then, there exists no normal system with the property that if its first letter is replaced by a, then the set of resulting sequences involving only the letter a is the N-set, i.e., there exists no formal system that can generate the N-set. This is true, since suppose there would be such a normal system in the σ-ordering, e.g. the m-th. This normal system, however, must differ from the N-set in at least one sequence, i.e., the sequence $aa...a$ with m a's, since, by definition, if this sequence is present in the normal system it cannot be in the N-set, and vice versa.

After having proven this result, Post states: "*As stated this theorem would be trivial were it not for the all embraciveness of normal systems.*" '([Pos65], p. 389). Indeed, this proof is only valid in as far as Post's thesis is valid. Earlier in the paper, Post already pointed out the significance of the non-effectiveness of the diagonalization: although it is possible to define the set N of a-sequences, this does not result in a counter-example, since one can only yield a true counter-example if one can set up a system of combinatory generation that effectively generates the set.[18]

Post then deduces several other "(theorems)", putting them between brackets because he did not provide the details of the proofs. These (theorems) concern:

[18] For the exact quotes, the reader is referred to Sec. 2.2.4.

1. A (theorem) stating that there exists a *complete normal system K* and a correspondence (encoding) *C* such that for each normal system and enunciation thereof, there is one and only one enunciation in *K* by correspondence *C*, such that an enunciation in *K* is asserted (generated) iff. the corresponding normal system is such that the enunciation is an assertion of that normal system. The normal system *K* is thus a normal system that generates all and only those assertions generated by any other normal system. Post later added in a footnote that this normal system *K* corresponds to Turing's universal machine.

2. A (theorem) stating that there exists no *finite-normal-test* for the complete normal system K. Given a normal system *M*. Then there exists a finite-normal-test for *M* if there exists a normal system *M*' such that among the letters of *M*' are all the letters of *M*, and in addition, among possibly others, a primitive letter *b*, such that if *P* is an assertion of *M*, *P* is also an assertion of *M*', while *bP* is an assertion of *M*' if *P* is not an assertion of *M*. Although Post of course did not use the terminology then, we can reformulate the existence of a finite-normal-test for a given normal system *M*, by stating that *M* is recursive. The (theorem) of the non-existence of a finite-normal-test for the complete system *K* would later be proven in more detail, and stated in more exact terms, by Post [Pos44]. He proved that there exists a complete set *C*, which is very similar to the normal system *K*, proving that the set *C* is a recursively enumerable set that is non-recursive, i.e. the complement of the set is not recursively enumerable.

3. A (theorem) stating that no normal-deductive-system is complete, there always existing a normal system *S* and enunciation *P* thereof such that *P* is not in *S* while b(S, P) is not in the normal-deductive-system. A normal-deductive-system *L* is a normal system that is such that for any normal system *S*, if *P* is an assertion of *S*, (S, P) is in *L*, while if *P* is not in *S*, $b(S,P)$ is in *L*. In [Pos44], Post would prove what he called a *Gödel in miniature* theorem, by using a reasoning that seems to go back to this (theorem).

4. A (theorem) stating that for any given normal-deductive-system there

> exists another one which proves more theorems than the first (*to put it roughly*, Post added). Also this (theorem) was stated in more exact form in Post's [Pos44].

After his Procter fellowship Post further investigated the results he considered to be unfinished. This is very clear from several of the footnotes of his *Account of an anticipation*. However, he was in a far from perfect situation to do his research. As is pointed out by Davis [Dav94]:

> Until 1935, he was unable to obtain a regular academic position, making his living, for the most part, by teaching in the New York high school system.

Post could not devote his whole time to research, not only because he did not have a secure academic position, but also due to his manic-depressive illness. This situation did not stop him from doing further research, on the contrary. One of the goals he set himself after his Procter fellowship was to provide a more complete analysis of the intuitive notion of generated sets. As we already know, to Post for his thesis to obtain its full generality, "*an analysis should be made of all the possible ways the human mind can set up finite processes to generate sequences.*" ([Pos65], p. 387) This quote is very similar to what Turing had pointed out as the real question to be asked in searching for the right formalism to capture the intuition.

The first traces of such analysis can be found in the appendix Post added to his *Account of an anticipation*, containing fragments from his notes and diary. The last entry of the appendix is dated February 4, 1922. We do not want to discuss the content of the appendix here, given its fragmentary character. It should be noted though that it mentions some of the restrictions Turing deduced out of his analysis of the process of a man computing, like the use of a finite number of states and symbols. That Post indeed made a start with such an analysis is clear from some quotes from his *Account of an anticipation*. We must mention some of them here, since they show that Post indeed started with such an analysis and, as a consequence, that his formulation 1 did not simply come out of the blue. We will only give two quotes here. The quote already mentioned, emphasizing the significance of such analysis, can be regarded as a third such quote.

After having stated the significance of the analysis, Post writes ([Pos65], p. 387):

> The beginning of such an attempt [the analysis] will be found in the Appendix.

In the introduction to the Appendix Post explains ([Pos65], pp. 394–395):

> While the formal reductions of Part I [the reduction from canonical form *A* to *B* to *C* to normal form] should make it a relatively simple matter to supply the details of the development outlined [i.e. the (theorems)] that development owes its significance entirely to the universal character of our characterization of an arbitrary generated set of sequences [...] Establishing this universality is not a matter for mathematical proof, but of *psychological analysis of the mental processes involved in combinatory mathematical processes* [m.i.]. [...] Actually, we can present but fragments of the *proposed analysis of finite processes* [m.i.]. [...] This theme [the idea that there exist problems we humans cannot solve] will protrude itself ever so often in *our immediate task of obtaining an analysis of finite processes* [m.i.].

As is clear, Post understood that his thesis was the fundamental result underlying his (theorems), and it was thus necessary to further establish the "universality" of this thesis through a deeper analysis of all the possible ways the human mind can set up finite processes. He clearly made a start with such an analysis. In fact, it would have been surprising if he would not have made an attempt for such an analysis, since he understood its fundamental significance.

formulation 1

Fifteen years later Post's paper *Finite Combinatory processes. formulation 1* [Pos36] was published. By then, Gödel had already published his fundamental 1931 paper. Post knew of Church's results, but was unaware of Turing's paper. Maybe the most remarkable thing about the formalism described in this paper, a formalism which must have been influenced to some extent by Post's earlier analysis announced in the Appendix of all the possible finite processes the human mind can set up to generate sets, in the end resulted in a formalism that is

almost identical to Turing's description of his machines. Post's paper however did not contain a proof of an unsolvable decision problem nor the description of a universal machine. Neither does it contain any explicit argumentation in the sense of Turing's: the paper is only three pages long. As a consequence, the paper is often only mentioned in a footnote, if it is mentioned at all, and has, to our mind, been handled in a stepmotherly way in many of the literature.
Still, the results from the paper are very important in several different ways. First of all, the fact that Post's analysis resulted in a formalism that is very similar to Turing machines adds strength to the idea that if one starts from an analysis of a an intuitive notion similar to the one considered by Turing, one indeed ends up with something like Turing machines. Secondly, as we will argue here, although Post never really makes very explicit what kind of intuitive notion exactly he wants to capture with his *formulation 1*, it is clear from the paper that he intends to formalize the notion of *solvability*. Thirdly, and this is the reason why the paper is most often mentioned, Post did not regard any such identification as a definition, and explicitly criticized Church's calling his identification between effective calculability and general recursiveness (λ-definability) a definition.
We will now first describe *formulation 1*, or, as it is also sometimes called, Post machines, in an intermezzo. [19]

> There are two basic concepts involved in formulation 1: the *symbol space* in which the work leading to a solution is to be carried out and the *set of directions* which direct operations in the symbol space and determine the order in which the directions have to be performed. The symbol space is a two way infinite sequence of *boxes*. The *worker* or problem solver can move and work in the symbol space, and is capable of being in, and operating in but one box at a time. A box can have two possible conditions: empty or marked with one symbol, e.g. "|".
> One of the boxes is called the starting point.
> Now what can our worker do, what is he capable of? The worker's work is limited to the following *primitive acts*:

[19] See for example the booklet by Uspensky called *Post's machine* [Usp83], where Uspensky describes how these machines can be used in elementary and secondary school to make school children familiar with some of the basics of computers and programming.

(1) Marking the box he is in (assumed empty).

(2) Erasing the mark in the box he is in (assumed marked).

(3) Moving to the box on his right.

(4) Moving to the box on his left.

(5) Determining whether the box he is in, is or is not marked.

The worker's acts are ordered through a set of directions, to remain unaltered once the worker has started with his work, i.e., the set of instructions is fixed. Every set of directions is headed by:

- Start at the starting point and follow direction 1.

The set always consists of a finite number of directions numbered $1,2,3,...,n$. The i-th direction ($i \in \mathbb{N}^+$) always has one of the three following forms:

(A) Perform operation O_i ($O_i = (1), (2), (3)$ or (4)) and then follow direction j_i.

(B) Perform operation 5, and according as the box is marked or not marked follow direction $j_{i'}$ or $j_{i''}$.

(C) Stop.

Post's formulation 1 has a very clear resemblance to Turing machines. There is, however, one significant difference. Whereas Turing uses the concept of an abstract computing machine, and is thus closer to physical computers, Post's description is in terms of a list of instructions in a formal language, and is thus closer to computer programs. So to say, Turing machines are closer to hardware, Post machines are closer to software. Despite this difference, the fact that both Post's and Turing's formulations are very similar to each other, in having started from the analysis of an intuitive concept, rather than from an existing formalism, shows that such analyses gives rise to formulations which are far closer to computers and computer languages than say normal systems or λ-calculus.

As is pointed out in [Dav94], Post was not satisfied with the analysis of an algorithmic process in terms of general recursiveness or λ-definability:

> Believing that the Herbrand-Gödel notion of general recursiveness and the Church-Kleene notion of λ- definability were both lacking in that neither constituted a "fundamental" analysis of the notion of algorithmic process, Post proposed as suitably "fundamental" the operations of marking an empty "box" or erasing the mark in a marked box.

In his [Pos36] however, the notion of an algorithmic process, or other similar notions such as "algorithmic procedure", "calculability" or "computability", are never mentioned by Post in relation to his formulation 1. The notion "effective calculability" is mentioned only once, at the end of the paper, but only with respect to Church's thesis. The notion Post does refer to quite often in the context of formulation 1 is *solvability*. The goal of formulation 1 indeed seems to have been to formally capture what exactly is meant with a general method to solve any decision problem which is intuitively considered solvable ([Pos36],p. 103):

> We have in mind a *general problem* consisting of a class of *specific problems*. A solution of the general problem will then be one which furnishes an answer to each specific problem.

After the description of such a solution [i.e. formulation 1], Post defined a whole set of notions in terms of solvability of a decision problem, thus making more explicit the identification between solvability and his formalism by adding several definitions. These notions are:

Applicability A set of directions is called applicable to a general problem, if in applying it to any specific case of the problem, instruction (1) is never ordered when the box the worker is in is already marked, and (2) is never ordered when the box is unmarked.

Finite 1-process A set of directions is considered as setting up a finite 1-process relative to a given general problem if it is applicable to the problem and if the process it sets up terminates for each specific case of the problem.

1-solution A finite 1-process is a 1-solution of a general problem if the answer it gives to each specific case of the problem is always correct.

1-given A problem is *1-given* if a finite 1-process can be set-up which, when applied to the class of positive integers symbolized in a certain way in the symbol space, yields in a one-to-one fashion the class of specific problems constituting the general problem. This 1-givenness can be compared to Gödelnumbering: the possibility to translate a problem to numbers and vice versa.

Given Post's purpose of the note stated at the beginning of the paper, i.e., to formulate a generalized form of what is meant with a solution to a given decision problem, his description of formulation 1, and the further explication of the identification through the terms defined above, we have deduced the following, second thesis, by Post:

> **Post's thesis II**. *A decision problem is considered intuitively solvable iff. the problem is 1-given and one can set-up a finite 1-process which is a 1-solution to the problem.*

That Post not only considered the problem of formalizing the intuitive notion of generated set, or, in later work, of effective calculability and computability is also reflected in some of his later papers. For example, in the paper in which one finds the statement of what is now known as the *Post Correspondence Problem* and a proof of its recursively unsolvability, he writes: ([Pos46], p. 364)

> We proceed to prove [...] that in its full generality *the correspondence decision problem is recursively unsolvable*, and hence, no doubt, unsolvable in the intuitive sense.

In his seminal paper on recursively enumerable sets of positive integers he furthermore notes ([Pos44], p. 289):

> [...] whether those [decision] problems are, or are not, solvable in the intuitive sense would be equivalent to their being, or not being, recursively solvable in the precise technical sense.

Recursively unsolvability however is no longer defined in terms of his formulation 1, but in terms of his normal form.

Post's paper ends with a paragraph in which he criticizes Church's statement of his identification in the form of a definition. After having stated that he expects formulation 1 to turn out equivalent to general recursiveness – an expectation soon to be proven true by Turing through the equivalence between Turing computability and λ-definability – Post immediately adds that its purpose is not simply to ([Pos36], p. 105):

> [...] present a system of a certain logical potency but also, in its restricted field, of psychological fidelity. In the latter sense wider and wider formulations are contemplated. On the other hand, our aim will be to show that all such are logically reducible to formulation 1. We offer this conclusion at the present moment as a *working hypothesis*. And to our mind such is Church's identification of effective calculability with recursiveness. [...] The success of the above program would, for us, change this hypothesis not so much to a definition or to an axiom but to a *natural law*. Only so, it seems to the writer, can Gödel's theorem concerning the incompleteness of symbolic logics of a certain general type and Church's results on the recursive unsolvability of certain problems be transformed into conclusions concerning all symbolic logics and *methods of solvability* [m.i.].

Post considered his formulation 1 as a system of psychological fidelity: generalizing Gödel's incompleteness theorem or Church's proof of the unsolvability of certain decision problems to conclusions concerning all symbolic logics and *methods of solvability* depends on the "faith" one can have in identifications such as that proposed by Post. In that sense, he suggests to contemplate as wide a variety of formulations as possible, each of which should be shown to be reducible to formulation 1. Indeed, at the time Post wrote this paper he considered his conclusions concerning the 'power' of formulation 1, merely as a *working hypothesis*. It is only if the method of finding wider and wider formulations reducible to formulation 1 has proven its worth, that the hypothesis can be considered as a *natural law*.

It is exactly at this point that Post criticizes Church's "definitional identification". After having noticed in footnote 8 that Church's, Kleene's and Rosser's work already carries the identification beyond the working hypothesis stage, in having shown that λ-definability and general recursiveness are equivalent, Post continues ([Pos36], p. 105):

> But to mask this identification under a definition hides the fact that a fundamental discovery in the limitations of the mathematicizing power of *Homo Sapiens* has been made and blinds us to the need of its continual verification.

Indeed, Post does not accept Church's definitional identification as the correct interpretation of such an identification. Rather it should be regarded as a hypothesis or law of nature, to be continually verified, and is thus subject to inductive reasoning. In a letter to Gödel, dated October 30, 1938 Post again emphasizes the significance of the hypothetical character of e.g. Church's thesis ([Göd03b], p. 171):

> the absolute unsolvability of [a] problem has but a basis in the nature of physical induction at least in my work and I still think in any work.

Considering identifications, such as those proposed by Post, as a hypothesis, indeed implies that an unsolvable decision problem can only presumptively be considered absolute: it is true only in supposing the validity of the identification. In the quote Post also underlines the fact that, to his mind, this is not only true for his but also for the work of others. But why did he add this small comment? To understand this we must have a closer look at how Church reacted on Post's criticism.

3.2 On the status of the identification. Definition, (Hypo)Thesis, Law, or Theorem?

In this section we will see how Church reacted on Post's criticism by writing a rather harsh review of Post's paper. Starting from this review and the review Church wrote of Turing's paper, we will discuss several possible interpretations and evaluations of the proposed identifications, starting from Church's, Turing's and Post's own ideas in this context.

3.2.1 Church's reviews of Post's and Turing's paper

In volume 2, number 1, 1937 of the newly founded *the Journal of Symbolic Logic* Church wrote two reviews, one of Turing's paper and one of Post's paper. About Turing's paper Church stated [Chu37b]:

> [Turing] proposes as a criterion that an infinite sequence of digits 0 and 1 be "computable" that it shall be possible to devise a computing machine, occupying a finite space and with working parts of finite size, which will write down the sequence to any desired number of terms if allowed to run for a sufficiently long time. As a matter of convenience, certain further restrictions are imposed on the character of the machine, but these are of such a nature as obviously to cause no loss of generality – in particular, a human calculator, provided with pencil and paper and explicit instructions can be regarded as a kind of Turing machine. It is thus immediately clear that computability, so defined, can be identified with (especially, is no less general than) the notion of effectiveness as it appears in certain mathematical problems (various forms of the Entscheidungsproblem, various problems to find complete sets of invariants in topology, group theory, etc., and in general any problem which concerns the discovery of an algorithm). [...] As a matter of fact, there is involved here the equivalence of three different notions: computability by a Turing machine, general recursiveness in the sense of Herbrand-Gödel-Kleene, and λ-definability in the sense of Kleene and the present reviewer. Of these, the first has the advantage of making the identification with effectiveness in the ordinary (not explicitly defined) sense evident immediately – i.e. without the necessity of proving preliminary theorems. The second and third have the advantage of suitability for embodiment in a system of symbolic logic.

Before discussing this review, it is interesting to contrast it with Church's review of Post's paper [Chu37a]:

> [Post] proposes a definition of "finite 1-process" which is similar in formulation, and in fact equivalent, to computation by a Turing machine (see the preceding review). He does not, however, regard his formulation as certainly to be identified with effectiveness in the ordinary sense, but takes this identification as

> a “working hypothesis” in need of continual verification. To this the reviewer would object that effectiveness in the ordinary sense has not been given an exact definition, and hence the working hypothesis in question has not an exact meaning. To define effectiveness as computability by an arbitrary machine, subject to restrictions of finiteness, would seem to be an adequate representation of the ordinary notion, and if this is done the need for a working hypothesis disappears.

As is clear from reading both reviews, Church is very positive about Turing’s paper and rather sharp with respect to Post’s paper. Why is this the case, especially given the fact that the formalisms presented by Turing and Post are very similar?

Church evaluated Turing’s thesis very positively. To his mind, it has the advantage over his thesis, “*of making the identification with effectiveness in the ordinary (not explicitly defined) sense evident immediately.*” The reason for this has to do with Turing’s above discussed analysis of the process of a man calculating, i.e., the argument making a direct appeal to intuition, leading to the identification of human computing with Turing machines. As Church states: “*It is thus immediately clear that computability, so defined, can be identified with [...] the notion of effectiveness as it appears in certain mathematical problems.*”

Church was and is not the only one who found Turing’s thesis the most adequate one, in the sense that his analysis of the process of a man computing almost naturally leads to Turing machines, while e.g. the identification between λ-definability or general recursiveness with effective calculability, is not as natural. But before further discussing these matters, it is important to compare this opinion with Church’s criticisms on Post’s paper.

Church’s main critique is, very clearly, a reaction on Post’s criticism, and opposes the possibility of calling his *definition* a *working hypothesis.* He gives two main objections to this. First of all he finds that Post has not given an exact definition of the ordinary notion of effectiveness so that the ‘working hypothesis’ is ambiguous. Secondly, if one does provide an *adequate* representation for the notion of effectiveness it should simply not be regarded as a working hypothesis but as a definition.

As far as the first critique is concerned, Church is either pointing at the inexactness of formulation 1 itself, or, the fact that Post indeed did not make very explicit the kind of identification he intended. Given the similarities between formulation 1 and Turing machines, it is improbable that Church is pointing at the inexactness of the formulation, so we think that Church is criticizing the vagueness of the identification. This criticism is, to a certain extent, understandable, but clearly cannot count as any serious objection to Post's calling the identification a working hypothesis, since for Post this not only applies for the identification he proposed, but for any other such identification. The second objection, that there is no need for a 'working hypothesis' once one has given an adequate definition of effectiveness, and here Church refers to Turing's not to his own, only seems to mean that Church preferred definition to further argument as is stated in [Gan88].[20]

This small "quarrel" between Church and Post becomes the more remarkable in the light of the significance Church attached to "presumptive evidence" in his work preceding his 1936 paper. As we know from Sec. 2.3 he frequently underlined the significance of "empirical evidence" for supporting the consistency of a given system of symbolic logic, in case one has no proof of the system's consistency. Although Kleene and Rosser had shown the fundamental problems that are always involved in using more heuristic arguments, by showing that Church's set of postulates is inconsistent, it was again the more empirical fact that any function Church, Kleene and Rosser could think of could be λ-defined, that led Church to the formulation of his original thesis of identifying effective calculability with λ-definability.

It is not completely clear how important the proof of the equivalence between λ-definability and general recursiveness has been for Church's first public announcement of his thesis (see Sec. 2.3), but, as is clear from his above discussed 1936 paper [Chu36c], he considered this equivalence as an argument further supporting the validity of his thesis in terms of general recursiveness and λ-definability. It was exactly this *argument by confluence* Post had in mind in order to get the identification beyond the working hypothesis stage, this argu-

[20] After having mentioned Church's review on Turing's paper, Gandy notes: "*But Church is slightly less dogmatic in his review of Post's paper, preferring, apparently, definition to further argument, and defending this move against Post's criticism.*" ([Gan88], p. 85)

ment being understood as 'presumptive evidence', to use Church's words, for the validity of the identification. Did Church indeed interpret the evidence offered by the equivalence proof in the sense Post understood it (and in the sense Church used such evidence in his earlier work), or was it merely a theoretically important fact, used to state a more natural definition, having no link whatsoever with his needing more evidence for λ-calculus' computational powers? Whatever the right answer might be, Church's rather strong reaction against Post's critique remains to our mind rather strange in the light of the work preceding this review.

As is clear from the two reviews, Church did not want to regard any identification, such as the one he proposed, as a working hypothesis. To Church it was more appropriate to talk about such identification in terms of definition, where Turing's "definition" was considered as the most adequate because the analysis of a man computing almost naturally leads to the formulation of Turing machines. As was said, Church is certainly not the only one who regards Turing's identification as the most adequate one.

3.2.2 On the adequacy of Turing's identification: From definition to theorem.

In Sec. 2.3 we quoted from a letter Church wrote to Kleene, describing a conversation between Church and Gödel (presumably to be situated in early 1934). In this letter, Church describes that Gödel regarded Church's proposal to identify effective calculability with λ-definability as "thoroughly unsatisfactory", while, to Gödel's mind, the identification with recursion could not be made satisfactorily, "except heuristically". In this sense, Gödel's opinion at that time was not that far removed from Post's some years later.

It was only after having read Turing's paper, that Gödel became convinced of the possibility of an adequate identification. As he writes in a Postscriptum to "On undecidable propositions of formal mathematical systems" ([Göd65], p. 72):

> In consequence of later advances, in particular of the fact that, due to A.M. Tur-

> ing's work, a precise and unquestionably adequate *definition* [m.i.] of the general concept of formal system can now be given [...] Turing's work gives an analysis of the concept of "mechanical procedure" (alias "algorithm" or "computation procedure" or "finite combinatorial procedure"). This concept is shown to be equivalent with that of a "Turing machine". A formal system can simply be defined to be any mechanical procedure for producing formulas, called provable formulas (and likewise vice versa), provided the term "finite procedure" [...] is understood to mean "mechanical procedure". This meaning, however, is required by the concept of formal system, whose essence it is that reasoning is completely replaced by mechanical operations on formulas.

It was Turing's analysis of the concept of a mechanical procedure, as a means to define a formal system in terms of a mechanical procedure proving formulas, and the fact that, as Gödel has it, mechanical procedure is *shown to be equivalent* to Turing machines, that results in a precise and unquestionably adequate *definition* for (finitary) formal systems.
He remained rather negative however with respect to identifying calculability with recursiveness or λ-definability ([Göd65], p. 72):

> As for previous equivalent definitions of computability, which however, are much less suitable for our purpose, see A. Church [Chu36c] [...].

To Gödel, it is only on the basis of the definition of "finite procedure" in terms of "mechanical procedure" that the identification between effective calculability and general recursiveness can be regarded adequate.[21]
As was the case for Church, Gödel also regarded the identification made by Turing as a definition – *not* as a hypothesis or a law – the adequacy of which is unquestionable. Again, this is due to Turing's argument making a direct appeal to intuition, through his analysis of the process of a man computing, leading to the Turing machine concept.
This result must have greatly impressed Gödel. In his contribution to the Prince-

[21] This is expressed in the Postscriptum: "*[...] if "finite procedure" is understood to mean "mechanical procedure", the question raised in footnote 3 [i.e. that any function computed by finite procedure, is recursive] can be answered affirmatively for recursiveness [...]*" ([Göd65], p. 73)

ton bicentennial conference, he expressed the "great importance" of both general recursiveness and Turing computability, since, these concepts have made it possible to provide absolute definitions for a significant epistemological notion ([Göd46], p. 84):

> Tarski has stressed in his lecture (and I think justly) the great importance of the concept of general recursiveness (or Turing's computability). It seems to me that this importance is largely due to the fact that with this concept one has for the first time succeeded in giving an absolute definition of an interesting epistemological notion, i.e., one not depending on the formalism chosen. In all other cases treated previously, such as demonstrability and definability, one has been able to define them only relative to a given language, and for each individual language it is clear that the one thus obtained is not the one looked for. For the concept of computability however, although it is merely a special kind of demonstrability or decidability the situation is different. By a kind of miracle it is not necessary to distinguish orders, and the diagonal procedure does not lead outside the defined notion.

Indeed, the absoluteness of the definition lies in its independence of the specific formalism used. This not only concerns the fact that so many different formalism had been shown to be equivalent, but, even more, also concerns the fact that for any formal system S of order or type i, what is computable in that system is already computable by e.g. Turing machines. This was more explicitly expressed in Gödel's [Göd36], p.83:

> It may also be shown that a function which is computable in one of the systems S_i or even in a system of transfinite type, is already computable in S_1. Thus, the concept "computable" is in a certain definite sense "absolute", while practically all other familiar metamathematical concepts (e.g. provable, definable, etc.) depend quite essentially on the system with respect to which they are defined.

The fact that both Church and Gödel accepted Turing's analysis as an *unquestionably adequate definition* for effectiveness shows how powerful Turing's argumentation actually is. In Section 3.1.2 we showed how several authors have interpreted several forms of Turing's identification as theorems. This further illustrates the cogency of Turing's analysis. As Gandy states ([Gan88], p. 82):

> Turing's analysis does much more than provide an argument for Church's thesis; *it proves a theorem.*

Gandy gave several versions of "Turing's theorem", and considers the proof of the "theorem" to be given by Turing's analysis. Sieg and Soare also claim Turing to have proven a theorem, however one rooted in what Sieg has called Turing's central thesis. Comparing both interpretations, Gandy's identification of Turing's thesis as a theorem is the strongest claim.

To Gandy's mind, Turing's "proof" of the identification between what can be computed by a human being and what can be computed by a Turing machine "*is quite as rigorous as many accepted mathematical proofs – it is the subject matter, not the process of proof, which is unfamiliar*" ([Gan88], p. 82). Although Gandy admits there are some gaps in Turing's "proof", they are not insurmountable. However, to state this identification as a theorem is, to our mind, far from unproblematic. Turing himself did not regard his analysis as a valid proof, but, on the contrary, stated that all arguments that can be given, are bound to be appeals to intuition. We think Turing is completely right here, since the identification concerns a formalization of an intuition. Indeed, one must ask here: how can one state that one has *proven* that there is a valid identification between an intuition and a formalism, what does it mean that one has *proven* that a given intuition is formalized? In a way this seems to imply that an intuition is something quite stable and universal.

It should also be noted that Gandy gave three different formulations of "Turing's theorem" he considers equivalent, without giving any clear argumentation for this equivalence. The last of these formulations, considered by Gandy as a restatement of Church's thesis, is in fact identical to the "theorem" Sieg (and Soare) deduced out of Turing's analysis, i.e., the identification between a comput*o*r satisfying several conditions and Turing machines. To make this claim more convincing Sieg interprets the conditions **L**, **B** and **D** as axioms. Still, interpreting the identification between computor and Turing machine as a theorem, remains, to our mind, as problematic as Gandy's first formulation of "Turing's theorem". The division between Turing's central thesis and the theorem, is based on the fact that Turing's analysis proceeds in three basic steps, of which only the last can be considered as a proof. As we know from Sec. 3.1.2 Turing first deduces an abstract computor, satisfying several conditions, starting from the analysis of the process of a man computing. Based on the several conditions, he then identifies the abstract computor with a machine, that should be capable to do the work of this computor. Since the description of this machine is basically that of a Turing machine, Turing concludes for the identification of "*a man in the process of computing [with] a [Turing] machine*".

However, the second step, identifying the computor with a machine has not been proven, at least, not as far as our notion of proof is concerned. It is true that the identification between a computor satisfying the several conditions and a computing machine is very convincing, and we think Sieg (and Soare) have correctly pointed out the significance of the identification between a man in the process of computing and an abstract computor satisfying the several conditions. Still we think both steps, the identification between a man computing and a computor, and, between a computor and a computing machine, are simply two basic aspects of Turing's *direct appeal to intuition* supporting his thesis. There is no reason in Turing's paper to assume the first identification as a thesis and the second as a theorem. In the end, identifying "state of mind" with "m-configuration" is a non-trivial step, except if one defines "state of mind" as "m-configuration". Nowadays we are so much used to the idea of Turing machines that this second identification seems more "obvious" than the first. This, however, does not give us a reason to regard it as a theorem, since a theorem needs proof.

Even if one identifies the several conditions as axioms, as Sieg does, these remain restricted to the description of the computor, not the machine, and the identification "computor = machine" does not follow as a theorem from these axioms, in the way $3+2=5$ follows from Peano's axioms of arithmetic.

To call Turing's analysis a proof of a given statement only makes sense if one understands proof rhetorically: the arguments given by Turing are indeed very convincing, in the sense that Turing managed to very correctly abstract several properties that are typical to the process of a man computing something on a piece of paper, almost naturally leading to the automatization of this process in a machine. But to make the claim that Turing's analysis results in a mathematical proof of a theorem, needs more argumentation and formalization.

Church and Gödel each regarded Turing's identification as an (unquestionably) adequate definition of effectiveness (or finite process, or computability, or any other similar intuitive notion). Turing himself also regarded his identification as a definition, as is clear from one of the quotes from Sec. 3.1.2.[22] To accept this (or any other similar) identification as a definition is, to our mind, a very reasonable point of view, since it allows us to prove certain theorems, like the unsolvability of the Entscheidungsproblem. It is indeed common to any branch of mathematics that one needs definitions, which can then help to further develop the theory.

Posing the identification as a definition, to a certain extent, *allows* to see the non-trivial character of introducing it: it is because one has given certain arguments that its introduction as a definition is made plausible. If, however, we interpret it as a theorem, one masks the non-trivial character of making the identification. Considering it as a theorem diverts attention from the fundamental aspect of the identification – the formalization of an epistemological notion – and, even more, its consequences both on a mathematical and a philosophical level.

3.2.3 The identification as (hypo)thesis or law.

Both Church, Gödel and Turing regarded the several identifications proposed in the thirties as definitions, accepted on the basis of certain arguments. Especially in Church's and Gödel's case, it is clear that the "direct appeal to intuition" argument, has been basic to consider Turing's thesis as the most adequate identification. It is this argument that furthermore led Gandy, Sieg and Soare to the conclusion that Turing's thesis can be (partly) stated in terms of a theorem. Nowadays, it seems to be general consensus that Turing's thesis offers the best identification, and it has become the dominant framework to discuss the more

[22] "*This usage [replacing "there is a general process for" by "there is a machine that will determine"] can be justified if and only if we can justify our definition of computability.*" ([Tur37], p. 134)

philosophical issues in this context.

Post and Kleene have both emphasized the hypothetical character of the several identifications. In this respect – although they each consider the kind of analysis as done by Turing, starting from a vague idea to deduce a formalism that captures it, as very important – they have each emphasized the significance of the other arguments supporting the several identifications, Turing's only being one of them.

We already know that Post opposed Church's interpretation of the thesis as a definition, since, to his mind, this definitional identification "*hides the fact that a fundamental discovery in the limitations of the mathematicizing power of Homo Sapiens has been made and blinds us to the need of its continual verification.*".

Church's criticism did not change Post's mind. Sec. 3.1.3 was ended with a quote from a letter Post wrote to Gödel in 1938 in which he explicitly states that the absoluteness of unsolvable decision problems has an inductive basis, i.e., the results are only valid in as far as the identifications they are based on are valid, identifications which can never be proven, but only be made more convincing the more evidence one finds for their general validity. To Post, this not only concerns his own work, but also that of others. This illustrates that for Post, Church's nor Turing's arguments, were convincing enough to regard any such identification as a definition, let alone a theorem.

As is explicitly stated by Post in his 1936 paper, the purpose of his formulation is not only to present a system of a certain logical potency, but also of psychological fidelity. In this sense, it is basic to consider wider and wider formulations, that should be shown to be logically reducible to his formulation 1. It is only if one can indeed find such wider and wider formulations reducible to formulation 1, that the identification can be carried beyond the *working hypothesis* stage, and become a *natural law*. This is indeed how Post later interpreted the several identifications that had already been established in the forties. In his [Pos44] he remarks in footnote 4:

> We still feel that ultimately "Law" will best describe the situation.

Sieg has contrasted Post's "method" of considering wider and wider formulations with Turing's ([Sie06], p. 60):

> It is methodologically remarkable that Turing proceeded in *exactly* the opposite way when trying to support the claim that all computable numbers are machine computable or, in our way of speaking, that all effectively calculable functions are Turing computable. He did not try to extend a narrow notion reducibly and obtain in this way additional quasi-empirical support; rather, he attempted to analyze the intended broad concept and reduce it to the narrow one – *once and for all.* I would like to emphasize this, as it is claimed over and over that Post provided in his 1936 paper "much the same analysis as Turing". As a matter of fact, Post hardly offers an analysis of effective calculations or combinatory processes in this paper; it may be that Post took the context of his own work, published only much later, too much for granted.

To state that Post's approach fundamentally differs from Turing's in that he did not reduce the broader notion to a narrow one, *once and for all*, is to our mind a misrepresentation of the facts. Besides the fact that Turing nowhere states that his "reduction" should have this "once and for all"-character, it should also be noted that, although Post very much emphasized the significance of considering wider and wider formulations, and of regarding the identification as a hypothesis or a natural law, this does not mean that he did not perform a similar kind of analysis as Turing, of "*all the possible ways in which the human mind [can] set up finite processes for generating sequences.*"
As we showed in Sec. 3.1.3, through several quotes, Post at least made a start with such an analysis in the twenties, and, it thus seems very reasonable that he indeed approached the problem in a similar way as Turing did. In fact, it was such analysis he considered basic in order to show the general validity of his original thesis, and understood the significance of his (theorems) only relative to the universality of such thesis. It would thus be rather surprising if Post did not regard his formulation 1 as a satisfactory result of such analysis.[23]
The fact that the purpose of his 1936 paper was exactly to offer not only a system of logical potency but also of psychological fidelity, only further strengthens this, especially in linking this with Post's remarks in his *Account of an anticipation* on the significance of providing a psychological analysis, rather than

[23] The reader is referred to the quotes from Sec. 3.1.3, ending the section on Post's first thesis.

a mathematical proof, to support the validity of his first thesis.[24] If Post did not intend his formulation 1 as the result of an analysis of the relevant intuitive notions, thus narrowing down the broader notion to a formalism, one should also ask: why else would he have submitted the paper anyway, after having read Church's? Besides, what other reasonable explanation can one give for the similarity between Post's and Turing's formalisms? We have assumed earlier that Post's 1936 paper resulted from such an analysis, and understood this as one of the reasons why Post's paper is very significant: it shows that if one starts from the analysis of the relevant intuitive notions to construct a formalism that should be able to capture the intuitive notion, one ends up with something like Turing machines or formulation 1.

It is true that Post *did not* provide such analysis in his paper, in the sense that he did not explicitly describe the deduction itself of several abstract conditions or properties of the mental processes involved in solving certain mathematical problems, but this does not imply that he did not start from such analysis.[25] In fact, we do not see any other explanation for Post having arrived at formulation 1.

Post did not *extend* a narrow notion *reducibly* in this paper, but rather stated the significance of making such extensions, *starting from* the restricted formalism he described in the paper. The basic difference between Turing and Post is that Turing talked about definitions, whereas Post emphasized the significance of not "hiding" the true philosophical character of the identification by calling it a definition, and thus the significance of making explicit its hypothetical or lawlike character.[26]

[24] Again, the reader is referred to the quotes mentioned above from Sec. 3.1.3.

[25] It is interesting to note that whereas Turing analyzed the process of a man calculating on a piece of paper, Post started from the mental processes underlying this activity. This difference might offer an explanation for the above mentioned difference between Post's and Turing's formulation.

[26] One could of course ask: But why did Post "make such a fuss" about these issues? Although we cannot provide real arguments here, we believe there is a close connection between Post's interpretation of these identifications as hypotheses or laws and his earlier work. As was shown in Sec. 2.2.5 Post only wanted to make a beginning with an analysis of all the possible finite processes the human mind can set up to generate a set, after his work on tag systems and the

To Post, the several identifications were more than a mere mathematical issue to be used to prove certain results: the results proven on the basis of the identifications are not only limitations for the formalisms considered, but, are in fact, limitations man himself cannot overcome. This was also stated explicitly in Post's introduction to the Appendix of his *Account of an anticipation* ([Pos65], p. 395):

> The unsolvability of the finiteness problem for all normal systems, and the essential incompleteness of all symbolic logics, are evidences of limitations in man's mathematical powers, creative though these be. They suggest that in the realms of proof, as in the realms of process, a problem may be posed whose difficulties we may never overcome; that is that we may be able to find a definite proposition which can never be proved or disproved.

From this perspective, calling such identifications natural laws, rather than definitions, is very reasonable, since they concern limitations of something physical or natural, i.e., the human mind. Of course, this only makes sense in as far as one accepts that man is indeed as limited as a Turing machine when trying to e.g. solve the halting problem for a given machine. But this is exactly where the identification's hypothetical character lies: not everybody seems to be willing to accept this limitation, *a limitation that should not be misinterpreted as the statement of a computationalist world view, stating that the human mind is an algorithm.* The limitation only concerns a specific class of problems connected to computing itself, and does not say a thing about e.g. creative processes or human consciousness.

insight that *Principia* is reducible to a normal system, i.e., it were his "analyses" of certain formalisms that led him to the statement of his first thesis and the reversal of his entire program. Thus, Post himself had gone through the process of considering wider and wider formulations of what was assumed to be capable to capture the whole of mathematics, i.e., *Principia*, and it were these generalizations that showed him what was really going on. We are aware that the possible connection between Post's earlier work and his emphasizing the inductive character of the identification is a mere speculation from our side, but we still wanted to add it here as a (speculative) footnote.

Post was not the only one who was involved in the developments of the thirties and pointed out the hypothetical character of the several identifications. It was Kleene – whose contributions to the development of not only λ-calculus but also the theory of recursive functions cannot be underestimated – who first called Church's identification a *Thesis* in a paper published in 1943. After the statement of Church's identification as Thesis 1, i.e.,"*Every effectively calculable function (effectively decidable predicate) is general recursive*", Kleene writes ([Kle43], p. 274):

> Since a precise mathematical definition of the term effectively calculable (effectively decidable) has been wanting, we can take this thesis, together with the principle already accepted to which it is converse, as a definition of it for the purpose of developing a mathematical theory about the term. To the extent that we have already an intuitive notion of effective calculability (effective decidability), the thesis has the character of an hypothesis – a point emphasized by Post and by Church. If we consider the thesis and its converse as definition, then the hypothesis is an hypothesis about the application of a mathematical theory developed from the definition. For the acceptance of the hypothesis, there are, as we have suggested, quite compelling grounds.

Kleene makes a differentiation between the thesis regarded as a definition or as a hypothesis, depending on the kind of context one is working in. If the purpose is to further develop a mathematical theory centered around effectiveness, the thesis can be included as a definition. However, looking at what the thesis itself actually is, i.e., a statement about the identification between an intuitive notion, *to the extent that we have already an intuitive notion of effective calculability*, and a given formalism, it has the character of an hypothesis. Thus, even if one includes the thesis as a definition in the body of a given mathematical theory, then the hypothetical character of the thesis, turns into an hypothesis of the applications of the mathematical theory developed from the inclusion of the thesis as a definition. In other words, although including the thesis as a definition into a theory is a very reasonable practice, the results from the theory themselves are rooted in the hypothetical character of the thesis. For example, the thesis as hypothesis, is a hypothesis concerning the statement of the absolute character of unsolvable decision problems.

In understanding the thesis as a hypothesis Kleene, surprisingly, not only refers to Post's [Pos36], but also to a paper by Church from 1938 on the constructive second number class [Chu38]. Now, we already know about the significance certain heuristic arguments have played in the developments leading to Church's 1936 paper [Chu36c], but, he clearly rejected the idea of calling the thesis a hypothesis, although, especially in Church's case, its original formulation goes back to certain heuristic arguments.

In the paper Kleene refers to, Church proposes a definition which makes it possible to distinguish between the constructive and the non-constructive ordinals of the second number class ([Chu38], p. 224):

> The existence of at least a vague distinction between what I shall call the constructive and the non-constructive ordinals of the second number class, that is, between the ordinals which can in some sense be effectively built up to step by step from below and those for which this cannot be done (although there may be existence proofs), is, I believe, somewhat generally recognized. My purpose here is to propose an exact definition of this distinction and of the related distinction between constructive and non-constructive functions of ordinals in the second number class [...]

We have gone through the paper by Church but we have not found any *explicit* mention by Church of calling the identification a hypothesis or a thesis. In discussing the several characterizations of effective calculability, i.e. λ-definability, general recursiveness and Turing machines, Church emphasizes the vagueness of the notion itself, but, as one should expect, considers the several characterizations as definitions ([Chu38], pp. 226–227):

> This notion of an effective process occurs frequently in connection with mathematical problems, where it is apparently felt to have a clear meaning, but this meaning is commonly taken for granted without explanation. For our present purpose it is desirable to give an explicit definition.

From this quote it is again clear that Church's reason for interpreting the several identifications as definitions, has to do with their usage in a mathematical theory. I.e., if one uses the notion to prove certain mathematical results, it should be taken as a definition, not as a hypothesis.

There are, however, two passages in the paper from which one could deduce that Church admits the hypothetical character of such definitions. In the beginning of the paper, Church makes statements about the absolute character of the definition he proposes in the paper, but admits that this is merely a belief from his side and describes ways to counter the belief for those who do not accept it ([Chu38], p. 224):

> Much of the interest of the proposed definition lies, of course, in its absoluteness, and would be lost if it could be shown that it was in any essential sense relative to a particular scheme of notation or a particular formal system of logic. It is my present belief that the definition is absolute in this way – towards those who do not find this convincing the definition may perhaps be allowed to stand as a challenge to find either a less inclusive definition which cannot be shown to exclude some ordinal which ought reasonably to be allowed as constructive, or a more inclusive definition which cannot be shown to include some ordinal of the second class which cannot be seen to be constructive.

Later on in the paper, Church proposes as a formal definition of the notion of a constructive function of ordinals in the second number class, the λ-definable functions of ordinals in the second number class and argues ([Chu38], p. 231):

> As a definition of the notion of a constructive function of ordinals in the second number class, it is proposed simply to identify this notion with that of a λ-definable function of ordinals in the second number class. This is rendered plausible by the known properties of the λ-formalism, and no definition with a more direct appeal suggests itself. It has been proved by Church and Kleene that a large class of functions of ordinals are λ-definable, including addition, multiplication, exponentiation, [...], the predecessor function of ordinals, and others.

Church assumes λ-definability to offer the best definition of the notion, and relies, not on the arguments that so much convinced him of Turing machines offering "the most convincing form" of the definition for effective calculability, as he states in the paper ([Chu38], p. 227), but on properties inherent to the λ-formalism itself, implicitly referring to the heuristic argument that originally

led him to the first (unofficial) statement of his thesis in terms of λ-definability. The fact that Kleene refers not only to Post but also to Church in the context of calling the thesis a hypothesis, together with the quotes (shortly) discussed here, and our conclusions from Sec. 2.3, suggest that Church's reaction on Post is not completely in line with his own beliefs. However, we cannot simply neglect the rather strong reaction by Church on Post's 1936 paper, so, for now, it is not completely clear to us what Church's exact opinion really was. We think that a study of the correspondence between Church and Post might throw some new light on this issue.
Returning to Kleene, contrary to his supervisor, he did explicitly recognize the hypothetical character of the thesis, although, if one is actually developing a mathematical theory, it is more reasonable to take it as a definition. In Sec. 2.3 we showed the significance Kleene attached to heuristic arguments, and it was Kleene, together with Church, who λ-defined as many functions over the integers he and Church could think of, in order to develop the theory of λ-definability as a theory of functions over the positive integers.
The "quite compelling grounds" Kleene refers to in the quote given above, indeed include heuristic arguments. The arguments are summarized, as he notes, in footnote 2 in Kleene's [Kle38]:

> This notion of effectiveness appears, on the following evidence, to be general. A variety of particular effective functions and classes of effective functions (selected with the intention of exhausting known types) have been found to be recursive. Two other notions with the same heuristic property have been proved equivalent to the present one, viz., Church-Kleene λ-definability and Turing computability. [...] Functions determined by algorithms and by the derivation in symbolic logics of equations giving their values (provided the individual steps have an effectiveness property which may be expressed in terms of recursiveness) are recursive.

In other words, the arguments supporting the thesis as hypothesis, considered by Kleene in this footnote, are the *argument by confluence*, the *argument by example* and the *step-by-step argument*, thus including two more heuristic arguments. Also Post refers to this footnote in a footnote to the following quote

([Pos44], p. 285):

> The importance of the technical concept recursive function derives from the overwhelming evidence that it is coextensive with the intuitive concept effectively calculable function.

In Kleene's *Introduction to metamathematics* [Kle52], he devotes two chapters to present the evidence for Church's thesis. Here Kleene emphasizes that one cannot prove the thesis, due to its reliance on a vague notion ([Kle52], p. 317):

> Since our original notion of effective calculability of a function (or of effective decidability of a predicate is a somewhat vague intuitive one, the thesis cannot be proved. The intuitive notion however is real, in that it vouchsafes as effectively calculable functions, and on the other hand enables us to recognize that our knowledge about many other functions is insufficient to place them in the category of effective calculable functions.

Kleene differentiates here between four different classes of arguments. It should be noted that he did not consider the argument by confluence as heuristic evidence, but as an argument in itself. These four classes are:

I Heuristic evidence

II Equivalence of diverse formalisms

III Turing's concept of a computing machine

IV Symbolic Logics and Symbolic Algorithms

For each of these classes, except for **III**, Kleene further differentiates between several further arguments. As for the heuristic support, Kleene not only mentions the argument by example, but adds two more. First of all, he mentions the fact that the methods for showing effectively calculable functions to be general recursive have been developed to a degree which virtually excludes doubt that one could describe an effective process for determining a function that cannot be transformed by one of these methods in a general recursive function. Secondly, and this is very interesting, Kleene adds the following argument ([Kle52], pp. 319–320):

> The exploration of various methods which might be expected to lead to a function outside the class of the general recursive functions has in every case shown either that the method does not actually lead outside or that the new function obtained cannot be considered as effectively defined, i.e. its definition provides no effective process of calculation. In particular, the latter is the case for the Cantor diagonal method.

As was the case for Post, Turing and Church also Kleene regarded the impossibility of doing the diagonalization effectively as an important argument supporting the thesis, an argument which he explicitly classified as heuristic in nature!

As far as **II** is concerned, Kleene mentions the significance not only of the equivalence between several different characterizations of effective calculability, which, as he mentions, have all the same heuristic property **I**, but also that several of these characterizations have a certain stability, i.e., there exist many variants of the same formalism that are all equivalent. Argument **III** concerns Turing's machines and, more specifically, the fact that it directly arose from an analysis of the intuitive notion, contrary to the other characterizations, and is thus an independent statement of a thesis equivalent to Church's ([Kle52], pp. 321–322):

> Turing's notion is the result of a direct attempt to formulate mathematically the notion of effective calculability, while the other notions arose differently and were afterwards identified with effective calculability. Turing's formulation hence constitutes an independent statement of Church's thesis (in equivalent terms). Post [Pos36] gave a similar formulation.

The class **IV** arguments concern a detailing out of Church's step-by-step argument ([Kle52], p. 323):

> In brief [**IV**] show[s] that if the individual operations or rules of a formal system or symbolic algorithm used to define a function are general recursive, then the whole is general recursive.

As is clear, as was the case for Post, Turing, and, actually, also for Church, Kleene attached great value to providing several different arguments supporting the thesis.

It has been (correctly) emphasized by Webb [Web80] that Kleene made major contributions to the domain of recursive function theory that are basic to add further support to Church's thesis. We will not discuss these contributions here in any detail. For more details the reader is referred to Webb's analysis [Web80], especially pp. 203–219, showing how several of Kleene's results give a kind of formal protection to Church's thesis. In this sense, Church's thesis is protected from the inside out, i.e., it is through results developed within the theory, that one finds new arguments supporting the thesis. The same goes for all arguments of type **I**.

Basic here is Kleene's 1938 paper [Kle38] in which he introduces the notion of partial recursive functions, functions that are not defined for all possible values ([Kle38], p. 151):

> If we omit the requirement that the computation process always terminate, we obtain a more general class of functions, each function of which is defined over a subset (possibly null or total) of the n-tuples of natural numbers, and possesses the property of effectiveness when defined. These functions we call partial recursive.

As is noted by Kleene [Kle87], introducing partial recursive functions has made it possible to separate the question of effectiveness from questions for given arguments whether the function being computed is defined.

In emphasizing the significance of partial recursive functions, Webb has introduced Kleene's thesis:

> **Kleene's Thesis.** *Every partial effectively computable function is partial recursive.*

On the basis of partial recursive functions Kleene was able to prove the important *recursion* or *fixed-point theorem* that gives in several different respects important support for Church's thesis. To be more specific, it can be used as an argument of class **I**, and has played a basic role in the proof of the equivalence between λ-definability and recursiveness [Kle36b], and is thus also important

for arguments of type **II**. [27]
A last argument supporting Church's thesis, or the other variants, that should be mentioned here is the protection offered by Gödel's incompleteness theorem and has been emphasized by Webb [Web80]. Indeed, it can be shown that:

> not-(Gödel's incompleteness theorem for F) → not-(Church's thesis)

where F is a system such as Gödel considered. As Kleene notes, if logicians would have been ignorant of Gödel's incompleteness theorem, one could have proposed that Church's thesis leads to Gödel's theorem. In fact, as is noted by Kleene, he used Church's thesis to give proofs of Gödel's incompleteness theorem [Kle36a], [Kle43]. What Kleene does not note is that Post in fact had already understood this relation in 1921, although it was not made explicit nor proven in any detail. In his [Pos44] this connection would be made more explicit by his so-called "Gödel in miniature" theorem, based on his proof of the existence of a recursively enumerable set that is not recursive.

3.2.4 Some further developments.

In this section we have discussed some of the possible interpretations of the status of the identifications proposed in the thirties. As is clear, there are several different possibilities. One can regard the identifications as definition, theorem, hypothesis, thesis or natural law. Important to note is that, given Kleene's and Post's interpretation of the identifications as hypo(theses) or laws, they do not seem to consider the kind of analysis as done by Turing (but also by Post) as the decisive argument. Instead, it is just one of the several arguments, that help to support the hypothesis. In this sense, although Turing's analysis gives basic

[27] In Kleene's paper the recursion theorem is called a circular definition, and is stated in terms of λ-calculus. The recursion theorem indeed includes a kind of circularity, i.e. it formalizes self-reference. As is noted by Webb, a special form of the theorem is that for any partial function ψ there is an index r such that $\phi_r(x) = \psi(r,x)$, where ϕ_r is the r-th partial recursive function in the enumerations (resulting from Kleene's enumeration theorem) of the partial recursive functions $\phi_i(x)$, with $\phi_i(x) = U(\mu y T(i,x,y)$. If we then define $\psi(r,x) = \phi_{f(r)}(y)$ we get $\phi_r(y) = \phi_{f(r)}(y)$.

support, its significance relative to the other arguments should not be overestimated. In fact, one might well wonder how the several identifications would have been perceived, if one would e.g. not have been able to show that Turing computability is equivalent to e.g. λ-definability, but would have merely been able to reduce it to λ-definability (but not vice versa).

Nowadays one normally understands or designates the identifications as theses. In fact, it seems to be common practice to talk about either Church's, Turing's or the Church-Turing thesis and one often neglects the existence of several other identifications that have been made in the meantime. Indeed, besides Post's theses, several other identifications have been proposed.

We already mentioned Kleene's thesis, but should also include Markov's thesis here. Markov developed his own kind of formalism known as normal algorithms. This was described in Markov's book [Mar54]. The contribution by Markov is only rarely mentioned, let alone studied. It would be particularly interesting to further study Markov's contribution in relation to Post's work, since Markov devotes several sections on Post's canonical forms. The reason for Markov to provide yet another (equivalent) identification is the fact that to his mind, the identifications proposed in the 1936 papers by Church, Post and Turing ([Mar54], pp. 2–3):

> [...] lead to a sharpening of the precision of the concept of algorithm *in an indirect matter* [m.i.] [...] In view of the foregoing, the author has considered it expedient to have the concept of algorithm rigorously established from the outset and to work out a general theory of algorithms on this rigorous basis. [...] The author considers, that he has succeeded in solving satisfactorily the problem formulated and that the theory of algorithms expounded here proceeds from a sufficiently simple and yet convenient definition of a "normal algorithm". In what measure this claim is justified, is left to the judgement of the reader.

Several other identifications were mentioned in a paper by Maslov which, besides Post's [Pos43], make precise the notion of a generated set [Mas67]. We cannot describe these results here and didn't have the chance yet to study these papers, but we think it important to at least mention their existence. These are Lorenzen's [Lor55], Curry's [Cur58], Smullyan's [Smu61] and Uspensky's [Usp53].

The several possible identifications that have been proposed in the meantime, and which are all equivalent, illustrate the rich variety of formalisms covered by for example Turing's thesis and makes the argument by confluence much stronger. Furthermore, these identifications illustrate that the dominance of Turing's thesis becomes even more relative in the light of the fact that other formalisms can be more suitable depending on the kind of intuitive notion one has in mind, even if in the end all the formalisms considered are equivalent.
Although, as we have shown, there are several arguments supporting the identifications, not everybody accepts them. In Sec. 4.3 we will discuss the (rather recent) attempts to get beyond the Turing limit. Before one wanted to go beyond the Turing limit, several other criticisms had already been formulated. We will not discuss these in any detail here, since these criticisms have been countered on many different occasion. Some critiques on the theses have been formulated in the following papers [Pét59], [Kal59], [Por60], [Pen89], [Pen94], [Luc61], where the last three are only indirect criticisms, in that they start from Gödel's incompleteness theorem. These critiques have been countered by several authors, see for example [Dav82], [Dav90], [Dav93], [Fef95], [Kle87], [Men63], [Web80].

Nowadays there are several developments that are closely connected to the *philosophical significance* of the identifications discussed here. We will only mention some of them here.
First of all, some researchers have developed new formalisms to deepen, explicate or make more general (certain aspects of) especially Turing's thesis, possibly to tackle certain objections. For example Gandy [Gan80] has proposed a class of machines, now known as Gandy machines. These were invented by Gandy because Turing's analysis of computability cannot be applied *directly* to discrete physical mechanical devices. Gandy's main purpose was to analyze the notion of mechanical process in order to add strength to what he called Thesis M, i.e., what can be calculated by a machine, i.e., a discrete mechanical device, can be computed by a Turing machine. This paper thus contains one of the first statements of a more physical version of the Church-Turing thesis. More will be said about this in Sec. 4.3.
Sieg and Byrnes developed *K-graph machines* to give a detailed mathematical

explication of what Sieg identified as Turing's theorem together with the conditions **L** and **B**. These K-graph machines are considered more "general" than Turing machines, i.e., they are based on planar computations. K-graph machines are then considered as providing "*a significant strengthening of Turing's arguments for his central thesis.*" ([SB96], p. 98).

In an, at this time, unpublished paper [Sie07], available on-line, Sieg formulated axioms for discrete dynamical systems which "*should be viewed as determining classes of "algebraic structures" of which particular models of computation are instantiations.*" ([Sie07], p. 12), starting from the several conditions he deduced from Turing's analysis. The purpose of these axioms is to gain "eine Tieferlegung der Fundamente" (a deepening of the foundations) and are suggested to be an answer to what Gödel considered to be the best approach to find a good identification, as he expressed in a conversation with Church already mentioned, i.e., to state a set of axioms embodying the generally accepted properties of effective calculability.

A second development that is often connected with the Church-Turing thesis are computational models of certain (aspects of) physical or biological processes, like e.g. cellular automata – first conceived by Von Neumann in developing an abstract model for self-reproduction of biological systems [vN66] – and neural networks. The idea of neural nets has many historical roots, but one of the most important historical papers here are the two papers by McCullough and Pitts [MP43, MP47]. They showed the equivalence between a finite network of formalized neurons and offered models for designing nervous nets that recognize visual input. Nowadays there are hundreds of researchers who work in this domain. Some of the more philosophical research in this context has been connected to especially Turing's thesis (if the theoretical models considered are simulated on a computer). The fact that so many biological or physical processes can be simulated by a computer, is sometimes interpreted as an indication that the Turing limit as a universal limit, i.e., nothing that is in our world could not be computed by a universal Turing machine (see for example Wolfram's [Wol02]). However, it should be emphasized again here that Turing's thesis does not necessarily imply this point of view.

A last development that should be mentioned here, and stands in sharp contrast with the previous, is the domain of hypercomputability. This will be discussed in more detail in Sec. 4.3. Within this context, several researchers have proposed models they consider to be able to "compute" the non Turing computable.

The developments (cruelly) summarized here are all concerned with the general-theoretical statement of the Church-Turing thesis, trying to generalize it or show its limitations. In other words, if one discusses Turing's thesis in this more philosophical setting, it is in most of the cases about the *general class* of Turing machines, or the general idea behind it, not about specific cases of machines. Indeed the question posed in the introduction, i.e., the connection between, on the one hand, the general statement of the thesis and the unsolvability results that can be proven by using it, and, on the other hand, particular machines and their decision problems, is only seldom taken into consideration in this context.

A further observation to be made here is that one only rarely takes into consideration anything else but Turing's thesis if a more historical reference is made, especially when the more philosophical issues are at stake. In fact, I do not know of any relatively recent (philosophical) paper that discusses the Church-Turing thesis in terms of say normal systems or λ-calculus, and take into account formalisms that have a less direct appeal to intuition.

In this dissertation we would like to start from exactly the opposite direction, and study formal systems that are far removed from our intuition of computability.

3.3 Strategies against intuition.

> The point about incompleteness is not that formal systems 'are missing some of our intuitions', but rather that the processes they *are* capable of expressing may be effectively undecidable, according to [the Church-Turing thesis], *no matter how many of our intuitions they might formalize.*

Judson W. Webb, 1980.[28]

> [...] if we are to really understand the evidence for (CT), and hence (CT) itself, we must examine equivalence proofs between concepts falling under different conceptual groups, at least to the point of isolating key-ideas, for they have too,often been taken for granted, with the result that discussions of (CT) tend to end just where they should really begin.

Judson W. Webb, 1980.[29]

Already from the first statements onwards of a thesis such as Turing's, several interpretations have been proposed with respect to, on the one hand, the status of the theses, and, on the other hand, the kind of arguments that are most important in discussing the several theses. Despite the overwhelming evidence for the validity of these theses, people are still attracted to the subject nowadays.[30]
As we showed in the previous section, both Gödel and Church agreed that Turing's thesis should be regarded as the most adequate identification given the directness of the identification, through Turing's analysis. In the meantime, there seems to be some general consensus, that Turing's thesis is the most convincing, witness its dominance in the philosophical literature. Indeed, even if one usually talks about the Church-Turing thesis, it is most often Turing's analysis and especially the resulting general Turing machine concept, that forms the main focus of the contemporary discussions surrounding the subject (at least as far as we can see). In this section, we would like to argue, on the basis of the results of this and the previous chapter that, although we understand Turing's "direct appeal to intuition" argument as an important one, it is *philosophically important* to also consider those identifications that are not directly covered by this argument.

[28] From [Web80], p. 198.

[29] From [Web80], pp. 211–212.

[30] Part of the ideas from this section have been presented at a talk I gave at a conference in Laval, France, *International conference on Computers and Philosophy (I-C&P)*, 3–5 May, 2006 [Mol06b].

As was shown, Church, Post and Turing all had their own way of arriving at their respective theses, and each provided their own arguments and interpretations. Clearly there are different types of arguments supporting the theses which were subdivided into four main classes by Kleene [Kle52]. Neither Church, Post nor Turing explicitly mentioned all of these arguments, partly due to the specific formalisms they used.

Although the argument by example could be said to have functioned as the method that led Church to the first statement of his thesis, he did not use it in his 1936 paper. He did mention the argument by confluence in a footnote, while the step-by-step argument functioned as the main argument in this paper. However, once he had read Turing's paper, it seems that the "appeal to intuition" argument was the most convincing for Church. Indeed, as is clear from his review [Chu37b] on Turing's *On computable numbers*, he considered Turing's identification as the most natural and convenient one as compared to the identification of effectiveness with λ-calculus or recursiveness.

Turing has made it very explicit what kind of arguments he took into account: the argument by example, the argument by confluence and, the "direct appeal to intuition" argument. The fact that the diagonalization cannot be done effective was also noted by Turing, but he did not mention it explicitly as an argument. He also pointed out that there is an important logical connection between Gödel's incompleteness result and the unsolvability of the Entscheidungsproblem, but, again, he does not really use it as an argument.[31]

In Post's case, it is clear that the argument by confluence played a very important role in his work. The normal form theorem, and the fact that he considered *Principia* to be reducible to a normal system, have been basic for him to realize the generality of his systems in normal form. Furthermore, considering wider and wider formulations which can be shown to be reducible to formulation 1, was understood as a way to turn the hypothesis into a law. As far the "direct

[31] "*If the negation of what Gödel has shown had been proved, i.e. if, for each $\mathfrak{U}$, either $\mathfrak{U}$ or $-\mathfrak{U}$ is provable, then we should have an immediate solution of the Entscheidungsproblem. For we can invent a machine $\mathcal{K}$ which will prove consecutively all provable formulae. Sooner or later $\mathcal{K}$ will reach either $\mathfrak{U}$ or $-\mathfrak{U}$. If it reaches $\mathfrak{U}$, then we know that $\mathfrak{U}$ is provable. If it reaches $-\mathfrak{U}$, then, since **K** is consistent [...], we know that $\mathfrak{U}$ is not provable.*"

appeal to intuition" argument, it is clear that it must have played an important role, since he most probably performed a similar kind of analysis as Turing, an analysis he considered basic for making his thesis more acceptable. He also indirectly referred to the argument by example, by mentioning Kleene's footnote from [Kle38] and was clearly aware of the fact that the diagonalization cannot be done effectively, since he mentioned it in his [Pos65].

To our mind, each of the different arguments has its own important value for supporting the several theses, i.e., taking them all together one has a very strong case for accepting the theses. Except for the "direct appeal to intuition" argument, all of these arguments can be directly applied to the different formalisms considered by Church, Post and Turing. If we do not take into account this one argument, each of the theses can be considered as being equally adequate for the intended identifications. However, this *if* is exactly the reason for many researchers to consider Turing's thesis as the most natural and adequate identification. This was clearly the case for Church and Gödel. Also Kleene has pointed out that he considers Turing computability as "*intrinsically persuasive*", while "*λ-definability is not intrinsically persuasive*" and "*general recursiveness scarcely so (its author Gödel being at the time not at all persuaded.)*" ([Kle81b], p. 49). As far as Post is concerned, I do not know of any explicit statement from his side where he considers his formulation 1 or Turing machines as better or more adequate identifications as compared to his first thesis or Church's thesis. Of course, he did regard the analysis leading to his formulation 1 as very important, but this does not necessary imply that one should consider one identification as being superior to another one. In the end, given the equivalence between the different formalisms they are all equally adequate, at least, from a theoretical point of view.

So why is it exactly that the "direct appeal to intuition" marks the difference, according to some researchers, between the several theses? The most important reason is that Turing computability is indeed intrinsically persuasive, resulting from a direct analysis of the intuitive notion involved. I.e., Turing machines or other similar machine-like formalisms like formulation 1, naturally follow from such an analysis, by taking into account the general properties of a man in the process of computing. That this is indeed the case, is illustrated by the fact that both Post and Turing formulated similar formalisms by starting from such an analysis. As a consequence of its directness, Turing machines, even if one does not go through Turing's analysis, are indeed far more intuitively connected to our notion of computability than e.g. normal systems. As we already argued, the mere use of the notion of a machine, instead of a form or a calculus, that moves and prints over a tape, recognizing symbols, and capable of changing its "state of mind", has indeed a very direct appeal to intuition, also, if not especially, for those who are not in the domain of mathematical logic or computer science, like e.g. philosophers. Especially now, in the computer era, this kind of identification has become even more obvious. Furthermore, Turing machines are far more easy to program than e.g. λ-calculus or systems in normal form, although it is not completely clear in how far this property is a consequence of Turing's way of having arrived at his machines.

While we certainly do not want to oppose the significance of Post's or Turing's proposal – it is a very important aspect of the several theses – some comments and questions are in place here.

3.3.1 The thesis as a definition. On the significance of using the right formalism for the development of the theory.

In the previous section we showed that there have been several different understandings of the status of the theses. To our mind Kleene's position is the most reasonable: he first termed the identification put forward by Church as a thesis, and emphasized that, depending on the context it is used in, it can be understood as a definition or as a hypothesis.

Taken as a definition, in order to develop a mathematical theory that is rooted in the thesis, it is clear that it is not a very good idea to only consider those formalisms that have a direct appeal to intuition, to develop the theory. As Turing remarked ([Tur37], p. 153):

> The identification of effective calculable functions with computable functions is possibly more convincing than an identification with the λ-definable or gen-

> eral recursive functions. For those who take this view the formal proof of equivalence provides a justification for Church's calculus, and allows the 'machines' which generate computable functions to be replaced by the more convenient λ-definitions.

As is clear from this quote, Turing considered the λ-calculus as more convenient and he would use this calculus, and not Turing machines, exactly because of this reason in his seminal Ph.D. dissertation on ordinal logics [Tur39]. It is interesting to note that Turing uses the equivalence between λ-calculus and Turing machines, as a sufficient reason to accept Church's thesis, for those who find that Turing machines are more convincing with respect to the formalization of computability. This is also our point of view. While we consider Turing's (and Post's) formalisms as basic support for the theses, the fact that these formalisms can be shown to be equivalent to e.g. λ-calculus results in theses that are equally strong, even if e.g. λ-calculus is not "intrinsically persuasive". Also Post understood that the existence of several different formalizations offers a theoretical advantage ([Pos47], p. 7):

> The writer has often felt that the multiplicity of equivalent formulations of recursiveness has been a deterrent to the general promulgation of this discipline. Yet, the writer's normal systems naturally lead to the unsolvable problem of [Pos46], while the deterministic character of the Turing machine is basic to the above unsolvability proof. From this point of view, the several formulations of recursiveness are so many different instruments for tackling new unsolvability proofs.

Church also seems to have shared this point of view. Indeed, in our analysis of Church's earlier work, we discussed a quote in which Church emphasizes that one cannot simply say that one system of logic is wrong and the other right, but only that one is more convenient than the other, depending on the goal one has in mind.[32] It is indeed common practice to simply use the formalization that is most convenient, as a definition, for the particular kind of goal one has in mind. It is also in this context that turning one's attention from the general identification to particular systems that are further removed from our intuitive notion, in

[32] See Sec. 2.3.3, p. 76.

that it is not quite clear what kind of function they are actually computing, can be very important both theoretically as well as philosophically. If one no longer cares about the exact interpretation, in an intuitive sense, of what a given system is exactly computing, one can more easily focus on other features of that system. In chapter 9 we will argue that a study of computational systems, on the basis of their behaviour rather than on the basis of the encoding of functions in the description of the machine, leads to important results in the context of studying limits of solvability and unsolvability.

3.3.2 Different intuitions, different formalisms

It is, to our mind, very important that besides the intuitive notion of computability, other notions have been considered. As we have shown, Post took into account two other notions, i.e., generated set and solvability. In the previous section, we also mentioned Markov's normal algorithms, which he considered to be more suitable to directly capture the notion of an algorithm as compared to the other formalisms by Church, Post and Turing.

These intuitive notions should cover the same kind of "intuitive meaning" as computability, given the equivalence between the several different formalisms. While solvability and the notion of a algorithm are still rather close to computability, it is not completely trivial to understand on an intuitive level that something that is computed is the same as something that is generated, *if one is not acquainted with the formalisms themselves.* Of course it is hard to provide any real arguments here, since everybody has his own intuitive understanding of several notions, depending on one's background. Still it is important to mention that something like normal systems seems to be better suited to work out a theory of generated sets, and the related notion of an effectively enumerable set, than say Turing machines.[33] But this is of course a personal opinion. More significant is the fact that in understanding that "generated" can be identified with "computed", or with "solved", the several proposed theses become stronger, since they cover not one but several intuitive notions. In a way, this is a kind of argument by confluence on the level of intuitive notions, rather than

[33] In this respect, see Post's seminal paper on recursively enumerable sets [Pos44].

on the level of the formalisms themselves. It leads to a broader intuition of what is meant with computability. Restricting one's attention to computability, blocks this kind of generalization of the intuition.

3.3.3 "Testing in practice": Studying formalisms instead of intuitions

In the previous chapter we have argued that the way Church and Post arrived at the original statement of their respective thesis, was not through direct analysis of a vague notion. On the contrary, the formalisms were already there and developed for quite different purposes. It was only by studying these formalisms that they became more and more convinced about the generality of their formalisms. In Church's case, it was the argument by example that played a fundamental role in this process. As for Post, there were two different aspects that should be mentioned here. First of all, one should not forget that Post was already searching for more general and abstract forms of mathematics, one of his major goals being to find a positive solution to the Entscheidungsproblem. It were his tag systems that first led him to the conclusion that finding such a solution might be impossible and led to the definition of his normal systems. Once he had established his normal form and proven the normal form theorem he realized how general these forms actually are, and proposed his thesis.

To our mind, it is rather remarkable that two of the three "pioneers of unsolvability" formulated their thesis not on the basis of an analysis of an intuitive notion, but on the basis of a study of certain formal systems. This shows how important other considerations have been and are for the formulation of such theses. But are there any other reasons, besides the historical one, to emphasize this?

Two questions are in place here. First of all, even if one regards Turing's thesis nowadays as more adequate, one can well wonder how his paper would have been received if he would have only proposed his thesis on the basis of his analysis, without the extra arguments (by confluence and example) and the fundamental results he has proven? Secondly, and this question is more fundamental, if one accepts Turing's thesis solely on the basis of his analysis, without having convinced oneself about the actual computational power of Turing machines by having studied and compared them with other formalisms, does one not risk to completely obscure the connection between a kind of very general, basically philosophical, identification of computability with the *general concept of* a Turing machine and the *actual use and implementation of particular Turing machines* and, as a consequence, to undermine the true value of the thesis itself? To put these two questions a bit differently: is it not because one only takes into account the *general* identification between computability and Turing machines based on Turing's analysis, without taking into account the other intuitive notions and formalisms that are covered by it as well as the actual formalisms themselves, that it seems to be so hard for some to accept the theses? As far as my own experiences go, it is only by having really worked with several different formal systems, and especially tag systems, that I not only began to understand more and more how significant and fundamental the respective theses are, but it were these experiences, rather than Turing's analysis, that really convinced me of the generality of the theses. It was me having worked on tag systems, for which it is far from clear what they are actually computing, that furthermore showed me a fundamental philosophical implication of the theses, i.e., that the limit implied by the several theses is not only a limit for the several formalisms, but also for me. I dare the person who says that he or she is convinced that it is possible to "*effectively solve*", understood intuitively, the halting problem for a given tag systems, to solve it for the very formally simple tag system, $\mu = 2, v = 3, 0 \rightarrow 00, 1 \rightarrow 1101$!

3.3.4 Can we trust our intuition?

From a pure philosophical point of view, one of the problems involved in considering the "direct appeal to intuition" argument as the most important one supporting Turing's thesis, is that intuitions or one's understanding of a vague notion can be plainly wrong and are always very much biased by one's own background. History has already proven that one should be very careful in accepting something on the basis of an intuition, the classic example in this context being Euclid's parallel postulate: for centuries people had searched

for a proof, until Bolyai and Lobachevsky considered the possibility of non-Euclidean geometries. Sieg, Gandy and some others have developed new formalisms, *on the basis of Turing machines*, to come to an even better, general or more fundamental formalization of certain intuitive notions. Although we consider their work very valuable, we think it equally important to challenge the intuition and to study formalisms that are further removed from the intuition. In having confronted myself with the rich variety of *different kind of processes* that can be used to compute, especially those that are *not* intuitively appealing, the notion of computability itself has been given a much more general meaning. Indeed, in looking at formalisms that are further removed from the intuition, one can only learn how much an intuition is in fact biased and that it can be very restrictive to only focus on those formalisms that capture the intuition, although it is of course basic to develop a formalism on the basis of certain vague notions, as Turing did. If one no longer challenges the intuition itself, one risks to forget about the rich variety of different processes that are actually covered by "computability".
Post proposed to consider wider and wider formulations to turn the hypothesis into a law. Sieg has criticized exactly this aspect of Post's work. To Sieg, Turing reduced a broad notion to a narrow one, while Post wanted to extend a narrow notion reducibly. To our mind, *both directions*, narrowing down and broadening up, are equally important. The former allows one to formalize the existing intuition in a direct fashion, the latter allows to broaden up that self-same intuition: to compute in λ-calculus or with tag systems is definitely a challenge for that intuition. In that way, both sides of the thesis, intuition and formalism can be generalized, thus leading to a much stronger case for any of the theses. This is exactly the significance of the argument by confluence: it allows one to see how general computability actually is.

In this and the previous chapter we have given a detailed historical picture of how Church, Post and Turing each arrived at their theses, the kind of arguments they considered important, as well as the kind of different interpretations that have been attached to the theses by some of the leading logicians at that time. Emphasis was put on the fact that there are some very clear differences present

in this history of the theses, differences we consider as fundamental both from a historical, a mathematical, as well as from a philosophical point of view.
However, the several formalisms considered, are not in any way "physical", i.e., they are idealizations and abstractions used to prove certain theoretical results, like e.g. the unsolvability of the halting problem. In this sense, the several theses concern abstract devices, not physical devices, that capture the vague notions or intuitions. Of course, and this is important to note in the light of our discussion on hypercomputability to follow (See Sec. 4.3), if either Church, Post or Turing would have believed that there are physical devices, which in their turn can be formalized, that solve e.g. the halting problem for Turing machines, they would probably not have published their results.
In the next chapter we will take a closer look at the development of what may be regarded as the physical realization of these formalisms, i.e., the computer, and the role it has played and plays in the context of computability and unsolvability.

Chapter 4

The computer.

> Mathematics deals with theorems, infinite processes, and static relationships, while computer science emphasizes algorithms, finitary constructions, and dynamic relationships. If accepted, the frequently quoted mathematical aphorism, 'the system is finite, therefore trivial,' dismisses much of computer science.
>
> Michael S. Mahoney, 2000.[1]

Four years after Church, Post and Turing published their papers, the world was at war. Turing would play a rather significant role in this war, since he worked at Bletchley park and helped to decode the Enigma.[2] He would also become one of the first computer pioneers, having designed a machine which can be regarded as the materialization of his own Turing machines.

Nowadays we can hardly live without the computer: if one would make a list of all the functions the computer fulfills in our society it would almost be scary. The idea of computing machines is very old, and goes back at least to the 17th century when several calculating devices were developed, like Pascal's calculator. In the 19th century, Charles Babbage conceived of the differential and analytical engine, which are very close to the ideas behind modern computers.

[1] From [Mah00], p. 17

[2] A detailed account of Turing's involvement in the war can be found in [Hod83]. In the meantime many more top secret documents have been declassified. The recent book [Gan06] gives a detailed account of the work at Bletchley park, taking into account these new documents, and focuses on the construction of the Colossus.

Especially the description of the analytical engine, that was never built, contains many typical features of current computers.[3]

To write a history of the development of the idea of computing machines through the ages, however, is not the purpose of this chapter. Our main interest here is with the earliest *general-purpose discrete electronic* computers that were built shortly after the second world war. One thing I still find very fascinating is that such computers – and the several generations of computers that have since been built – can be regarded as a kind of physical realization of the rather abstract ideas developed in the twenties and thirties. This is not only true for the more obvious similarity between "real" computers and a universal Turing machine. Many of the abstract methods behind the results from this period can be directly linked with certain aspects of the computer. It is thus not surprising that these mathematical ideas have had (and still have) an important influence on what is now known as the domain of computer science.

Placed in this context, the computer might be regarded as the physical answer to the question of what exactly is meant with "effective calculability". Since the early rise of modern computers, not only their size has been reduced to an almost absolute minimum,[4] but their field of application has grown in an almost unforeseen way. The computer is no longer restricted to "pure" calculation, but is involved in almost every aspect of our society. In this respect, the computer can be said to have changed our intuition of effective calculability. Calculability itself is no longer restricted to the domain of mathematical logic. It is remarkable that the formalization of computability has played a significant role in its own materialization and extension to other domains, going from biology to the development of interactive computer games.

It didn't take very long before people began to see the possibilities of computers. Several of the pioneers, like Turing and von Neumann, almost immediately began to ask questions concerning the link between, on the one hand, "nat-

[3] A still very interesting text to read is Ada Lovelace's translation of a French text describing the Analytical Engine, and the notes she attached to the translation [oLAA43].

[4] Although the computer itself might be further reduced in size, the user needs a screen that is big enough to keep the output discernible, except of course if one would start to develop computers with an "audible" interface.

ural" automata, like the brain, and, on the other hand, "artificial" automata. The idea of building "intelligent machinery" was put forward when the first computers were still in full development.[5] Although these ideas were regarded rather "heretical" at that time,[6] nowadays hundreds of people are involved in the domain of Artificial Intelligence. Still, there is a clear taboo surrounding the subject. Even I have difficulties to start a conversation with chatbots on the net! This taboo seems to be closely related to the fact that several researchers try to prove that the Turing limit is not a "real" limit for nature. But before we can discuss these matters in a bit more detail (Sec. 4.3), it is important to take a closer look at the early beginnings of the computer. It is, however, impossible to give a detailed account of the origin of the first computers. This would at least ask for another dissertation. Several books and papers have been published on the subject, focussing on different aspects of this history. The interested reader is referred to these sources.[7]

[5] See for example von Neumann's [vN58] and Turing's [Tur50].

[6] Turing wrote a paper with the explicit title: *Intelligent Machinery. A heretical theory* [Tur51a].

[7] In this footnote we would like to mention some of the references in this context, but we should warn the reader that these references are far from complete. First of all, we should mention the very important book *A history of computing in the Twentieth Century.* [HMR80], containing many papers that were presented at a conference in 1976 on the history of computers held at the equally historical location, Los Alamos. This is, to our mind, still one of the most important sources on early computer history, since it contains detailed accounts of the people who were actually there when it all happened. As for the history of computer languages, we should mention two papers by Knuth, one in programming languages [KP80] and one on compilers[Knu62]. The conference proceedings *History of Programming Languages* [Wex81] gives a very interesting account of early programming languages, since the authors were all involved in the development of the first computer languages. Goldstine's book [Gol72] is also known as a classic, but its main focus is the ENIAC and is biased with respect to the significance of von Neumann's contributions in this context. If one reads Goldstine's book, when should also read the accounts by Mauchly [Mau80] and Eckert [Eck80] in [Gol72], or, Scott McCartney's book [McC99]. Some books and papers have been written that situate the first computers in the context of the history of logic and mathematics. Martin Davis wrote a very accessible book describing this history [Dav01b], extending the ideas described in his [Dav87]. There is also a German book discussing similar matters by Sybille Krämer [Krä88]. In general, there is the important journal *IEEE annals for the history of computing*, that contains many contributions by

The main purpose of this chapter is to show how through he rise of the computer, on the one hand, computability has obtained a physical form, and, on the other hand, how this physical form has given rise to new possibilities and problems in the context of computability and unsolvability. In a first section 4.1 we will describe some aspects of the history of early computers and the need for developing programming languages. Focus will be put on the question of how much the developments described in the previous chapters have had their influence on the rise of the computer.

In the next section 4.2, we will show how, from its early use on, the computer was used to make available "the discourse" of mathematics. We will argue that the role of the computer as a physical realization of the formalisms we have described earlier, used to study the actual execution of these formalizations, can hardly be underestimated.[8]

The last section, discusses two theoretical developments that are situated in the context of computability and unsolvability, and are very closely connected to the rise of the computer, developments which stand in sharp contrast with each other, i.e., *computational complexity theory*, where one studies the feasibility of computations, and *hypercomputability* in the context of which one asks questions concerning the physical feasibility of devices that can give noncomputable answers.

4.1 The first computers: From rewiring to the need for programming languages.

> COMPUTER: any device capable of accepting information, applying prescribed processes to the information, and supplying the results of these processes; sometimes, more specifically, a device for performing sequences of arithmetic and logical operations; sometimes, still more specifically, a stored-program digital

people who were involved with the developments described.

[8] We are very much indebted to Maarten Bullynck for the very frequent and helpful conversations we have had when I was writing this chapter. The reader is referred to Bullynck's [Bul07], for an interesting discussion on the first computations done on the ENIAC.

> computer capable of performing sequences of internally-stored instructions, as opposed to calculators on which the sequence is impressed manually desk calculator) or from tape or cards (card programmed calculator).

Martin H. Weik, 1961.[9]

In [Ula80] Stanislaw Ulam, who worked together with von Neumann at Los Alamos, beautifully summarized the two main "streams" that gave rise to the first computers ([Ula80], pp. 94–95):

> It is perhaps a matter of chance, that computer development became possible only by a confluence of at least two entirely different streams. One is the purely theoretical study of formal systems. The study of how to formalize a description of natural phenomena or even of mathematical facts. Professor May has spoken felicitously of "genetic development": we call it axioms and rules of procedure. The whole idea of proceeding by a given set of rules from a given set of axioms was studied successfully in this connection. The second stream is the technological development in electronics, which came at just the right time. Of course, the war had greatly accelerated the availability of funds and effort just a few years later.

The confluence of formal systems and technology in the computer is still a very remarkable fact of history, especially in the light of the fact that the computer itself is nowadays used to study the formal systems it is the physical realization of.

The significance of the war in early computer development can hardly be overestimated: there was a very direct need for computing machinery for several different goals and several governments thus invested in this research domain. One could say that the war accelerated the development of the computers. However, many of the machines developed during the war, were special-purpose machines and not all were discrete.[10]

[9] From [Wei61]

[10] For example, the differential analyzer developed in the 30's by Vannevar Bush at MIT was analogue.

It is the idea of discrete general-purpose machines that is basic to the development of the computer as we know it today. A very important paper in this context is Shannon's master thesis *A Symbolic Analysis of Relay and Switching Circuits* [Sha38], that lay the foundations of digital circuit design. He showed how Boolean algebra can be used to efficiently analyze and synthesize relay circuits. The now almost trivial idea of AND, OR and other ports basically goes back to Shannon's thesis.This thesis is thus a very clear example of how logic and technology can go hand in hand.[11]

During the war there were several people at different places involved in the project of building computing machines in the U.S. It was here that Mauchly and Eckert signed a contract to build what is probably known as one of the most famous first computers, the ENIAC.

4.1.1 The ENIAC and the EDVAC

Neither Mauchly nor Eckert were mathematicians. Mauchly was a physicist, Eckert was an engineer. In 1941 Mauchly took a course in wartime electronics at the Moore School of Electrical Engineering, and this is were the two met. In 1942 Mauchly wrote a memo proposing to build an electronic computer. Lieutenant Herman Goldstine heard about the memo and asked Mauchly to write a more formal proposal. On 1 June, 1943 the contract was signed, and Mauchly and Eckert could begin with constructing the ENIAC. As Eckert emphasizes in [Eck80], the ENIAC was, from the very beginning, conceived as a general-purpose machine ([Eck80], p. 526):

> Long before we met von Neumann on the ENIAC project, it was John and I who had to figure out how to arrange for flexible controls that would do all the things that we felt were absolutely necessary for the generality of use that was our goal. We are glad that Leland Cunningham, then working for the Ballistic Research Laboratory (BRL), also had this philosophy and purpose. Unfortunately, it is often said that the ENIAC was built just for preparing firing tables. Cunningham and others at BRL all supported us in making the ENIAC as generally useful as

[11] A more detailed discussion is given in [Dav88].

> we could contrive to make it within the limited time that conditions of war demanded. Yes, BRL wanted firing tables, but they also wanted to be able to do "interior" ballistics, and all kinds of data reduction, and they went on and on with examples of what they would hope to be able to do with a truly flexible computer. We wince a little when we hear the ENIAC referred to as a special-purpose computer; it was not. The name "ENIAC," where the "I" stands for "integrator," was devised to help sell the Pentagon that what the BRL was getting would compute firing tables, which were, in 1943, the greatest need of Ordnance. But there was a flexibility of control far beyond the implications of the name.

Indeed, although computing fire tables was a very important military task of the ENIAC, it was not intended to be its sole purpose.[12] Does this mean that ENIAC was a real general-purpose computer as we know it today? No, especially not if one understands "general purpose" to be a kind of finite approximation of a universal Turing machine, including the stored program idea. One of the problems with ENIAC was that it was hard-wired: the sequence of instructions the machine had to follow had to be physically programmed. It was only after it had been rewired that it was able to "simulate" *stored program computers.*
And then John, Johnny for friends, von Neumann got involved in the ENIAC project. According to Eckert, the first visit of von Neumann to the ENIAC project could not have been before 7 September, 1944. It was Goldstine who introduced von Neumann to Mauchly and Eckert ([Eck80], p. 532).

> Apparently, while we were racing ahead on plans involving obvious uses of the delay storage devices, Goldstine had spent a great deal of time in the hospital with hepatitis, and had failed to get the full impact of the delay storage on con-

[12] The following quote, explains what fire tables were used for: "*The army used its lush fields and rolling hills to test artillery guns and other weapons. Since a gunner often couldn't see his target over a hill, he relied on a booklet of firing tables to aim the artillery gun. How far the shell travelled depended on a host of variables, from the wind speed and direction to the humidity and temperature and elevation above sea level. Even the temperature of the gunpowder mattered. A gun such as the 155-millimeter "Long Tom" required a firing table with five hundred different sets of conditions. Each new gun, and each new shell, had to have new firing tables , and the calculations were done at Aberdeen based on test-firings and mathematical formulas.*" [McC99], p. 53.

> trol problems. This makes more understandable his apparent belief that von Neumann was the source of ideas that in fact we had generated before Goldstine had met von Neumann at the Aberdeen railroad station. That chance meeting in Aberdeen was the very beginning of von Neumann's high interest in electronic computation. The clearance document for von Neumann's first visit to the Moore school ENIAC project has been found, and his first visit could not have been before 7 September 1944. In my own records, which also became a court document, is confirmation that Eckert and I had a commitment to meet von Neumann about 7 September. I believe that was our first meeting with him.

As is clear from this quote, there have been problems between, on the one hand, Eckert and Mauchly, and, on the other hand, Goldstine and von Neumann. Reading any paper by any of these people that is related to these matters, should thus always be done with the awareness that there has been a serious fight between them!

It is often stated that one of the main contributions by von Neumann to the first computers is the idea of stored programs, i.e., to store the instructions internally with the data. This was considered as one of the basic advantages of the next computer to be build at the Moore school, i.e., the EDVAC. Eckert and Mauchly however, have claimed that they already had the idea of stored programs even before they met "Johnny" for the first time. The most important cause for the problems between the "engineers" and the "logician" was the fact that von Neumann did not give full credit to Eckert and Mauchly when writing his *First Draft of a Report on the EDVAC* [vN45]. The EDVAC is up to today considered as one of the first all-purpose stored-program computers, and in this respect one of the first forerunners of our present-day computers (cfr. the so-called Von Neumann-architecture). Its completion was delayed due to the dispute between the "engineers" and the "logician", leading to Eckert and Mauchly to leave the University of Pennsylvania to form the Eckert-Mauchly Computer Corporation.

We will not discuss here the controversy about who had which ideas first. Until now it is still not completely clear how much Mauchly and Eckert contributed to the design of the EDVAC. It should be noted though that Eckert had already written an earlier report in January 1944, before the first meeting with von Neumann, with a proposal of developing a new computer, that included some ideas pointing in the direction of stored program.[13] In fact, one of the reasons for Eckert and Mauchly to develop a new computer, as they recount, was to avoid the long set-up time that could be avoided by stored programs, allowing for automated programming. We think that the most correct interpretation of what did actually happen is that the EDVAC design cannot be assigned to one person, but emerged from the several discussions between Mauchly, Eckert, von Neumann, Burks and some others, before the actual report was written. As far as the idea of stored programs is concerned, we believe that the "engineers" must have very quickly realized that internalizing the instructions would make it possible to significantly speed-up the programming. Maybe they were not able to put it in a clear well-cut logical language, as von Neumann was able to do, but it seems to our mind very probable that their accounts on this matter are true, although possibly slightly overstated.

Anyway, as is also acknowledged by Eckert and Mauchly themselves in their "progress report on the EDVAC" they wrote to make clear their contributions to the design of EDVAC, von Neumann wanted to emphasize the *logical design* of the EDVAC, replacing "*the physical structures and devices proposed by Eckert and Mauchly [...] by idealized elements to avoid raising engineering problems which might distract attention from the logical considerations under dis-*

[13] "*Either discs of the etched or alloy type may be used to remember combinations required in the conversion from the decimal to the binary system and the reverse if such a system is used. If multiple shaft systems are used a great increase in the available facilities for allowing automatic programming of the facilities and processes involved may be made, since longer time scales are provided. This greatly extends the usefulness and attractiveness of such a machine. This programming may be of the temporary type set up on alloy discs or of the permanent type on etched discs. The principal virtues of such a machine are largely due to the alloy discs which allow numbers to be stored indefinitely and to be put on and taken off by a conveniently controlled electric circuit, and that none of the mechanical parts have to accelerate or decelerate during the operation of the machine. The advantages of the electric control are not only that it allows rapid operation but that the design is simplified and capable of more readily being extended and interconnected to other apparatus.*" (quoted from the memo Eckert wrote for designing a new computer, published as an appendix in [Eck80].

cussion."[14]

Whoever had the idea of stored-programs first, it is important to understand its significance. Before the ENIAC was rewired in order to simulate stored program computers, after it had been moved from the Moore school to Aberdeen, it took days to set up a program for the ENIAC. A clear account of this *set-up problem* is given by Alt ([Alt72], p. 694):

> One of the peculiarities that distinguished ENIAC from all later computers was the way in which instructions were set up on the machine. It was similar to the plugboards of small punched-card machines, but here we had about 40 plugboards, each several feet in size. A number of wires had to be plugged for each single instruction of a problem, thousands of them each time a problem was to begin a run; and this took several days to do and many more days to check out.

[14] This is quoted from the progress report on the EDVAC by Mauchly and Eckert. The full quote, crediting von Neumann's contributions is: "*[von Neumann] has fortunately been available for consultation. He has contributed to many discussion on the logical control of the EDVAC, [and] has proposed certain instruction codes for specific problems. Dr. von Neumann has also written a preliminary report in which most of the results of earlier discussions are summarized. In his report the physical structures and devices proposed by Eckert and Mauchly are replaced by idealized elements to avoid raising engineering problems which might distract attention from the logical considerations under discussion.*" I got the quote from McCartney's book [McC99], p. 121, that emphasizes the significance of the contributions by Mauchly and Eckert. Goldstine also used this quote in his book to claim credit for von Neumann! Martin Davis took over this quote from Goldstine in his book, stating that "*[a]lthough Eckert and Mauchly later denied that von Neumann had contributed very much, shortly after they wrote as follows [...].*" ([Dav01b], p. 183). I think that it is not completely true that Eckert and Mauchly later denied that von Neumann contributed very much. The main conclusion I made after reading their [Eck80] and [Mau80] is that they were very much disappointed in not having been given their due credit. They do not however imply that von Neumann's contribution was not important. Eckert for example clearly states that he had long conversations with von Neumann on the EDVAC. He also acknowledges that the permanent set up of the instructions for the rewired ENIAC were chosen with von Neumann's consultation: "*A project was set up for operating the ENIAC with a permanent numerical code set. The permanently set up "instructions" were chosen with von Neumann's consultation, and became known as the "von Neumann code for the ENIAC."*" ([Eck80], p. 529). We think that in the end, the best way to form ones own opinion of who had what ideas first, is through a detailed research of the accounts and papers written by the people who were actually involved, looking at both sides of the story.

> When that was finally accomplished, we would run the problem as long as possible, i.e. as long as we had input data, before changing over to another problem. Typically, changeovers occurred only once every few weeks. In between we had to cope with malfunctions of the machine, usually due to dead or submarginal tubes. A faulty tube could be replaced in minutes, but it might have taken days to locate it.

Indeed, setting up a program for the ENIAC was a cumbersome task, mostly relegated to women. In Fig. 4.1.1 a picture is shown of two women programming the ENIAC. Fig. 4.1.1 shows the external function tables of the ENIAC, that would later show important for internalizing the instructions.

It was the idea to make the program instructions internal that was basic to speed-up the programming process. One of the main insights behind stored-programs was that instructions could be encoded as numbers and in this sense be manipulated on the same level as numbers, i.e., they could be altered in and by the machine itself. Eckert summarized it as follows (as one of his best computer ideas, clearly claiming credit for the idea) ([Eck80], p. 531):

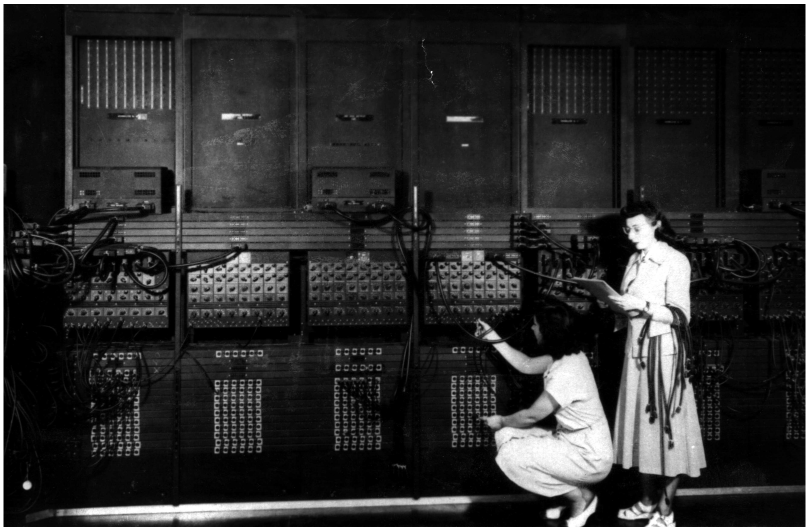


Figure 4.1: Two women programming the ENIAC

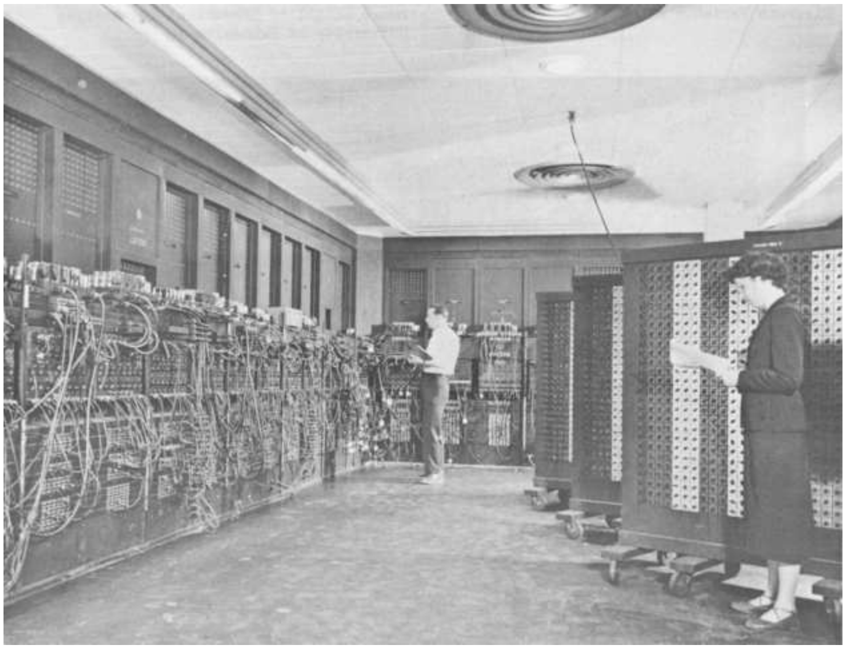


Figure 4.2: Function tables for the ENIAC (right hand-side).

My best computer idea, today briefly called "stored program," became to us an "obvious idea," and one that we started to take for granted. It was obvious that computer instructions could be conveyed in a numerical code, and that whatever machines we might build after the ENIAC would want to avoid the setup problem that our hastily built first try ENIAC made evident. It was also obvious that the functions supplied by the Master Programmer in controlling loops and counting iterations, etc., would be achieved very naturally by allowing instructions to be subject to alterations within the calculator. We even thought that Goldstine, who had frequent contact with us, understood all of the uses to which these delay lines could be put. Not so, it seems, as it turned out.[15]

[15] We would like to add some further quotes here by Eckert and Mauchly in which they describe what they meant with stored-program and state that they already had this idea before von Neumann entered the scene. "*At the Los Alamos conference I had the chance to check with Harry Huskey, who says he started on the staff of the ENIAC Project about April, 1944. I asked him whether, when he first came to the Moore school, he had heard any notions about storing*

A more explicit description is given by Alt, comparing the stored-program idea with compilers and interpreters ([Alt72], p. 694):

> In retrospect, it seems to have been a forerunner of what we now call higher-order programming languages. The idea was to encode the instructions of a problem on the "function tables," three panels of the machine, each of which bore 1200 ten-way switches. They had been intended as a computer-accessible table lookup, e.g. for empirical functions, but it was now proposed to use them to set up the succession of instructions, each represented by a two-digit number. The wiring of the plug boards would be set up permanently on the machine in a way that would cause the machine to read a number from the table, carry out the instruction encoded by it, go on to reading the next number, etc. Thus, the background wiring played the role of a present day "compiler" – more specifically, of an interpretative routine, since the source code had to be read and interpreted

programs in the same storage used for computer data. He said, "Yes. My immediate reaction was, 'Why didn't I think of that?' " But for some reason, Goldstine did not understand this, if I correctly understand what he says in his volume [Gol72]." Mauchly comments on Eckert's paper ([Eck80], pp. 531–532). "*In January 1944, I wrote a memo, Disclosure of Magnetic Calculating Machine, which I typed on my home typewriter and then gave to my supervisor for retyping. For some reason it never got typed, but I finally did get my own version back. I had also read a Master's Thesis by Perry Crawford, at MIT, where he had proposed using a disk with some spots magnetized on it for storage of numbers. My memo stated that we could use magnetic disks either erasable or permanently for the storage of information both alterable or unchangeable. The concept of general internal storage started in this memo.*" ([Eck80], pp. 530–531.) "*We conceived of another mode of ENIAC operation, in which the function tables would control "the program." The switch setting then would not represent numerical data for calculating, but arguments fed to the function tables would elicit patterns of program pulses output to prearranged program lines. There were two ways in which this might ease the burden of "patch-cord setups" – (1) if a new program were put on a function table, another program could check or verify the witch setting of the "read-only" portable panel, making it unnecessary to go through the tedious manual checking of the patchcord setups, or (2) a possibly long-term setting of the function table switches might be given a permanent set of arguments corresponding to some permanent set of program functions to be stimulated. Then every time a new set of numbers was read from the card input, a new set of operations would be caused to occur. The foregoing ideas could be easily implemented on the ENIAC, and we expected that at some time someone would want to do this, so we built the necessary cable to connect "program pulses" into the function tables in place of "digit pulses"*." ([Eck80], pp. 528–529)

> anew for each run, and no permanent object code was set up. This mode of operation would slow down the machine, of course; it was estimated that its speed would decrease at least by a factor of 5, a small price to pay for eliminating the long set-up time. We were, in effect, using ENIAC to "simulate" the future stored-program computers, which were then still on the drawing board.

The main insight for stored-programming, i.e. to regard instructions as numbers, in order to make internal manipulations possible, has a very clear resemblance to Gödel's coding: to express things about a given system in the system itself, i.e., in terms of stored programs, to encode the operations on numbers as numbers. It was this kind of feature that was also basic to Turing's construction of his universal machine: encoding operations and that on which is operated in one and the same language, through standard descriptions, was a fundamental step in Turing's construction!
The fact that this idea was, most probably, developed independent of Gödel and Turing by Mauchly, Eckert, and Zuse (see Sec. 4.1.3), in a completely different context, is to our mind of historical significance. It very clearly shows the parallelism between, on the one hand, the theoretical problems logicians were facing in the twenties and thirties, and, on the other hand, the problems "engineers" and "programmers" were facing in trying to improve their computers. They were not at all concerned with problems of how to find a good formalization of an intuitive notion. Instead they were working against the clock to find efficient methods to compute certain problems on a machine that was already then a kind of physical realization of that selfsame intuitive notion. As is very clearly expressed by Eckert, in referring to work done by Captain Grace Hopper, one of the first female programmers who developed one of the first compilers, A0, one of main reasons for them to internalize operations in the computer itself, was to reduce the human work to make the machines more efficient ([Eck80], p. 533):

> Later, she [Captain Grace Hopper] used the UNIVAC itself to work out the memory allocations. That was akin to our general philosophy, of course. You should use the computer to do all the tedious dirty work if you possibly can. That was the origin of all the languages, interpreters, and such that have since been devel-

oped.

In the meantime there was a growing need for programming languages. The reason behind developing these languages was very pragmatic. This is very clearly expressed by Captain Grace Hopper herself ([Hop81], pp. 10–11):

> There was also the fact that there were beginning to be more and more people who wanted to solve problems, but who were unwilling to learn octal code and manipulate bits. They wanted an easier way of getting answers out of the computer. So the primary purposes were not to develop a programming language, and we didn't give a hoot about commas and colons. We were after getting programs written faster, and getting answers for people faster. I am sorry that to some extent I feel the programming language community has somewhat lost track of those two purposes. We were trying to solve problems and get answers. And I think we should go back somewhat to that stage.[...] I'm hoping that the development of the microcomputer will bring us back to reality and to recognizing that we have a large variety of people out there who want to solve problems, some of whom are symbol-oriented, some of whom are word-oriented, and that they are going to need different kinds of languages rather than trying to force them all into the pattern of the mathematical logician. A lot of them are not.

Hopper clearly opposed the idea of developing languages that are forced into the pattern of mathematical logic. Indeed, although we cannot exclude the possible influence of the developments from the twenties and thirties of mathematical logic on Hopper's pioneering work, it is clear that she did not regard the earlier development of programming languages as being heavily influenced by logic. In general, as far as the development of the earliest programming languages are concerned, one can only conclude that mathematical logic must have had an influence on some pioneers in this domain, but not on everybody. For example, Zuse was acquainted with the predicate calculus when he described his Plankalkül, and acknowledged its significance for his work (see Sec. 4.1.3). However, the development of Short Code, that was first implemented on the BINAC, developed at the company by Eckert and Mauchly, was originally suggested by Mauchly himself [KP80] and the influence of mathematical logic on this language thus seems rather doubtful.[16]

Of course, we do not want to oppose the idea that the developments from the twenties and thirties discussed in the previous chapters, did not have an important influence on the beginning of the computer era. Still, we believe it important to emphasize that there have been developed techniques in the forties and early fifties, that have a very clear resemblance to certain of the techniques used by, e.g., Turing, but have arisen in a very different context: that of physical computing. Stored-programs, compilers, programming languages, etc., are developments from the beginning of the computer era that can be directly connected to mathematical logic, but cannot in every respect be traced back to mathematical logic. In a way, one could state that where both Church and Post developed certain techniques and found important results by studying their respective formalisms, many computer pioneers developed similar techniques by working with the physical version of such formalisms.

For now, it is not completely clear how many computer techniques can be called to have been developed quite independent of mathematical logic. A more detailed research would be needed here, taking into account the developments in several different places by several different people. Indeed, one should not forget here that there was research on computing in many different countries by several different research groups. As a consequence many techniques were re-invented in different places by different people. In this sense, research connected to the first computers is in fact yet another examples of a confluence of ideas, of which some were very explicitly influenced by the developments sketched in the previous chapters, and some were not.

The influence of the results by Gödel, Turing and others on "Johnny" von Neumann cannot be neglected. He knew all these papers and had been there in 1930 at the conference in Königsberg where Gödel announced his incompleteness results for the first time. He was also fully aware of Turing's paper and

[16] It should be noted however that mathematical logic has had a very important influence on the development of the first "true", more high-level, programming languages. For example, λ-calculus has played a significant role in the development of LISP, and John Backus has stated that Post's canonical forms have had an influence on the development of the Backus-Naur form, important for ALGOL60 (see for example [Bac80]).

acknowledged this in some of his papers and lectures on computers. As is argued in [Hod83] and [Dav87], based on a letter by Ulam to Hodges ([Hod83], p. 145),[17] von Neumann must have read Turing's paper before the outbreak of the war.

After Gödel's results von Neumann wanted to stay far away from logic. But then he got involved with the ENIAC project, and with "real" computing. In this context von Neumann's logical background would prove very useful, although it was not an interest in logic that triggered his interest in the subject. In [Ula80], Ulam explains why von Neumann got interested in computers (pp. 93–94).

> It must have been in 1938 that I first had discussions with von Neumann about problems in mathematical physics, and the first I remember were when he was very curious about the problem of mathematical treatment of turbulence in hydrodynamics. [...] He was fascinated by the role of Reynolds number, a dimensionless number, a pure number because it is the ratio of two forces, the inertial one and the viscous, and has the following importance: When its value surpasses a critical size, about 2000, the regular laminar flow, as it is called, becomes highly irregular and turbulent. [von Neumann] [...] wanted to find an explanation or at least a way to understand this very puzzling large number. Small numbers like π and e, are of course very frequent in physics, but there is a number of the order of thousands, and yet it is a pure number with no dimensions: it does whet your curiosity. I remember that in our discussions von Neumann realized that the known analytical methods, the method of mathematical analysis, even in their most advanced forms, were not powerful enough to give any hope of obtaining solutions in closed form. This was perhaps one of the origins of his desire to try to devise methods of very fast numerical computations, a more humble way of proceeding. Proceeding by "brute force" is considered by some to be more lowbrow. [...]
> I remember also discussions about the possibilities of predicting the weather at first only locally, and soon after that, about how to calculate the circulation of meteorological phenomena around the globe.

[17] This letter is available on-line through Andrew Hodges website on Turing: http://www.turing.org.uk/sources/vonneumann.html

von Neumann was thus particularly interested in computers for doing numerical calculations in the context of theoretical physics.[18]
In 1944 he got involved with the ENIAC project, and the plans for developing another computer, the EDVAC. Indeed, the main design ideas for the EDVAC were described by von Neumann in the first draft of this machine [vN45]. As we already know, it is not completely clear how much of the ideas described in this draft originated from Eckert and Mauchly, but, the emphasis on the logical aspects of the EDVAC clearly came from von Neumann. He also made important contributions to the rewiring of the ENIAC at Aberdeen, since the permanent set of instructions to be internalized were chosen with von Neumann's consultation. The main design of the rewiring was done by R.F. Clippinger, who states in the introduction of a report describing the new coding system for the ENIAC, that the rewiring was suggested by von Neumann and that "*[t]he role of J. von Neumann in working out the details has been a central one* ([Cli48]).[19]
von Neumann's interest in the *logical design* of computers, is also expressed by Eckert ([Eck80], p. 525):

> We thought the most important development problem we faced in the ENIAC was to provide a control system consistent with and adequate for its general-purpose use. And it was about the controls of the computer that von Neumann first asked when he came in September 1944, for his first visit to the ENIAC project. If he had first asked questions like "How fast does it work?" we would have been disappointed. Because he asked about the control logic, there was an immediate

[18] The use of computers in mathematics and physics, as regarded by von Neumann, will be discussed in more details in the next section.

[19] Neukom's paper [Neu06] gives a detailed description of "the ENIAC's second life". It should also be noted here that although the account on the rewiring of the ENIAC – von Neumann suggesting the idea and Clippinger having detailed out the design – seems to be the generally accepted account, in the end, Clippinger himself has stated the significance of von Neumann here, Metropolis tells us a slightly different story: "*In the meantime Richard Clippinger, a staff member at Aberdeen, had suggested that the ENIAC had sufficient flexibility to permit is controls to be reorganized into a more convenient (albeit static) stored-program mode ofoperation. [...] Although implementing the new approach is an interesting story, suffice it to say that Johnny's wife, Klari, and I designed the new controls in about two months and completed the implementation in a fortnight.*" ([Met87], p. 128).

> rapport.

Although Eckert and Mauchly most probably already had the idea of stored programs, it is von Neumann who was able to convert it into logical terms, independent of whether he got this idea through Mauchly and Eckert or not, and very clearly understood the link between problems related to computing with those occurring in logic. We think this is the most important contribution by von Neumann: he understood that computers can be considered as *logical machines*, or, as Martin Davis has called it, *engines of logic* ([Dav87], p. 166).
According to Martin Davis [Dav87, Dav01b] this is one basic advance of the computers of the postwar period ([Dav87], p. 166):

> The computers of the postwar period differed from previous calculating devices in having provision for internal storage of programs as well as data. They were conceived, designed, and constructed, not as mere automatic calculators, but as *engines of logic*, incorporating the general notion of what it means to be computable and embodying a physical model of Turing's universal machine.

This is maybe a slight overstatement from Davis's side, since it obscures the many different roads that led to the modern computer. However, there is some clear truth in this statement, especially with respect to von Neumann's contributions, and, as we will later see, Turing's.
The EDVAC design is often considered as laying the basis of present day computers and has come to be known as the von Neumann architecture. We will not discuss this design in detail, but it is important to ask in how far this design could have been influenced by the developments discussed in the previous chapters. Von Neumann had a broad and solid background in these matters and must have read Turing's paper. In this sense it only seems "logical" that, especially Turing's paper, must have had an influence here. Davis has in fact argued that the concept of a universal Turing machine must have had a significant influence on von Neumann's conception of computers as logical machines, as reflected in his *first draft.* A clue is given by the only reference in von Neumann's text, i.e., the paper by McCullough and Pitts [MP43]. As is pointed out by Martin Davis, the reference to this paper gives a very direct link with

Turing's universal Turing machine: not only did McCullogh later state that their paper was in fact directly inspired by Turing's, but the paper itself states that the possibility of representing a universal Turing machine in their neural model, is in fact the main reason for believing in the adequacy of the formalism. Davis has given some further arguments for the significance of the universal machine concept in von Neumann's work on computers.[20]

That logic has had a very significant influence on von Neumann's work on computers is obvious when reading several of his papers. In the introduction to von Neumann's work on Natural and artificial automata [Bur86] in the volume [vN86], Burks discusses several of these influences. The first such influence Burks mentions concerns the influence of Gödel's work on von Neumann's programming methods ([Bur86], p.382–383):

> There are some concepts in von Neumann's program codes and programming methods that are analogous to logical ideas that Gödel employed. I think it likely that, in his programming work, von Neumann was guided by his knowledge of Gödel's work, at least intuitively.

Burks mentions two aspects of von Neumann's work that might have been influenced by Gödel. The first concerns the distinction between metalanguage and object language ([Bur86], p. 383):

> In his program codes for the EDVAC and IAS machines, von Neumann used partial substitution instruction for changing the addresses in a program during computation [...] Arithmetic instructions and partial substitution instructions both transform words, but they differ in this: The address in an arithmetic instruction refers to a word usually interpreted as a number, whereas the address in a partial substitution instruction refers to another instruction. This is an instance of the metalanguage versus object language distinction.

This technique of partial substitution is one way to implement the stored program idea. The technique shows how data and instructions can be manipulated on the same level. In the following quote, one sees how von Neumann understood this substitution technique as a way for the computer to manipulate its

[20] The interested reader is referred to [Dav87], pp. 167–169 and [Dav01b], pp. 180–193.

own code. The quote is also interesting because this ability of the computer is considered as a necessary feature to obtain flexible codes, and is in fact one of the features that makes coding a non-trivial operation according to von Neumann. We give the full quote, including the explanation of how the process of substitution works ([GvN46], p. 31–32):

> It should be added here that there are two ways to send a number a from the arithmetic organ to the memory, say to the memory position y. We either want to place the entire 40 digit number a to occupy the entire space at y, or there may be two orders at y, and we may only want to replace the memory-position-reference x in one of these orders by part of a. Since we plan to have $4096 = 2^{12}$ viewed as a binary digit number, hence it will require 12 digits of a, say the 12 last ones (to the right). In view of this possibility we may also call the disposal orders *substitutional orders*. The first use (40 digits of a moved) is a *total substitution*, the second use (12 digits of a moved) is a *partial substitution*, and according to whether the first or the second order at y is modified, the partial substitution is *left* or *right*. It should be added that this technique of automatic substitutions into orders, i.e. the machine's ability to modify its own orders (under the control of other ones among its orders) is absolutely necessary for a flexible code. Thus, if a part of the memory is used as a "function table", then "looking up" a value of that function for a value of the variable which is obtained in the course of the computation requires that the machine itself should modify, or rather make up, the reference to the memory in the order which controls this "looking up", and the machine can only make this modification after it has already calculated the value of the variable in question. On the other hand, this ability of the machine to modify its own orders is one of the things which makes coding the non-trivial operation which we have to view it as.

As compared to some of the quotes in which Eckert explains the stored program idea, von Neumann's explication of its use is stated in more logical terms, and resembles more not only Gödel's ideas but also those Turing's. The ability of the machine to manipulate its own code is an idea that becomes very explicit in Turing's universal machine. Indeed whereas Eckert uses terms such as "digit pulse" and "program pulse", i.e. engineering terms, von Neumann applies the much more logical terminology of substitution.

A further influence of Gödel's work on von Neumann that Burks identifies, is summarized in the following quote ([Bur86], p. 383):

> In his programming procedures von Neumann explicitly uses the distinction between bound and free variables, and his single program loops are analogous to Gödel's bounded quantifiers.

We will not discuss this last influence in any further detail. Let me merely point out that the differentiation between bound and free variable was effectively used by von Neumann,[21] while the form of bounded quantifier expressions is considered by Burks to be quite similar to certain routines, like e.g. a *for loop.* One can of course question whether these influences of logic on von Neumann, as pointed out by Burks, are solely due to his knowledge of Gödel's work, but, in any way, it is clear that formal logic did have a significant influence on von Neumann's work on coding and thus on early computing.

Much of von Neumann's papers in the late forties are indeed devoted to coding. In collaboration with Goldstine, he wrote a sequence of reports, *Planning and Coding of Problems for an Electronic Computing Instrument,* in which von Neumann develops methods for coding problems [GvN47, GvN48a, GvN48b]. If one scans through these reports, the influence of logic can hardly be neglected,

[21] In [GvN47], pp. 90–91, von Neumann indeed uses this terminology: "*A mathematical-logical procedure of any but the lowest degree of complexity cannot fail to require* variables *for its description. It is important to visualize these variables are of two kinds, namely: First, a kind of variable for which the variable that occurs in an induction [...] is typical. Such a variable exists only within the problem. It assumes a sequence of different values in the course of the procedure that solves this problem, and these values are successively determined by that procedure as it develops. It is impossible to substitute a value for it and senseless to attribute value to it "from the outside". such a variable is called (with a terms borrowed from formal logic) a* bound variable *Second, there is another kind of variable for which the parameters of the problem are typical – indeed it is essentially the same thing as a parameter. Such a variable has a fixed value throughout the procedure that solves the problem, i.e. a fixed value for the entire problem. If it is treated as a variable in the process of planning the coded sequence, then a value has to be substituted for it and attributed to it ("from the outside"), in order to produce a coded sequence that can actually be fed into the machine. Such a variable is called (again, borrowing a term from formal logic) a* free variable."

given the logical terminology used. In these papers one finds the development of *flow diagrams*, as a means to represent algorithms in a precise and more structured way, at a higher level than the machine language. Because these reports were distributed to ([KP80], p. 208):

> [...] the vast majority of people involved with computers at that time [...] coupled with the high quality of presentation and von Neumann's prestige, [...] their report had an enormous impact, forming the foundation for computer programming techniques all over the world.

Given the clear influence of mathematical logic in these reports, one cannot underestimate the significance of logic in the domain of programming.
In an earlier paper, we already quoted from, von Neumann makes clear how the "coder" should proceed in developing a program ([GvN46] p. 30):

> In addition to a quite flexible set and general set of basic orders that can be understood by his machine, the coder needs certain further things: An effective and transparent logical terminology or symbolism for comprehending and expressing a particular problem, no matter how involved, in its entirety and in all its parts; and a simple and reliable step-by-step method to translate the problem (once it is logically reformulated and made explicit in all its details) into the code.

From this quote it is very clear what value von Neumann attached to logic, not only with respect to the design of computers, but also with respect to programming. A last quote we want to mention here that illustrates the significance of logic for von Neumann's work on programming, was also used by Martin Davis in this context ([GBvN46], p. 37):

> It is easy to see by formal-logical methods that there exist codes that are *in abstracto* adequate to control and cause the execution of any sequence of operations which are individually available in the machine and which are, in their entirety, conceivable by the program planner. The really decisive considerations from the present point of view, in selecting a code, are more of a practical nature: simplicity of the equipment demanded by the code, and the clarity of its application to the actually important problems together with the speed of its handling of those problems.

The first sentences of this quote seem to reflect the universal Turing machine concept stated in terms of programs.

The influence of Turing's universal machine on von Neumann's work, seems also to be reflected by von Neumann's understanding that a small programming vocabulary is not a problem, i.e., a few dozen of instructions are enough to express all of mathematics (at least the computational part of it). This is recounted by Alt, who attended a lecture by von Neumann at the first meeting of the Association for Computing Machinery at Abderdeen Proving Ground in 1947 ([Alt72], p. 694):

> [von Neumann] discussed the need for, and likely impact of, electronic computing. He mentioned the "new programming method" for ENIAC and explained that its seemingly small vocabulary was in fact ample: that future computers, then in the design stage, would get along on a dozen instruction types, and this was known to be adequate for expressing all of mathematics. (Parenthetically, it is as true today as it was then that "programming" a problem means giving it a mathematical formulation. Source languages which use "plain English" or other appealing vocabularies are only mnemonic disguises for mathematics.) von Neumann went on to say that one need not be surprised at this mall number, since about 1000 words were known to be adequate for most situations of real life, and mathematics was only a small part of life, and a very simple part at that. This caused some hilarity in the audience, which provoked von Neumann to say: "If people do not believe that mathematics is simple, it is only because they do not realize how complicated life is."

The expressibility of the whole of mathematics by a few dozen of instructions very clearly resembles the ambitions of many a logician at the beginning of the 20th century, including Post. However, von Neumann knew of the unsolvability results published ten years before, and is probably pointing at the ability of a universal Turing machine to compute anything computable by any other Turing machine. Some years later, during the second lecture of a sequence of lectures delivered at the University of Illinois in 1949, he would explicitly state that the significance of Turing's work lies in his proof of exactly this fact ([vN66], p. 50):

> $\overline{A}$ [a universal machine] is able to imitate any automaton, even a much more complicated one. Thus a lesser degree of complexity in an automaton can be compensated for by an appropriate increase of complexity of the instructions. The importance of Turing's research is just this: that if you construct an automaton right, then any additional requirements about the automaton can be handled by sufficiently elaborate instructions. This is only true of A is sufficiently complicated, if it has reached a certain minimum level of complexity. In other words, a simpler thing will never perform certain operations, no matter what instructions you give it; but there is a very definite finite point where an automaton of this complexity can, when given suitable instructions, do anything that can be done by automata at all.

Later, he would also state that the universal Turing machine lies at the basis of the construction of a self-reproducing automaton, i.e., von Neumann's version of cellular automata [vN51]. More generally, as is e.g. expressed in the above mentioned lectures, von Neumann considered logic to be a basic part of the development of a theory of automata. However, given the physical nature of "artificial automata" and thus the fact that they will malfunction and make mistakes, such theory should also include statistical considerations. To this end, he wanted to develop a kind of probabilistic logic, which was described in his [vN56].

4.1.2 Alan Turing's work on computers and programming

In the previous chapter we showed how Turing's analysis of the process of a man computing led him to the formulation of his Turing machines. It was his construction of a universal Turing machine von Neumann regarded as one of the most significant results of Turing's paper in the context of building real computers, and was thus an important influence on von Neumann's work in this context. Turing himself also got involved in the project of designing a "real" universal machine, i.e., a digital general-purpose stored program computer, and, was well-aware of the fact that these computers are in fact the physical counterpart of his universal machine.

Some years after the publication of his seminal paper, it was war, and Turing started to work at Bletchley park as a cryptanalyst. As is described in [Hod83] Turing made many important contributions during the war, his background in mathematical logic and the idea of automated processes being "extraordinarily relevant" in this context. One of his first successes was his contribution to the development of the *British Bombe*, a device more general than the Polish Bomba in that it was capable of breaking all German Enigma message. The crucial contribution by Turing to this generalization was the mechanization of certain logical deductions ([Hod83], pp. 179–181):

> It was Alan [...] who first formulated the principle of mechanising a search for logical consistency based on a 'probable word'. The Poles had mechanised a simple form of recognition, limited to the special indicator system currently employed; a machine such as Alan envisaged would be considerably more ambitious, requiring circuitry for the simulation of 'implications' flowing from a plugboard hypothesis, and means for recognising not a simple matching, but the appearance of a contradiction. [...] The idea of automating processes was familiar enough to the twentieth century; it did not need the author of *Computable numbers*. But his serious interest in mathematical machines, his fascination with the idea of working like a machine, was extraordinarily relevant. Again, the 'contradictions' and 'consistency' conditions of the plugboard were concerned only with a decidedly finite problem, and not with anything like Gödel's theorem [...] But the analogy with the formalist conception of mathematics, in which implications were to be followed through mechanically, was still a striking one.

Turing also played an important role in breaking the *Fish* code, messages that had a special encoding used for Hitler's communications. The code was finally cracked by the Colossus machines.[22]

It was also during the war that Turing had the occasion to build up his knowledge of electronic technology and he even developed his own speech secrecy system, with the aid of Donald Bailey, called *Delilah*.[23]

[22] The reader is referred to [Hod83] for more details on Turing's work during the war.

[23] A transcription of a report dated 6 June 1944 by Turing, with the title *Speech System 'Delilah' - Report on Progress*, can be found at Andrew Hodges website: http://www.turing.org.uk/sources/delilah.html.

So to what extent did Turing's theoretical work on computing machines, i.e. his 1936 paper, influence the development of early computing? We already know that his work must have had an important influence on von Neumann's work on computers. Turing's work also played its role in the development of the Colossi machines, special-purpose machines, that were developed at the Post Office Research Station, Dollis Hill, in close collaboration with people from Bletchley Park. In [Ran80] Randell discusses the influence Turing's mathematical work might have had on these machines. Randell's paper starts with a quote from the explanatory caption accompanying a set of photographs of COLOSSUS that were only made available by the British governement in 1975 ([Ran80], p. 48):

> Babbage's work in 1837 first established the logical principles of digital computers. His ideas were developed further in Turing's classical paper in 1936. The COLOSSUS machine produced by the Department of Communications of the British Foreign Office, and put into operation in December 1943, was probably the first system to implement these principles successfully in terms of contemporary technology [...] The requirement for the machine was formulated by Professor M.H.A. Newman, and the development was undertaken by a small team led by T.H. Flowers. A. Turing was working in the same department at that time, and his earlier work had its full influence on the design concept.

Although this quote states the influence of Turing's paper on the design of the COLOSSUS, it is far from clear how far this influence actually went. As is noted by Randell – whose main information came from several interviews with people involved, since, at that time, not much documents had been made available yet – Turing's influence on the Colossi machines should not be overestimated. However, his 1936 paper was well-known ([Ran80], p. 78):

> [...] the early projects that the Dollis Hill people carried out for Bletchley Park, were done in close cooperation with Turing [...] Apparently he did not have any direct involvement in, or influence on, the design or use of COLOSSUS. His visits to Dollis Hill occurred prior to the start of the COLOSSUS work, and Newman does not remember his presence at any of the meetings that that Newman and Flowers held at Bletchley Park, Turing's prewar work on computability was well

> known, and virtually all of the people I have interviewed recollect wartime discussions of his idea of a universal automaton.[...] Good has written that "Newman was perhaps inspired by his knowledge of Turing's 1936 paper". However, Newman's view now is that although he and his people all knew that the planned COLOSSUS was theoretically related to a Turing machine, they were not conscious of their work having any dependence on either these ideas or those of Babbage.

It is thus clear that Turing's paper might have played its role in the development of the Colossi machines, but it is not completely clear how far this influence extends.
More important here is Turing's own work on computing machines. By the end of the war, J.R. Womersley who had become a member of the National Physics Laboratory (NPL) in the U.K. and headed the new mathematics division of the NPL, made a trip to the U.S. He was allowed access to the brand new ENIAC and informed of the EDVAC report. He already knew Turing's paper, and, after he returned from his visit to the U.S. he made arrangements with Newman to meet with Turing. Womersley hired Turing and some months later Turing produced his ACE report [Tur45], describing the design of an *Automatic Computing Engine*, a *general-purpose digital computer with stored programs*.[24] In the introduction of the ACE report, Turing makes explicit the typical feature of the ACE as compared to other machines ([Tur45], p. 20):

> Calculating machinery in the past has been designed to carry out accurately and moderately quickly small parts of calculations which frequently recur. [...] It is intended that the electronic calculator now proposed should be different in that it will tackle whole problems. Instead of repeatedly using human labour for taking material out of the machine and putting it back at the appropriate moment all this will be looked after by the machine itself.

Turing then sums up three basic advantages of his ACE. We will not give them here, but it is maybe interesting to point out that they concern the elimination

[24] The paper by Carpenter and Doran [CD77] gives a detailed analysis of Turing's ACE report and makes a comparison between Turing's report and von Neumann's first draft for the EDVAC.

of human work, to speed-up the computing process and to avoid errors. In the spirit of his 1936 paper, he also makes the analogy between what a computer needs to compute, as compared to what a man needs in computing ([Tur45], p. 20–21):

> It is evident that if the machine is to do all that is done by the normal human operator it must be provided with the analogues of three things, viz. firstly, the computing paper on which the computer writes down his results and his rough workings; secondly, the instructions as to what processes are to be applied; these the computer will normally carry in his head; thirdly, the function tables used by the computer must be available in appropriate form of the machine. These requirements all involve *storage of information* or *mechanical memory*.

The design of the ACE was based on the idea of stored programs. The fact that the human work could be reduced to a minimum was considered as one of the basic advantages of this computer. Indeed, all the human operator has to do is to write the program and feed it to the machine, which will then do all the work ([Tur45], p. 21):

> It is intended that the setting up of the machine for new problems shall be virtually only a matter of paper work. Besides the paper work nothing will have to be done except to prepare a pack of Hollerith cards in accordance with this paper work, and to pass them through a card reader connected with the machine. There will positively be no internal alterations to be made even if we wish suddenly to switch from calculating the energy levels of the neon atom to the enumeration of groups of order 720. It may appear somewhat puzzling that this can be done. How can one expect a machine to do all this multitudinous variety of things? The answer is that we should consider the machine as doing something quite simple, namely carrying out orders given to it in a standard form which it is able to understand.

As is clear from these quotes, Turing regarded the ACE as a true general-purpose stored program computer, with a clear focus on the *programmability* of the machine such that it can solve all different kinds of problems. Indeed, Turing conceived of the ACE as a machine that should be able to solve a variety of problems, i.e., “*those problems which can be solved by human clerical labour, working to fixed rules, and without understanding [...]*”. Among the examples of problems that Turing considers his machine to be capable to solve, are, jigsaw puzzles and playing chess. Given the general-purpose character of the ACE in the way Turing understood it, and the consequent need for the ACE to be a truly programmable machine, one can contrast Turing’s design with von Neumann’s first draft. As Martin Davis points out ([Dav01b], p. 188):

> Turing’s ACE was a very different kind of machine from von Neumann’s EDVAC, corresponding closely to the different attitudes of the two mathematicians. Although von Neumann was concerned that his machine be truly “all-purpose,” his emphasis was on numerical calculation and the logical organization of the EDVAC (and of the later johnniacs) was intended to expedite this direction. Since Turing saw the ACE being used for many tasks for which heavy arithmetic was inappropriate, the ACE was organized in a much more minimal way, closer to the Turing machines of the *Computable numbers* paper. Arithmetic operations were to be carried out by programming – by software rather than hardware. For this reason, the ACE design provided a special mechanism for incorporating previously programmed operations in a longer program [i.e. the use of stacks]

Turing clearly knew the EDVAC report and even recommends the reader to read his report in conjunction with von Neumann’s.
In a lecture to the London Mathematical Society on 20 February 1947 [Tur47], Turing made an explicit comparison between his ACE and the universal Turing machine, and it is thus clear that, as far as Turing is concerned, he indeed understood the ACE as a truly logical machine, or, an engine of logic [Dav87].
After having sketched the advantages of going digital, i.e., greater accuracy and applicability to a wide range of problems, Turing says ([Tur47], p. 106–107):

> Some years ago I was researching on what might now be described as an investigation of the theoretical possibilities and limitations of digital computing machines. I considered a type of machine which had a central mechanism, and an infinite memory which was contained on an infinite tape. This type of machine appeared to be sufficiently general. One of my conclusions was that the idea of

> a ‘rule of thumb’ process and a ‘machine process’ were synonymous.[...] It was essential in these theoretical arguments that the memory should be infinite [...] Machines such as the ACE may be regarded as practical versions of this same type of machine. There is at least a very close analogy. Digital computing machines all have a central mechanism or control and some very extensive form of memory. The memory does not have to be infinite, but it certainly needs to be very large.

Turing thus clearly conceived of his ACE as a universal machine. But this is not where it stopped for Turing.

Although his Turing machines were developed with the idea of finding a formal equivalent of the process of a man computing a number, and the ACE could thus be regarded as the physical embodiment of this process, Turing no longer wanted to restrict these computing machines to computability of numbers by a human being, but wanted to know what else these machines are capable of. During his lecture he made this idea very explicit and later devoted several papers to the subject of the possibility of building *intelligent machinery* [Tur69, Tur50]. To Turing’s mind, the machine should be educated just as a child needs training. And as a child, and any adult human being, often makes mistakes, this machine should also be allowed to make mistakes. Turing in fact regarded the possibility of the machine to make mistakes, as a precondition for it to become intelligent ([Tur47], pp. 123–124):

> [...] I would say that fair play must be given to the machine. Instead of it sometimes giving no answer we could arrange that it gives occasional wrong answers. But the human mathematician would likewise make blunders when trying out new techniques. It is easy for us to regard these blunders as not counting and give him another chance, but the machine would probably be allowed no mercy. In other words then, if a machine is expected to be infallible, it cannot also be intelligent.

The ACE was never built as Turing had envisioned it. After several difficulties with finding resources and the right people to effectively build the ACE, due to bad management at the NPL, Turing had had enough and in the end accepted

a job at Manchester University. A small version of the ACE was finally built, called the pilot ACE, but Turing was not really involved in its actual construction. In the meantime several computers were being built at several locations. In Manchester, the Mark I was constructed. Instead of being involved in its construction, Turing now really began to use computers to do research, directly programming in the binary machine language.
In 1951, Turing wrote what might be the first programming book, for the new Mark II computer in Manchester. We will not discuss this book in any detail here, but, it is important to note that Turing again made an explicit analogy between human and machine computers, in the spirit of his 1936 paper. However, as was also the case for the ACE, focus is now much more on the software aspect of the machine, rather than on hardware, emphasizing that the machine itself should not be too complicated, since it only has to obey instructions that can be made explicit enough, without the need for complicated hardware. We will give a rather long quote from the programmer's book here, because it beautifully shows Turing's way of reasoning ([Tur51b]):[25]

> Electronic computers are intended to carry out any definite rule of thumb process which could have been done by a human operator working in a disciplined but unintelligent manner. The electronic computer should however obtain its results very much more quickly. The human computer with whom we are comparing it may be imagined as supplied with various computing aids. He should have a desk machine, paper to write his results on, and more paper on which is written a detailed account of how the calculation is to be carried out. These aids have their analogous in the electronic computer. The desk machine is transformed into the computing circuits, and the paper becomes "the information store" or more briefly the "store", whether it is paper used for results or paper carrying instructions. There is also a part of the machine called the control which corresponds to the computer himself. If his possible behaviour were very accurately represented this would have to be a formidable complicated circuit. However we really only require him to be able to obey the written instructions and those can

[25] We would like to thank the creators of the Turing digital archive, available at http://www.turingarchive.org/, for having made available many texts by Turing.

> be made so explicit that the control can be quite simple. There remain two more components of the electronic computer. These are the input and output mechanisms, by which information is to be transferred from outside into the store or conversely. If the analogy of the human computer is to be maintained these parts would correspond to his ears and voice, by which he communicates with his employer. [...] The information stored on paper by the human computer will mostly consist of sequences of digits drawn from 0, 1, ..., 9. There may also be other symbols such as decimal points, spaces, etc. and there may be occasional remarks in English, Greek letters etc. There may in fact be anything from 10 to 100 different symbols used, and there is no particular need to decide in advance how many different symbols will be concerned. With an electronic computer however such a decision has to be made; the number of symbols chosen is ruled very largely by engineering considerations, and with the vast majority of machines the number is two. [...] It is not difficult to see that information expressed with one set of symbols can be translated into information expressed with another set of suitable conventions [...] Although we shall not need these translation conventions we shall often wish to interpret a sequence of 0's and 1's as meaning some integer.[...] Although the scale of two is appropriate for use within an electronic computer it is not so suitable for work on paper, and it is not possible to avoid paper work altogether. Without attempting to explain the reasons at this stage let us accept that there are occasions when it is desirable to write down on paper the sequence of symbols stored in some part of the machine. Suppose for instance that the sequence was 100011101110100010011000111001010101101100100110. The copying of such sequence is slow and very liable to inaccuracy. It is very difficult to 'keep one's place'. It is therefore advisable to represent such a sequence on paper in a different form not subject to these difficulties. The method chosen is to divide the sequence into block of five [...] and then to replace each block by a single symbol, according to the table below. The above given sequence then becomes z"SLZWRFWN.[...]

As is clear from this quote, Turing was very much aware of the fact that limiting the number of symbols *in advance* is not a necessary restriction for humans, but it is necessary for the machine. Since it was very obvious for Turing that

one can translate any information expressed with a given set of symbols to another set of symbols, the restriction of the binary alphabet is in no way fundamental. The fact however, that it is not very convenient to work out a program in binary for us humans, it becomes important to develop an intermediary language between humans and computers, i.e., replacing blocks of binary digits by other symbols, since this makes the programming more efficient. Or, to state it in Hopper's terms: "*We were after getting programs written faster, and getting answers for people faster.*" Nowadays, the language Turing describes in his programming book would not really be regarded as very much adapted to the user, since it is still rather close to machine language. Indeed, it has become very usual to replace the blocks of bits by human language words or symbols that are recognizable by most humans, like "if", "next", "+",...
It is always interesting to go back to the roots of something. In case of programming languages it is significant to see how programming evolved from physical rewiring to languages that got further and further removed from the machine language. In fact, one might well say that the Graphical User Interface, is the "programming" most people are used to nowadays. We regret the fact that most have completely forgotten what is actually going on when they e.g. push a mouse button to select a sequence of letters in a word document. It is probably one of the reasons why people can't stand it when their computer makes mistakes: they no longer know the machine that has become part of their lives.

For now we have only looked at the developments of the earliest computers in the U.S. and the U.K. As should be clear, some of these developments can be connected with the developments from the previous chapters, others can't. Before drawing any definite conclusion in this context, it is important to have a closer look at the development of computers by the "enemy": the Germans, centered around the work of one person, Konrad Zuse.

4.1.3 Zuse's Z1, Z2, Z3, Z4 and Plankalkül

In [Bau80], Bauer very clearly describes the very different situation Konrad Zuse was facing at the end of the war ([Bau80], p. 505–506):

> In April 1945, a truck left Göttingen, heading for Bavaria. It carried an instrument that had been built in Berlin during the war for the Aerodynamische Versuchsanstalt [...] and had been brought to their Göttingen laboratory a few weeks before. Here it had been put into operation for the first time. But then, the Russian army approached Göttingen. The instrument had the code word V4 (Versuchmodell 4) and because of the parallel with V1 and V2, the code word for buzz bombs, the man who had built the instrument got permission to bring it "in Sicherheit." The adventurous journey via Hof, München, and Ettal ended at the village of Hinterstein near Hindelang, a small town in the Bavarian Alps, in a province called Algäu, near the Austrian border. A few days later, North African troops of the French army occupied Hinterstein. They found may things, but not the instrument that was hidden in a cellar. [...] In the winter of 1944-1945 a Swiss soldier was on duty in the Rätikonand Silvretta mountains, at the border of Austria, some 50 miles away from Hinterstein [...] Later he would use the instrument the fugitive from Prussia had built. This fugitive was Konrad Zuse [...] He had [...] started in 1934 to build a computer that could ease the calculations of statics. V4 was his fourth model; V3, which was destroyed in 1944 in a bomb attack, was the first fully programmable computer when it became operational in 1941. [...]

Zuse was born in Berlin. In 1935 he got his master degree in engineering at the Technische Hochschule Berlin-Charlottenburg (nowadays, Technische Universität Berlin). As an engineer, Zuse was often confronted with pure calculations, which he found too time-consuming. It was exactly this tremendous work involved in making certain calculations, that led Zuse to the idea of constructing machines that could take over this laborious task ([Zus80], pp. 611–612):

> I was a student in civil engineering in Berlin. Berlin is a nice town and there were many opportunities for a student to spend his time in an agreeable manner, for instance with the nice girls. But instead of that, we had to perform big and awful calculations. Also later as an engineer in the aircraft industry I became aware of the tremendous number of monotonous calculations necessary for the design of static and aerodynamic structures. Therefore, I decided to design and construct calculating machines suited to solving these problems automatically.

Zuse can be considered as a real computer pioneer. His work has to our mind been neglected too much, standing in the shadow of the more heroic work by the Allies. In a way he had bad luck to be born in Germany. Zuse's work very clearly shows how engineers should not necessary be regarded as non-logicians who do not take into account logical considerations leading to more "elegant" designs for computers, and vice versa.

Contrary to Eckert, Mauchly, Turing and von Neumann, Zuse worked in relative isolation, did not have many resources, and, quite naturally, was unaware of the Top Secret work of the Allies. It is thus the more amazing to see how many ideas he invented and implemented. At the time he developed his major ideas, Zuse was neither aware of Turing's *On computable numbers*, nor of Babbage's work.[26].

The Z1, a mechanical computer, was constructed in his parent's living room, and finished in 1938. The machine used punched cards, included a 2-dimensional storage, a selection mechanism that connects storage locations with the arithmetic unit, and a control unit. Significant to note is that, already in the first machine he developed, Zuse used binary representations instead of decimals! The machine did not work well [Zus80], except for the storage unit, and Zuse decided to change to electromechanical technology, using relays. The work on this relay machine, the Z2, was started in 1937.

Unlike Mauchly and Eckert, already from the beginning, Zuse wanted to base the development of his computers on a solid theoretical foundation. He developed what he called a *Bedingungskombinatorik*, combinatorics of conditions, he could use to easily describe circuits. His former mathematics teacher read a report describing the calculus Zuse developed for this purpose, and advised Zuse to read the books by Hilbert and Ackerman, Hilbert and Bernays, Frege

[26] This is acknowledged by Zuse in his autobiography: "*When I began to build the computer, I neither understood anything about computing machines nor had I ever heard of Babbage. It was only many years later, when my constructions and switches were basically set, that an examiner from the American Patent Office showed me Babbage's machines. The otherwise extremely thorough German examiners had not been acquainted with Babbage.*" ([Zus93], p. 34) "*It also illustrates just how hard I tried to build bridges between theoretical logic and practice. Unfortunately, I was not yet familiar with the then already published work of Turing [Tur37].*" ([Zus93], p. 53).

and Schröder, which he did [Bau80]. Having made himself more familiar with propositional calculus and predicate calculus, he determined that the propositional calculus was basically isomorphic to his *Bedingungskombinatorik*, but he found that the mathematics had been worked out more exactly. Furthermore, propositional calculus provided him with rules he was not familiar with [Zus93]. Having made himself more acquainted with logic as it already existed at that time, he could now rewrite his calculus in these terms. This resulted in *Einführung in die allgemeine Dyadik* [Zus37],[27] which is in fact as revolutionary a paper as Shannon's thesis [Sha38].[28] As Zuse explains ([Zus80], p. 614–615):

> Right from the beginning I tried to base the whole development on a new and solid theoretical foundation. At first, the analogies between switching circuits and the calculus of propositions were discovered and a switching algebra was set up.[...] Unfortunately, I never published my ideas concerning this matter. Later on I learned that there were some papers, two in German language by Hansi Piesch and Eder, and one in English language by Shannon. But I missed there the consequent confrontation with the calculus of propositions. For us the terms "And", "Or", "Not" belonged to our daily language. We really worked with them and made the step to apply the mathematical logic to the computer design. I translated the logical rules systematically into switching algebra. [...] So, switching algebra was consequently applied in all the computers we constructed. When Schreyer changed to electronic technology he first had only to design the switching elements corresponding to the three propositional operations: conjunction, disjunction, and negation. After that he was able to translate one to one the already proven diagrams for the electromechanical machines.

[27] We would like to thank the creators of the Zuse digital archive, available at http://www.zib.de/zuse/, for having made available the work by Zuse.

[28] It is maybe also interesting to note that Zuse, given his familiarity with [AH28], considered the Entscheidungsproblem in the context of his Dyadik: "*So können die Normalformen dazu dienen, äusserlich verschiedenen Schaltungen miteinander zu vergleiichen, indem beide auf die Normalform gebracht werden. [...] Das ist nicht immer einfach, und in der theoretischen Behandlung von Schaltungen werden wir später auf ähnliche Probleme Stossen, wie sich in der formalen Logik unter den Namen "Funktionenkalkül", "Prädikatenkalkül", "Entscheidungsproblem" usw. bekannt sind.*" ([Zus37], p. 11). He did not work this out in any detail however.

Indeed, given his Dyadik, it was very easy to make the switch from, e.g., electromechanical machines to electronic machines. Using this abstract scheme, the design of Z2 could easily be translated to an electric version. In 1937, Zuse began to study electronic circuits, using vacuum tubes, together with Schreyer. Although both Zuse and Schreyer had seen how much speed they could obtain with electronic devices, they had to give up on the idea ([Zus80], p. 619):

> During the war we submitted the concept of an electronic computer with 2000 tubes to the German Government Research authorities, but their reaction was negative. We would never have attempted to construct a computer with 18000 tubes and I admire the heroism shown by Eckert and Mauchly.

In 1939, the relay-computer Z2 was almost finished when Zuse had to join the army. In 1940 he worked for *Henschel Flugzeugwerke* and could finish his Z2 over the weekends [Bau80]. Zuse gave a demonstration of the Z2 for the *Deutsche Versuchsanstalt für Luftfahrt* (DVL) and they gave him approval to continue work on the Z3, which was completed in 1941. Although not a stored program computer, it has been proven by Rojas [Roj98] that the Z3 can in fact be considered *in principle* as a universal computer, i.e., in the way present-day computers can be considered universal.[29] After the Z3, Zuse also build the Z4, which was the only machine that survived the war.

For Zuse, his Dyadik were the "*die theoretische[n] Grundlage[n] [für die] mechanische[n] Durchführung von Rechnungen*" ([Zus37], p. 3), i.e., the theoretical foundation for the mechanical execution of computations. Computations were, in his mind, not solely restricted to calculation with numbers. He understood it as something far more general, Zuse's understanding here being very close to Turing's in the context of computing machines, as discussed above ([Zus37], p. 1):

> In the following we want to develop a theory to mechanically solve schematic thinking tasks. As schematic thinking operations we see all formulae, derivations, algorithms etc. for which specific resulting informations (*Angaben*) can

[29] For more detailed information on Zuse's Z-machines, the reader is referred to [Roj00].

> be derived from output states for all considered cases after a clear rule. Calculating with numbers belongs to the lowest level; the process of calculating is so schematic and clear that mechanical solutions are already applied to a considerable extent. [...] but we want to note already now, that calculating with numbers is only a special domain within general calculation. The question what other domains of logic and their applications can be developed, will be investigated when we can oversee more clearly the theory to be constructed here in all its possibilities. *Under "calculation" we understand: Form informations out of given informations after given rules.* Thus, we first have the concept information. These can have very different meanings, e.g., numbers, statements, names, codes, military degrees, data, commands, messages, deductions etc.[30]

As is clear from this quote, Zuse had a very broad, "general-purpose", conception of computability, and identifies it with processes that produce certain informations (*Angaben*), from other informations, where these informations can, in a way, almost be anything. Given this generalized interpretation of computability, it only takes a small step to connect computability with human thinking. This is indeed the step Zuse made ([Zus80], p. 614):

> General considerations concerning the relations between calculating and thinking followed. I realized that there is no border line between these two aspects and

[30] "Im folgenden soll eine Theorie entwickelt werden, um schematische Denkaufgaben mechanisch zu lösen. Als schematische Denkoperationen gelten alle die Formeln, Ableitungen, Algoritmen und dergl., bei denen für alle in Frage kommenden Fälle nach einer klaren Vorschrift ausgegebenen Ausgangszustände bestimmte Resultatangaben abgeleitet werden. Zur untersten Stufe gehört das Rechnen mit Zahlen; hier ist der Rechnungsgang so schematisch und klar, dass mechanische Lösungen bereits in Großem Umfang angewandt werden.[...] jedoch wollen wir jetzt schon beachten, dass das Zahlenrechnen nur eine Spezialgebiet des allgemeinen Rechnens ist. Die Frage, welche anderen Gebiete der Logik und ihrer Anwendungen sich durch "Rechnen" erschliessen lassen, wollen wir erst untersuchen, wenn die hier aufzustellende Lehre in ihren Möglichkeiten klarer zu überblicken ist. *Unter "Rechnen" willen wir also verstehen: Aus gegebenen Angaben nach einer Rechenvorschrift neue Angaben zu bilden.* Wir haben also zunächst den Begriff der Angaben. Diese können sehr verschiedene Bedeutung haben, z.B; Zahlen, Aussagen, Namen, Kennziffern, Dienstgrade, Daten, Befehle, Nachrichten, Schlussfolgerungen u.s.w." This translation as well as the following translations from Zuse are due to Maarten Bullynck. Remark that the second sentence is grammatically ambiguous if not unclear in the original and in the translation.

> by 1938 it was already perfectly clear to me that the development would progress in the direction of the artificial brain. At that time I knew scarcely anything about the working method of the human brain. [...] I took these ideas very seriously and this may have influenced my whole philosophy of the further development. At that time there was practically nobody to discuss with me the consequences of the possible innovations following this line. Even ten years later when – after the war – I became acquainted with the pioneer work on the other side of the Atlantic I sometimes had the impression that they were playing with computers like children play with matches without overlooking the whole scope of the new field.

It is very remarkable, that Zuse very clearly understood that any of these "informations", and the processes that manipulate them to produce other "informations", could be encoded in the binary system, i.e., he knew that instructions could be encoded as numbers. This idea is basic to the stored program idea. Zuse indeed considered this possibility, but, as was said, he never implemented it in one of his Z1–4-machines. To implement both instructions and data in the same hardware location, was understood as closing a contract with the devil, and Zuse thus searched for other solutions to be able to build the kind of machines he envisioned ([Zus80], p. 616):

> The idea of general calculating or information processing, as we say today, induced me to consider that the program, too, is information and can be processed by itself or by another program. This general concept was elaborated in all consequences in the Plankalkül. In hardware it means that we not only have a controlling line going from left to right, but also from right to left. I had the feeling that this line could influence the whole computer development in a very efficient but also very dangerous way. Setting up this connection could mean making a contract with the devil. Therefore, I hesitated to do so, being unable to overlook all the consequences, the good as well as the bad. So first I concentrated on theory. This led to the Plankalkül. [...] My colleagues on the other side had no scruples about the problem I just mentioned. John von Neumann and others constructed a machine with a storage for all kinds of information including the program. This idea may have been trivial, as soon as the programs were binary coded and there

> existed storage units for storing any binary coded information. This requirement was already fulfilled by the machines Z1 to Z4 and others. Besides this, the idea of storing the program was already mentioned for instance in one of my patent applications in 1936. Other pioneers may have had the same idea rather early. I think it was the special organization of the machine of John von Neumann which opened the door for universal calculating. He gave the signal "all clear" for the scientists but for the devil, too. [...] My own design for future machines on paper were more structured with instructions stored independently and special units for the handling of addresses and subroutines nested in several levels.

For Zuse the idea of stored program was not something he wanted to implement in hardware, he did not want to locate instructions and data in the same storage, and he indeed searched for another solution resulting in his Plänkalkul. In the meantime he had also designed a machine that was never build, the Z394, which was based on the logical operations, AND, OR and NOT [Zus44]. This machine can be considered as the hardware for Plankalkül. As is noted by Bauer, he hoped in this way to build a "*Planfertigungsgerät*" – a device that prepares programs, a separate computer that could e.g. be connected to the Z4. Instead of storing instructions and data in the same storage, this *Planfertigungsgerät* could work as a kind of compiler in which programs could be processed by themselves. In this way, Zuse conceived of a special hardware unit to prepare a program, that was connected to the part of the computer that executes the purely numerical operations ([Zus80], p. 616–617):

> In our situation the only realistic way to process a program by itself was to build a separate computer for this purpose. Thus, the construction of the computers for numerical calculations could be continued without drastic modifications. We called this type of machine "Planfertigungsgerät", that means, a special computer to make the program for a numerical sequence controlled computer. This device was intended to do about the same sophisticated compilers do today. But in 1945 we had to stop this interesting development.

Although Zuse did not want to use hardware stored programs he did know how to encode programs as numbers, and he did understand the "computational" powers of logic. In fact, in his patent description of the Z394 he explicitly stated that he considered logic to be capable to compute anything computable he considered as computable. And we already know how general his intuitive conception of computability was ([Zus44] pp. 2–3):

> The inventor has recognized that all calculation problems can be resolved into the elementary operations of theoretical logic. These calculation problems do not only consider calculation with numbers, but more generally also calculation with states, occurrences and conditions. In the context of this invention, we understand under "calculation" the derivation of resulting informations from arbitrary informations after a rule. The invention wants to build a calculation instrument, that fulfills the formalism of propositional logic, with this instrument one can execute autonomically all calculation processes after the *definition* [m.i.] given supra, i.e., not only number calculations.[31]

Zuse considered this machine as a "*Logistische Rechenmaschine*" [Zus45a], a logical machine, contrary to what he called his "algebraic machines", Z1, Z2, Z3 and Z4. We think it reasonable to regard Zuse's conception of this machine as a universal machine, in the more intuitive sense of the word. For him, this machine should be able to solve any general combinatorial problem or other problems ([Zus45a], p. 9):

> The result of these developments will be the general calculation machine, that solves general combinatorial problems and mechanical thinking problems on

[31] "Der Erfinder hat erkannt, dass alle Rechenaufgaben in die Grundoperationen der theoretischen Logik aufgelöst werden können. Diese Rechenaufgaben betreffen hier aber nicht nur ein Rechnen mit Zahlen, sondern darüber hinaus ganz allgemein auch ein Rechnen mit Zuständen, Begebenheiten und Bedingungen. Im Rahmen dieser Erfindung wird also unter "Rechnen" das Ableiten von Resultatangaben aus irgendwelchen Angaben nach einer Vorschrift verstanden. Die Erfindung stellt sich die Aufgabe eine Rechenverrichtung zu bauen, die den Formalismus des Aussagenkalküls der theoretischen Logik genügt, mit dieser Vorrichtung kann man dann alle Rechenvorgänge gemäss obiger *Definition* [m.i.], also nicht nur alle Zahlenrechnungen, entsprechend, selbstätig durchführen." The translation is due to Maarten Bullynck.

> the basis of applied logistic. I call this development the “logistic calculation machine”.[32]

As was said, this logistic machine can be considered as the hardware counterpart of Plankalkül. Plankalkül can be regarded as one of the first programming languages ever, and influenced Bauer and Rutishauer for their contributions to the development of ALGOL.[33] Scrolling through the on-line Zuse archive (http://www.zib.de/zuse/) one sees that already in 1941 there are notes on Plankalkül. A hand-written text denoted by the editors of the digital archive as the *Urschrift des Plankalküls* [Zus45b] is dated 1945, and is basically the written version of [Zus72]. The title of this *Urschrift* however is not Plankalkül, but *Theorie der angewandten Logistik*, “theory of applied logistic”, with the -ik in Logistik underlined.
Although we will not discuss this language in detail here,[34] it is important to mention some of its possibilities, summarized in the following quote ([Rojnd], p. 1):

> The Plankalkül was the software counterpart of the logistic machine. Complex structures could be built from elementary ones, the simplest being a single bit. Also, sequences of instructions could be grouped into subroutines and functions, so that the user had only to deal with a very abstract instruction set that masked the complexity of the underlying hardware. The Plankalkül exploited the concept of modularity, so important today in computer science, almost in an extremist way: several layers of software make the hardware transparent for the programmer. The hardware itself is able only to execute the absolutely minimal instruction set.

[32] “Das Ergebnis dieser Entwicklungen wird die allgemeine Rechenmaschine sein, die auf der Grundlage angewandter Logistik allgemeine kombinatorische Probleme und mechanische Denkaufgaben löst. Ich nenne diese Entwicklung die “Logistische Rechenmaschine”.”

[33] The paper by Bauer [Bau80] discusses these matters. Rutishauer knew Z4, and programmed it, once it was moved to the Institute for Applied Mathematics of ETH (Swiss Federal Polytechnic Institute).

[34] For a detailed discussion on Plankalkül the reader is referred to [BW72].

The basic principle of the Plankalül is indeed the bit,[35] a feature that allows for the hardware to be very simple. In his [Dav01b] Martin Davis has pointed out that the philosophy behind Turing's ACE had been to keep the hardware as simple as possible. As should be clear, Zuse's general philosophy is in this respect very similar to (and predating) Turing's with respect to computing machines, even if Zuse was an engineer and did not know about the universal Turing machine at that time. There are more similarities to be found between Zuse and Turing. For example, Zuse had a very broad conception of what should be considered computable, emphasizing that computing is not restricted to calculations with numbers. In this respect, it is interesting to note that already in 1941 one finds notes for developing a "Schachprogramm", a chess program, as a preliminary investigation for Plankalkül.[36] Later, Zuse describes that he wanted to investigate the efficiency and generality of the Plankalkül by applying it to chess problems ([Zus80], p. 623):

> It was interesting for me to test the efficiency and the general scope of the Plankalkül by applying it to chess problems. I learned to play chess especially for this purpose. This field seemed to me suited for the formulation of rather sophisticated data structures, nested conditions, and general calculations.

For Zuse, his Plankalkül is a real "universal" language, as he would later point out, in the sense that he considered it capable to compute anything he considered computable. For him, computability was not restricted to numerical calculations, but could be generalized even to certain thinking processes. It is

[35] This was rather important for Zuse, especially given the fact that in contemporary languages, the bit is often only tolerated as a Boolean object for controlling conditional branching: "*The first principle of the Plankalkül is: data processing begins with the bit. [...] Since about 20 to 30 years the priority of numerical calculation has only slowly been overcome; and in this time, in conventional computers the bit has been tolerated only as a Boolean object for controlling conditional branching and so on. In contrast, the Plankalkül is fundamentally based on the bit. To express logical relations I used the notation and results of the propositional and the predicate calculus. Any arbitrary structure may be described in terms of bit strings; and by introducing the idea of levels we can have a systematic code for any structure, however complicated, and can identify any of its components.*" [Zus80], pp. 621–622.

[36] The reader is referred to the digital Zuse archive http://www.zib.de/zuse/ to check this.

interesting to note that he effectively tested this generality of his language, by applying it to several different problems, like chess problems: ([Zus80], p. 625)

> Behind the Plankalkül there is a special philosophy based on my early conviction, that there is a steady way from simple numerical calculation to high-level thinking processes. In order to test the universality of this language I applied it for several unusual fields. Thus, for instance, I made some steps in the direction of symbolic calculations, general programs for relations, or graphs, as we call it today, chess playing, and so on.

Zuse did more than building computers and developing a programming language. He for example developed self-reproducing automata, independent of von Neumann and Ulam, but contrasted his with that of von Neumann by stating that for him this was an engineering problem.[37] We will not discuss these other research interests by Zuse here since they lie beyond the scope of our discussion here.

4.1.4 Conclusion.

The "behemoth" ENIAC, as Martin Davis has described it, was a machine invented and constructed by engineers. The basic advantage of the EDVAC design over ENIAC is that it is more general purpose and includes the idea of stored programs. It is far more close to a universal Turing machine than the behemoth.

Until now the answer to the question of who made what kind of contribution to the EDVAC design is still not completely clear. According to Mauchly and Eckert they made significant contributions to the design, including the stored program idea. So, is the EDVAC an engineer's or a logician's machine? To our mind, it is the result of the combination of both. On the one hand, we do not think that Eckert or Mauchly lied about the significance of their contributions. Why else would they have made such a thing about this whole issue? On the other hand,

[37] "*Another field of my research is "self-reproducing systems." But I see the problem not from a mathematical point of view, as, for instance, von Neumann did, but as an engineer.*" ([Zus80], p. 627)

the emphasis on the logical design of the machine and its link with McCullough and Pitts's "neural computers" – influenced by Turing's universal machine – is due to von Neumann. One might well ask whether the general-purpose digital stored program computer would have resulted without von Neumann's contributions. Of course, we cannot answer this question in any definite way. More important, here is the fact that several very similar solutions to problems were being developed by people who were less familiar with logic than von Neumann and Turing, solutions which clearly have their formal counterpart, like the idea of encoding instructions as numbers such that they can be manipulated as numbers.

That the first such solutions were "behemoths", and still far removed from the more elegant and general design of an ACE or and EDVAC, is to our mind only normal. Is it not a typical feature of many scientific and technological innovations that the original form of a solution is very far removed from the elegant or efficient form it evolves in through the years? Or, to put this slightly rhetorically, who would object that the universal Turing machine, as originally described by Turing, is not a behemoth in its own genre, with its difficult if not obscure description, containing several mistakes? It is true that the theoretical ideas underlying the original universal Turing machine are still basically the same as any theoretical universal machine constructed today, while the ENIAC was quite different from the EDVAC. After it was rewired however, it was still a behemoth, but far more general-purpose than it was originally. The point is that if one does not already have a theory that can be useful to, e.g., build a computer, one can only learn from the problems that arise in developing a behemoth without theory. Luckily there were people like von Neumann and Turing who knew about the theoretical counterpart of the computer, and were thus able to very quickly propose their "logical engines", much more in the spirit of a (theoretical) universal Turing machine. In this sense, their work – especially the EDVAC design since it was more well-known – has most probably accelerated the technology and has contributed significantly to the form of present-day computers.

A completely different story is given by Zuse, one that to our mind throws a refreshing light on this whole engineers vs. logicians controversy. Zuse was an engineer who started to build computers because he wanted to automate

the monotonous calculations he had to perform as an engineer. Very soon, he developed a theoretical foundation for designing these machines. After having read Hilbert he was able to put this foundation in a more logical form. Logic did play a very important role in Zuse's thinking, but he was not very familiar with the subject and did not know the results by Turing. One can thus conclude that Zuse developed his own kind of logical system because he needed one, quite unaware of many of the developments that would (of course) have contributed significantly to his work, if he had known them.
To our mind, Zuse's work beautifully shows how an engineer's work became more and more connected with logic, in having thought and worked for several years on computing machines. The first of these machines were more special-purpose and intended to merely perform certain calculations. But Zuse soon realized that such machines might also be used to solve far more general problems. This idea was probably more exactly formulated, once he had abstracted from the engineering details and developed an abstract calculus to make more easy the design of circuits. From that point on, Zuse becomes very explicit about the possibilities of computing machines. In this sense, we do not think that Zuse's understanding of the possibilities of computing machines was less general-purpose than e.g. Turing's, as is clear from some of the quotes given and the fact that he considered chess problems as a test case for his constructions. One could very well say that Zuse very quickly formulated a kind of thesis, identifying a very general all-purpose concept of computability with the kind of logistic and algebraic machines he envisioned, his Plankalkül being the final form of his thinking in this direction.

So, was logic a fundamental prerequisite to develop digital general-purpose stored program computers? To our mind, the answer to this question can only be affirmative. However, as the case of Zuse shows, this does not mean that such "logistic" ideas, i.e. logic turned into a technical instrument, could not arise in a context quite independent from the development of mathematical logic before the war. Indeed, in Zuse's case, his logistic machines cannot be seen independent from what he had learned from his first experiences with computing machines. The more logical ideas only followed in having had the time to think about these machines. A kind of reversed conclusion is, to our mind, valid for Turing's work. Turing already knew the formal equivalent of a computer and this must have heavily influenced his design of the ACE. Thus, in 1936 he did have a theory for computing machines, but not the knowledge for really building one. Turing always had a keen interest in real machines, an interest he could turn into reality after having built up his knowledge of the more technical aspects of building real machines, during his time as a cryptanalyst at Bletchley Park. Given his theoretical knowledge combined with the more practical knowledge he could then design a kind of practical version of his theoretical universal machine.

To return to the quote by Ulam from the beginning of this section: the computer is a marvellous machine, developed after one of the most ugliest periods of human history, as a result of the confluence of engineering ideas with logic. This is exactly what makes the computer such an interesting device from the point of view of this dissertation: it is a physical engineered thing that embodies the theoretical ideas developed by Church, Gödel, Post, Turing, Kleene et al, a fact quite independent of the question of how much "logic", *as it already existed at that time*, was necessary for the development of the computer. Regarded as the physical form of the intuitive notion of computability, one that can compute far more quickly than we humans, this device becomes the more interesting if one uses it to disclose the "discourse" of computability. This was exactly one of the things it was used for, from its early years onwards.

4.2 Exploring the "universe of discourse": heuristic methods and computer experiments.

> If mathematics describes an objective world just like physics, there is no reason why inductive methods should not be applied in mathematics just the same as in physics. The fact is that in mathematics we still have the same attitude today that in former times one had towards all science, namely we try to derive everything by cogent proofs from the definitions (that is, in ontological terminology, from the essence of things). Perhaps this method, if it claims monopoly, is as wrong in

> mathematics as it was in physics.
>
> Kurt Gödel, 1951.[38]

> It will seem not a little paradoxical to ascribe a great importance to observations even in that part of the mathematical sciences which is usually called Pure Mathematics, since the current opinion is that observations are restricted to physical objects that make impression on the senses. As we must refer the numbers to the pure intellect alone, we can hardly understand how observations and quasi-experiments can be of use in investigating the nature of the numbers. Yet, in fact, as I shall show here with very good reasons, the properties of the numbers known today have been mostly discovered by observation, and discovered long before their truth has been confirmed by rigid demonstrations. There are even many properties of the numbers with which we are well acquainted, but which we are yet able to prove; only observations have led us to their knowledge. Hence we see that in the theory of numbers, which is still very imperfect, we can place our highest hopes in observations; they will lead us continually to new properties which we shall endeavor to prove afterwards. The kind of knowledge which is supported only by observations and is not yet proved must be carefully distinguished from the truth; it is gained by induction, as we usually say. Yet we have seen cases in which mere induction led to error. Therefore, we should take great care not to accept as true such properties of the numbers which we have discovered by observation and which are supported by induction alone. Indeed, we should use such a discovery as an opportunity to investigate more exactly the properties discovered and to prove or disprove them; in both cases we may learn something useful.
>
> Leonhard Euler, 1761.[39]

From its early use on, the computer has been used as a means to study and solve a rich variety of different problems and questions, of which some are “pure” mathematical problems, others are related to other domains like physics and even biology. Given its high speed it can be used to make calculations to tackle problems in a way that was hardly within human reach before. However, it was soon understood that “pure” calculations are not the sole domain of the computer. We already know that both Turing and Zuse regarded the computer as

[38] From [Göd51], p. 313

[39] From [Eul61], translated and quoted in [Gre82], p. 4

something that might be used to study human reasoning, the idea of playing chess being one typical example of how one could start to deal with the question in how far computers can be considered "intelligent".
From the introduction of a volume of the *Annals of Mathematics Studies* from 1956, called *Automata Studies*, it is clear that the idea of linking the computer with questions concerning the functioning of the human brain very soon became a "fashionable" topic: ([MS56b], p. v):

> Among the most challenging scientific questions of our time are the corresponding analytic and synthetic problems: how does the brain function? Can we design a machine that will simulate a brain? Speculation on these problems, which can be traced back many centuries, usually reflects in any period the characteristics of machines then in use. Descartes, in DeHomine, sees the lower animals and, in many of his functions, man as automata. Using analogies drawn from waterclocks, fountain and mechanical devices common to the seventeenth century, he imagined that the nerves transmitted signals by tiny mechanical motions. Early in the present century, when the automatic telephone system was introduced, the nervous system was often linked to a vast telephone exchange with automatic switching equipment directing the flow of sensory and motor data. Currently it is fashionable to compare the brain with large scale electronic computing machines.

From this context of researching the idea of "intelligent machinery", several different branches and theoretical frameworks have arisen. Automated theorem proving, artificial neural networks, self-reproducing automata,...are different developments tackling different aspects of the self-same question: in how far can computers be considered capable to perform certain tasks a human can perform, and vice versa.[40]
Although these are very interesting developments, they lie beyond the scope of this research. Of more significance here is the use of the computer as a pure powerful computing machine. We will mainly focus on some of the work and remarks made by two computer pioneers in this context: John von Neumann

[40] In [Dav01a] a historical survey is given of automated reasoning, by one of the pioneers of this branch of computer science, i.e., Martin Davis.

and Derrick H. Lehmer. It is not the purpose of this section to be complete. Rather we want to give an impression of how it was very clearly understood, by some of the first computer users, that the mere computing power of computers makes it possible to enclose what Lehmer has called the "universe of discourse" and how this possibility has led to more heuristic research within the context of mathematics as well as new results. We first planned to add a small subsection on Turing's work in this context, but finally decided to merely mention it here in the introduction. Turing used the Mark I for research on the distribution of the zeros of the Riemann zeta-function [Tur53] on the Manchester Mark I, and emphasized the significance of mathematical rigour in this context, even if one is working with "mere" computations. Furthermore, he used the computer for studying certain problems connected to his work on morphogenesis and did some numerical simulations of non-linear equations, that are now studied in the context of chaos theory.

4.2.1 von Neumann and theoretical physics.

In the previous section we already mentioned that according to Ulam, von Neumann got interested in computers, due to the realization that some of the usual methods of mathematics fell short to study certain problems in theoretical physics. Also Burks has pointed out von Neumann's interest in computers as a way to obtain certain information about the solutions to non-linear partial differential equations underlying certain physical phenomena, like turbulence ([Bur66] pp. 2–3):

> von Neumann became especially interested in hydrodynamical turbulence and the interaction of shock waves. He soon found that existing analytical methods were inadequate for obtaining even qualitative information about the solutions of non-linear partial differential equations in fluid dynamics. Moreover, this was so, of non-linear partial differential equations in general. von Neumann's response to this situation was to do computing. During the war he found computing necessary to obtain certain answers to problems in other fields, including nuclear technology.[...] The procedure he pioneered and promoted is to employ computers to solve crucial cases numerically and to use the results as a heuris-

> tic guide to theorizing. von Neumann believed experimentation and computing to have shown that there are physical and mathematical regularities in the phenomena of fluid dynamics and important statistical properties of families of solutions of the non-linear partial differential equations involved. [...] From the special cases one could get a feeling for such phenomena as turbulence and shock waves, and with this qualitative orientation could pick out further critical cases to solve numerically, eventually developing a satisfactory theory.

As is clear from this quote, von Neumann wanted to use the results generated through the computer as a "heuristic guide"for further theorizing. To give an example, that is not immediately connected to theoretical physics, von Neumann expressed an interest in using the ENIAC to compute the value of π and e to many decimal places in order to get an idea about the statistical distribution of these two numbers.[41] The computations for e were finished in July 1949, those for π during Labor-Day weekend, in September 1949.[42] In [Rei50] and [MRvN50] set-up and results were discussed: the first 2000 decimal digits of both numbers were computed. A statistical analysis of the data led to the conclusion that "*the material has failed to disclose any significant deviations from randomness for π, but is has indicated quite serious ones for e.*" ([MRvN50], p. 109).[43] von Neumann's interest in the subject of randomness and, more generally, Monte Carlo methods, was triggered by its usefulness in the context of

[41] "*Early in June, 1949, Professor John von Neumann expressed an interest in the possibility that the ENIAC might sometime be employed to determine the value of π and e to many decimal places with a view toward obtaining a statistical measure of the randomness of distribution of the digits [...]*" ([Rei50], p. 11)

[42] As was the case for many computations done on the ENIAC, these were all done outside the "official time", during holidays. As Reitwiesner [Rei50] explains, four members of the ENIAC staff and Reitwiesner himself did 8-hours shifts to keep the ENIAC operating continuously throughout the Labor-day weekend.

[43] It is interesting to point out that part of the research on the random character of the digits in π is still situated in a more heuristic research context. Recently, an important paper was published in the journal *Experimental Mathematics* on this topic [BC01], in which it is shown that the statistical randomness of several constants, including π, depends on an hypothesis concerning the distribution of the iterates of certain dynamical maps, and is thus situated in a branch of mathematics, characterized by the numerous computer experiments underlying it, i.e., chaos theory.

nuclear physics, i.e., in the context of developing the H-bomb. It was Ulam who came up with the idea of Monte Carlo methods and its possible use in this context and communicated it to von Neumann. In fact the name "Monte Carlo" goes back to a story about Ulam's uncle, who would borrow money from relatives because "he just had to go to Monte Carlo" [Met87]. As Ulam recounts (Remark dated 1983 by Ulam, quoted in [Eck87], p. 131):

> The first thoughts and attempts I made to practice [the Monte Carlo Method] were suggested by a question which occurred to me in 1946 as I was convalescing from an illness and playing solitaires. The question was what are the chances that a Canfield solitaire laid out with 52 cards will come out successfully? After spending a lot of time trying to estimate them by pure combinatorial calculations, I wondered whether a more practical method than "abstract thinking" might not be to lay it out say one hundred times and simply observe and count the number of successful plays. This was already possible to envisage with the beginning of the new era of fast computers, and I immediately thought of problems of neutron diffusion and other questions of mathematical physics, and more generally how to change processes described by certain differential equations into an equivalent form interpretable as a succession of random operations. Later [in 1946, I] described the idea to John von Neumann and we began to plan actual calculations.

The Monte Carlo method is used as a way to explore the behaviour of various physical and mathematical systems, in order to make certain predictions. The basic idea is to use a randomly distributed sample, and look at what happens to the sample, or to make certain random decisions that determine the future behaviour of the sample. Metropolis, who worked together with von Neumann, Ulam et al, explained how the method was originally implemented, describing an example from von Neumann in a letter to Richtmyer (p. 127):

> Consider a spherical core of fissionable material surrounded by a shell of tamper material. Assume some initial distribution of neutrons in space and in velocity but ignore radiative and hydrodynamic effects. The idea is to now follow the development of a large number of individual neutron chains as a consequence of

> scattering, absorption, fission and escape. At each stage a sequence of decisions has to be made based on statistical probabilities appropriate to the physical and geometric factors. The first two decisions occur at time $t = 0$, when a neutron is selected to have a certain velocity and a certain spatial position. The next decisions are the position of the first collision and the nature of that collision. If it is determined that a fission occurs, the number of emerging neutrons must be decided upon, and each of these neutrons is eventually followed in the same fashion as the first. If the collision is decreed to be a scattering, appropriate statistics are invoked to determine the new momentum of the neutron. When the neutron crosses a material boundary, the parameters and characteristics of the new medium are taken into account. Thus, a genealogical history of an individual neutron is developed. The process is repeated for other neutrons until a statistically valid picture is generated. [...] How are the various decisions made? To start with, the computer must have a source of uniformly distributed pseudo-random numbers.

As is clear, von Neumann's interest in the statistical distribution of π and e might not have been completely innocent: the use of a good random number generator on the ENIAC must have been basic for the results from the computer experiments to be reliable. One of the random generators known then was von Neumann's "middle-square digits" method. For this method, an arbitrary n-digit is squared creating a 2n-digit product. A new integer is generated by extracting the middle n-digits from the product. This method is known to be a rather bad random number generator, though selecting certain 'runs' and imposing restrictions on the generated sequences, one could attain more or less appropriate random sequences. One can only speculate how important it has been for the computations done on the ENIAC in the context of nuclear physics.[44]

von Neumann understood very well how the computer could be used to in-

[44] See e.g. Metropolis's paper [Met87]. For more information on the history of the Monte Carlo method, the reader is referred to the special issue of *Los Alamos Science*, nr. 15, 1987, available on-line at http://www.fas.org/sgp/othergov/doe/lanl/pubs/number15.htm. In [Bul07], one also finds a discussion of the Monte Carlo method in the context of a study of the first computations done on the ENIAC.

vestigate the behaviour of certain physical systems, by computing numerically approximate solutions to the non-linear partial differential equations underlying them. Monte Carlo methods are then indeed invaluable instruments in this context. In one of the lectures delivered at the University of Illinois in 1949, he made explicit how the computer can be used not only in the context of physics but also in mathematics ([vN66], pp. 33–35):

> In pure mathematics the really powerful methods are only effective when one already has some intuitive connection with the subject, when one already has, before a proof has been carried out, some intuitive insight, some expectation which, in a majority of cases, proves to be right. In this case one is already ahead of the game and suspects the direction in which the result lies. A very great difficulty in any new kind of mathematics is that there is a vicious circle: you are at a terrible disadvantage in applying the proper pure mathematical methods unless you already have a reasonably intuitive heuristic relation to the subject and unless you have had some substantive mathematical successes in it already [...] progress has an autocatalytic feature. Almost all of the correct mathematical surmises in [the area of the non-linear sciences] have come in a very hybrid manner from experimentation. If one could calculate solutions in certain critical situations [...] one would probably get much better heuristic ideas. [...] there are large areas in pure mathematics where we are blocked by a peculiar inter-relation of rigor and intuitive insight, each of which is needed for the other, and where the unmathematical process of experimentation with physical problems has produced almost the only progress which has been made. Computing, which is not too mathematical either in the traditional sense but is still closer to the central area of mathematics than this sort of experimentation is, might be a more flexible and more adequate tool in these areas than experimentation.

To von Neumann, the computer was very clearly a means to "de-block" certain areas of mathematics for further exploration, allowing to build up an intuition of a certain problem. Indeed, although the task the computer has to perform in this context seems quite inessential, it "merely" computes quicker than we can, it has been basic to several branches of science exactly in this respect. To give just one example, the area of fractal geometry and chaos theory would probably have remained "blocked" were it not for the computer. The role "computer experiments" have played in this domain can hardly be overestimated, i.e., most results go back to such "experiments". As we will see in part II, it has been this kind of approach that has been invaluable for building up our intuition of tag systems.

For von Neumann, the possibility of using computers to build up heuristic knowledge useful for solving certain mathematical problems, must have been a fascinating development because it offered the possibility of connecting hard theoretical mathematical problems with more down-to-earth problems. For von Neumann this link between "pure" mathematics and more concrete, "empirical" ideas was very important, if not basic. We would like to end this section on von Neumann with the conclusion from *The Mathematician*, where he very much opposes the idea of mathematics becoming too much 'l'art pour l'art' ([vN47], p. 378):

> I think that it is a relatively good approximation to truth – which is much too complicated to allow anything but approximations – that mathematical ideas originate in empirics, although the genealogy is sometimes long and obscure. But, once they are so conceived, the subject begins to live a peculiar life of its own and is better compared to a creative one, governed by almost entirely aesthetical motivations, than to anything else and, in particular, to an empirical science. There is, however, a further point which, I believe, needs more stressing. As a mathematical discipline travels far from its empirical source, or still more, if it is a second and third generation only indirectly inspired by ideas coming from "reality", it is beset with very grave dangers. It becomes more and more purely aestheticizing, more and more purely *l'art pour l'art*. This need not be bad, if the field is surrounded by correlated subjects, which still have closer empirical connections, or if the discipline is under the influence of men with an exceptionally well-developed taste. But there is a grave danger that the subject will develop along the line of least resistance, that the stream, so far from its source, will separate into a multitude of insignificant branches, and that the discipline will become a disorganized mass of details and complexities. In other words, at a great distance from its empirical source, or after much "abstract" inbreeding,

> a mathematical subject is in danger of degeneration. At the inception the style is usually classical; when it shows signs of becoming baroque, then the danger signal is up. It would be easy to give examples, to trace specific evolutions into the baroque and the very high baroque, but this, again, would be too technical. In any event, whenever this stage is reached, the only remedy seems to me to be the rejuvenating return to the source: the reinjection of more or less directly empirical ideas. I am convinced that this was a necessary condition to conserve the freshness and the vitality of the subject and that this will remain equally true in the future.

von Neumann died in 1957 of cancer, possibly caused by exposure to radiation during (one of) the A-bomb tests at the Bikini islands.

4.2.2 Lehmer's computational work on number theory.

Contrary to von Neumann, Derrick Henry Lehmer had a clear background in computing machinery when he, together with his wife Emma, implemented his first ENIAC program. He had already build several special-purpose devices, i.e., prime sieves.[45] Lehmer was a number theorist who, very explicitly, considered mathematics and more specifically number theory, as an experimental science, a lesson he learned from his father, D.N. Lehmer, also a number theorist. It is thus not surprising that he very quickly understood the possibilities of using computers for making progress in number theory ([Leh74], p. 3):

> My father did many things to make me realize at an early age that mathematics, and especially number theory, is an experimental science. [...] Exploring in discrete variable mathematics is generally simpler than in continuum mathematics. One can see the input and the resulting experimental output with absolute clarity. For the same reason a digital or discrete variable computer is a better aid to discovery than an analog machine. This advantage is due to the enormous flexibility possessed by digital computers. Exploits such as moon missions would be utterly impossible without discrete variable techniques, despite the continuity of space. We should regard the digital computer system as an instrument to assist

[45] A detailed account on Lehmer's sieves can be found in [Leh80].

> the exploratory mind of the number theorist in investigating the global and local properties of this universe, the natural numbers and their algebraic expansions. The role of this system in such an investigation can be of varying importance, ranging from the production of a single counterexample, to the organization of data to suggest ideas, through the search for patterns in data, to the ultimate role of proving theorems on its own.

Lehmer has always been very explicit of how he understood the role of the computer in the context of mathematical research. For him, computers can be used to assist man in studying the "universe" of the natural numbers. In fact, they give us direct access to this universe, and allow us to explore parts of the theory of numbers inaccessible before. Although Lehmer understood number theory as an experimental science and the computer as an instrument that can assist us in our research of this science, it is clear that the computer's role is not restricted to the one emphasized by von Neumann. It *can* help us to build up an intuition of a given problem. Furthermore, by using it to generate mathematical tables, it can produce counterexamples.[46] However, to Lehmer, the computer can also be used to generate genuine proofs. We already mentioned the newly arising domain of automated theorem proving in the fifties. The theorems Lehmer was able to prove, however, were at that time not considered as real machine proofs in the sense of the theorem proving programs then developed. As is explained by Lehmer ([LLMS62], pp. 407–408):

> The referee comments that the proof of theorem 1 [...] is "not a machine proof in the sense of the theorem-proving programs now being developed." This is true. The aim of most writers on this subject is to consider a very general program enabling a digital computer to prove a wide class of theorems at a very low level, beginning with the axioms, setting its own goals, and trying to achieve them without human intervention. This is, in a way, a simulation problem. Speculations about such programs involve (significantly) such notions as decidability. Meanwhile, no really new theorems seem to emerge. Perhaps too much is expected of a single program. In our work, instead of starting with axioms, we did

[46] This is basically what Turing had in mind with the set-up of an experiment described in [Tur53]: he hoped to find with the help of the machine a 0 of the critical line for a given interval.

> not hesitate to use any device or previously known result that might be useful. In particular, the authors aided and abetted the machine in its search for a theorem and its proof. Nevertheless, all three results [...] are due to the machine. Even the verification of these results using the data supplied by the machine would be far too long and hazardous a calculation to do by hand.

At that time, the domain of automated theorem-proving had indeed not been able to prove any really new theorems.[47] Of course, Lehmer did not have the same research goals as the pioneers of automated theorem proving: he was not interested in studying human deductive processes, but wanted to study number theory and obtain new theorems. In this sense, it was basic that the process for finding a proof with the help of a computer, was far more interactive, combining both human and machine knowledge, *during the process itself*, depending on intermediary output and input.

So what kind of theorems were proven by the computer in this way? We will mention two such theorems, which Lehmer considered as genuine machine proofs, in the sense just described.

Theorem 1 *Every set of 7 consecutive integers greater than 36 contains a multiple of a prime* ≥ 43

Before stating the second theorem, it is important to explain the notion of cubic residues. Let $p = 3n + 1$ be a prime, then if $1^3, 2^3, 3^3, ..., (p-1)^3$ are reduced modulo p, only n remainders will be distinct. These are the cubic residues of p.[48]

Theorem 2 *All primes, except*

$$7, 13, 19, 31, 37, 43, 61, 67, 79, 127, 283$$

[47] As was pointed out by Martin Davis, in having implemented a decision procedure for Presburger arithmetic on the IAS computer, the first example of an algorithm implemented in the context of automated theorem proving, "*[i]ts great triumph was to prove that the sum of two even numbers is even.*" (quoted in [Dav01a]). That nothing really stunning could be proved was due to the fact that the algorithm was more than exponentially slow on certain inputs. The most important contributions of early automated theorem proving, are most probably the development of some standard algorithms and techniques [Dav01a].

[48] For example, with $p = 13$, the cubic residues are: 1, 5, 8, 12.

have three consecutive cubic residues. The first triple occurs not later than

$$23532, 23533, 23534.$$

This result is the best possible because there are infinitely many primes for which no three residues less than 23534 are consecutive.

In [LLMS62], where Lehmer discusses the set-up and the proof of theorem 2, he explains that one of the reasons for considering the theorem as genuine is the fact that, although the proof involves only a finite number of steps, it is a statement about an infinite class.
The proofs *and* the exact statement of both theorems are impossible to establish without the help of the computer, or as Lehmer describes it, are humanly impractical. For the proof of the first theorem, the computer had to solve thousands of instances of the Pell equation:

$$x^2 - 2\Delta y^2 = 1$$

where Δ is a certain parameter. As is noted by Lehmer, there was one "small" difficulty with the proof. The Pell equation has for each $\Delta \neq 2$ infinitely many solutions. It is here that the human mathematician must supply an important lemma, in that only the 21 smallest solutions (x, y) for the equation have to be examined by the machine. We will skip the details of the lemma.
The proof of the second theorem is much more complicated, as noted by Lehmer. The outcome was at first very much in doubt because they did not know in advance whether the proof tree that had to be constructed would come to a halt. The limit of 55 "stories" for each branch of the proof tree was determined by the machine. Furthermore, the "world constant" 23532 , as Lehmer called it, was found by the machine, independent of the method of proof.
For Lehmer, the fact that the proofs and theorems are humanly infeasible is fundamental, because only then it becomes possible to prove really new theorems, theorems that could not have been proven or even been stated by a human being. In this respect, it is very important that the outcome has a certain sense of unpredictability. Indeed, in order to outclass the human being, it is important that the machine has to perform thousands of steps, that are different enough

from each other, such that the outcome cannot be predicted by human beings ([Leh63], pp. 141–142):

> I would like to speak briefly of some theorem proving programs that we have been running in which the human is completely outclassed in what, I think you will agree, are fair contests. Our aim was to prove mechanically some really new theorems of some interest to humans. The novelty of the theorems is guaranteed by the fact that the proofs are humanly impractical. [...] In casting about for genuine theorems the proofs of which will tax the powers of a human being, we want to exploit the speed of the machine. This means that the proof must involve many thousands of steps all sufficiently different so that the outcome cannot be forecast. We must also exploit those features of the logical system of the machine that permit it to supervise and organize its own program. We should make it proceed in an unpredictable way by laying its own track ahead of it like a caterpillar tractor. At the same time it should keep a record of where it has been, so that it can return at a previous point and branch out along another path whenever it decides that this is necessary. Humans find this kind of work difficult even when it occurs in only moderate amounts. Of course if the proof is to be too difficult for humans we cannot be sure in advance that the theorem is true or, if true, that even the machine can prove it.

The significance of the unpredictability of the outcome in using a computer, was also pointed out by Turing, during a radio discussion on the idea of intelligent machinery ([Tur52a] p. 19):

> Sometimes a computing machine does do something rather weird that we hadn't expected. In principle one could have predicted it, but in practice it's usually too much trouble. Obviously if one were to predict everything a computer was going to do one might just as well do without it.

Because of this unpredictability, and the thousands of steps and decisions made by the machine itself, we do not have access to all the details of the proof ([Leh63], p. 143):

> Of course no one has all the details. The machine was asked to make progress

> reports from time to time and studying these reports we can follow the proof in broad outline only.

In this respect, one might well wonder what would happen if some computer would prove a long-standing important conjecture, but nobody would really understand the proof. One is then faced with the situation that one knows that something is true, but without understanding why, a situation which is quite opposite to what has been basically Penrose's argument to show that human mathematical insight cannot be algorithmic, using Gödel's incompleteness theorem.

Besides proving theorems, Lehmer used the computer for producing mathematical tables. As is pointed out by Franz Alt [Alt72] a "computations committee" was assembled, including Lehmer, Haskell Curry (!), Leland B. Cunningham, and Alt for testing the ENIAC. Lehmer describes, amongst others, the following "test" program: ([Leh74], p. 4):

> The ENIAC, the first electronic computer, was to have been shut down from Wednesday night till the following Monday morning. Instead, this chunk of 111 hours of machine time was made available to me and my wife to keep the ENIAC warm and active. The problem we decided to run was the following: For each odd prime p there is a least positive integer $e = e(p)$ such that $2^e \equiv 1(\bmod p)$, sometimes called the *order* or *exponent* of 2 modulo p. The problem proposed to the ENIAC was to find those primes p for which $e(p) \leq 2000$ until Monday morning 8 o'clock. During the weekend the limit 2000 was reduced to 1000 and then to 300 in order to speed things up. By Monday, we had reached $p = 4538791$. This successful run had a number of consequences, even legal ones, which I shall not discuss.

Making mathematical tables has a long-standing tradition in mathematics. They were, e.g,. very important for the work of Gauss.[49] At that time, these tables were computed by human computers. With the availability of the computer, it has become possible to seriously enlarge several tables, and construct new ones. Vandiver very clearly summarized the significance of tables, and the role of computers in this context, as follows ([Van58], p. 459):

[49] See [Bul05].

> Examination of numerical tables tells the number theorist so often what *isn't* true. For example, before we were able to use extensive pertinent data recorded by computing machines, we had made conjectures as to the answers to a particular problem; but when more extensive computations were made, it turned out that a number of conjectures were inaccurate. So a lot of time had been lost in trying to prove or disprove the false conjectures. In addition, we hope, by study of tables, to observe patterns indicating the existence of certain theorems which may turn out to be of an entirely novel character. For example, it is fairly clear that Euler must have discovered the Law of Quadratic Reciprocity by a study of the results of his extensive computations. There seems to be nothing in the literature which could have suggested it to him. He did not prove the law. This was achieved by Gauss, and this was the beginning of the development of a far-reaching subject in number theory.

The journal, *Mathematical Tables and Other aids to Computation*, that was founded in 1943 by D.H. Lehmer and R.C. Archibald, and now goes under the name *Mathematics of Computation*, only further emphasizes the role mathematical tables have played in the domain of “computational mathematics”.

4.2.3 Conclusion

In this section, we have mainly focussed on the significance of the computer, from its early beginnings, for tackling certain more mathematical problems. von Neumann clearly understood how the computer could help us to build up an intuition of a certain problem. Due to its computational power, it can be used to “simulate” certain physical phenomena numerically, in a way hardly possible in physical experimentation, varying several parameters, such that we are able to draw certain heuristic conclusions that can help to advance more theoretical results. Contrary to Lehmer, however, von Neumann believed that the production of a huge amount of information is only useful for the machine itself, not for man ([vN66], pp. 38–39):

> [...] let me point out that we will probably not want to produce vast amounts of numerical material with computing machines, for example, enormous tables of

> functions. The reason for using fast computing machines is not that you want to produce a lot of information. After all, the mere fact that you want some information means that you somehow imagine that you can absorb it, and, therefore, wherever there may be bottlenecks in the automatic arrangement which produces and processes this information, there is a worse bottleneck at the human intellect into which the information ultimately seeps. The really difficult problems are of such a nature that the number of data which enter is quite small. All you may want to know is a few numbers, which give a rough curve, or one number. All you may want in fact is a "yes" or a "no," the answer as to whether something is or is not stable, or whether turbulence has or has not set in. The point is that you may not be able to get from an input of, say, 80 numbers to an output of 20 numbers without having, in the process, produced a few billion numbers in which nobody is interested.

Indeed, for von Neumann, producing large mathematical tables as output to be studied by man is no longer necessary, since the computer itself can process these tables, tables, that are too large to be of any interest to man. Von Neumann regarded the computer rather as a machine that provides man with certain answers that can then be used for further research. This opinion stands in contrast with Lehmer's understanding of the computer, who considered the production of mathematical tables as an important possibility of these new machines. For Lehmer the fact that the computer gives as access to information – even in the "raw" form of tables – that was not within human reach before, is very fundamental ([Leh51], p. 146):

> There is no doubt that these new machines are creating new service jobs for mathematicians, young and old. However, it seems to me, the most important influence of the machines on mathematics and mathematicians should lie in the opportunities that exist for applying the experimental method to mathematics. Much of modern mathematics is being developed in terms of what can be proved by general methods rather than in terms of what really exists in the universe of discourse. Many a young Ph.D. student in mathematics has written his dissertation about a class of objects without ever having seen one of the objects at close range. There exists a distinct possibility that the new machines will be used in

> some cases to explore the terrain that has been staked out so freely and that something worth proving will be discovered in the rapidly expanding universe of mathematics.

For Lehmer, the most significant influence of the computer is the fact that it gives us access to what he has called "the universe of discourse". The disclosure of this "universe", although accessible within certain limits before, has indeed made it possible to directly observe certain objects of mathematics, that could only be considered in a theoretical fashion before. One only has to think about the "disclosure" of the Mandelbrot set, and many other attractors, to understand the significance of "this disclosure. Looking at the words by von Neumann and Lehmer from our contemporary perspective, it is clear that both contain a certain truth. On the one hand, computers are used to produce vast amounts of data, that are then studied by man. In the meantime, we have developed several methods to "summarize" these amounts of data in a form that can be easily accessed by humans. The significance of several forms of visualizations, including graphs and tables, in this context can hardly be underestimated.[50] On the other hand, the computer is often only used to provide answers, without the necessity of producing vast amounts of information.

In part II, we will consider some examples of how the computer has been and can be used as an instrument making available the universe of discourse, in the context of computability and unsolvability. Our main focus there will be on tag systems. The production of vast amounts of data, has played a significant role in our research on tag systems. To mention the most important result in this context, it was only after having studied hundreds of periodic strings produced by several different tag systems, through the computer, that we were able to

[50] I have worked for some time on computer visualizations. I wanted to use them to study tag systems and, at that time, several other formal systems. Soon however, I had the feeling that, although I was and am convinced of the significance of computer visualizations even in this context, I would be better off in looking at the raw data some of these visualizations were based on, for the things I wanted to do. I should also add that during this research, I ended up with a very strange result, published as [Mol05], that shows through visualization, that there is a certain property concerning how space can be structured, by using the chaos game. I was in fact not searching for this result, I "discovered" it by accident. For me, this experience still serves as a very clear examples of how the computer indeed discloses the universe of discourse.

deduce several types of periods. Before we will enter the universe of tag systems, we will first look at some developments in the context of computability and unsolvability that are very directly connected to the computer.

4.3 Going beyond or not beyond the Turing limit? Developments arising from computability, unsolvability and the computer.

In this section we will consider two theoretical developments in the context of computability and unsolvability that are very closely related to the use of the computer. As in the two previous sections of this chapter, we will not attempt to be complete here, and the reader should thus be warned that the two areas discussed do not exhaust this field.[51] We have chosen these two domains because they beautifully illustrate how the computer has given rise to research on theoretical and practical limits of solvability. The domains we will discuss are *computational complexity theory* and the rather recent development of what has been called *hypercomputability*, the idea of being able to go *effectively* beyond the Turing limit.

4.3.1 On the practical feasibility of the computable: Computational Complexity Theory.

> The entscheidungsproblem does have practical importance in addition to it's philosophical significance. Mathematical proof is a codification of more general human reasoning. An automatic theorem prover would have wide application within computer science, if it operated efficiently enough. Even though this is hopeless in general, there may be important special cases which are solvable. It would be nice if Church's or Turing's proofs gave us some information about

[51] A domain that will for instance not be discussed here is *Algorithmic Information Theory*, where complexity is not defined in terms of the smallest number of steps needed to compute something, but in terms of the smallest possible algorithm to compute something. The theory was founded by Chaitin, Kolmogorov and Solomonov in the sixties. See for example [Cha87] for a quite clear but formal presentation of this theory.

> where the easier cases might lie. Unfortunately, their arguments rest on "self-reference," a contrived phenomenon which never appears spontaneously. This does not tell us what makes the problem hard in interesting cases. Conceivably, a proof that P is not equal to NP would be more informative.
>
> Michael Sipser, 1992.[52]

In a letter dated 20 March, 1956, Kurt Gödel presented the following problem to John von Neumann ([Göd56], p. 375):

> Since, as I hear, you are feeling stronger now, I would like to take the liberty to write to you about a mathematical problem; your view on it would be of great interest to me: Obviously, it is easy to construct a Turing machine that allows us to decide, for each formula F of the restricted functional calculus and every natural number n, whether F has a proof of length n [length = number of symbols]. Let $\psi(F, n)$ be the number of steps required for the machine to do that, and let $\varphi(n) = \max_F \psi(F, n)$. The question is, how rapidly does $\varphi(n)$ grow for an optimal machine? It is possible to show that $\varphi(n) \geq Kn$. If there really were a machine with $\varphi(n) \sim Kn$ (or even just $\sim Kn^2$) then that would have consequences of the greatest significance. Namely, this would clearly mean that the thinking of a mathematician in the case of yes-or-no questions could be completely[53] replaced by machines, in spite of the unsolvability of the Entscheidungsproblem. n would merely have to be chosen so large that, when the machine does not provide a result, it also does not make any sense to think about the problem. Now it seems to me to be quite within the realm of possibility that $\varphi(n)$ grows that slowly. For 1.) $\varphi(n) \geq Kn$ seems to be the only estimate obtainable by generalizing the proof of the unsolvability of the Entscheidungsproblem; 2.) $\varphi(n) \sim Kn$ (or Kn^2) just means that the number of steps when compared to pure trial and error can be reduced from N to $\log N$ (or $\log N^2$). Such significant reductions are definitely involved in the case of other finitist problems, e.g., when computing the quadratic remainder symbol by repeated application of the law of reciprocity. It would be interesting to know what the case would be, e.g., in determining whether a number is prime, and how significantly *in general* for finitist

[52] From [Sip92], p. 603

[53] Except for the formulation of axioms. [Gödel's note]

> combinatorial problems the number of steps can be reduced when compared to pure trial and error.

This letter by Gödel contains one of the first statements of a problem that is very closely connected to the famous **P** vs. **NP**-question, i.e., he asked whether it is possible to "feasibly" compute for any formula from first-order predicate calculus, whether it can be proven in n steps. This kind of problem, i.e., the problem of whether one can practically compute a certain (decision) problem, is nowadays studied in the context of *Computational Computability Theory*.

In the early days of actual computing with computers, one was soon confronted with this kind of problems. As is recounted by Martin Davis [Dav01a], when he implemented the Presburger procedure[54] on the IAS computer, it did not perform very well because, as is now known, it performs worse than exponential. In this respect, computational complexity theory can be said to be directly linked with the rise of the computer: besides the general unsolvability of certain decision problems one was soon confronted with a whole range of decision problems which seemed to be uncomputable in any practical or realistic way, *although not unsolvable in the theoretical sense.*

Besides its clear connection with the computer, computational complexity theory is very closely connected to the theory of computability and unsolvability. In fact, many of the concepts used in the context of computability and unsolvability recur in the context of computational complexity theory. Thus, computational complexity theory can be said to result from the confluence of, on the one hand, experiences based on computing on a machine, and, on the other hand, mathematical logic.

One of the founding papers of computational complexity theory is the paper by Hartmanis and Stearns, *On the computational complexity of algorithms* [HS65], published in 1965.[55] They formulated the computational complexity of a given

[54] A procedure that makes it possible to compute for any formula in the language of Presburger arithmetic, i.e. a first-order theory of addition, whether it is deducible within that arithmetic.

[55] This paper of course did not come out of the blue. There exist several historical surveys on computational complexity theory and the related P vs. NP problem in which this issue is discussed in more detail. See for example [FH03] and [Sip92]. A paper discussing the official

decision problem through (multitape) Turing machines, introducing hierarchies of computational complexity in terms of the minimum speed needed by a (multitape) Turing machine to compute a given problem:

> The computational complexity of a sequence is to be measured by how fast a multitape Turing machine can print out the terms of the sequence. This particular abstract model of a computing device is chosen because much of the work in this area is stimulated by the rapidly growing importance of computation through the use of digital computers, and all digital computers in a slightly idealized form belong to the class of multitape Turing machines.

Nowadays, the computational complexity of a given problem is still stated in terms of (one-tape) Turing machines and they are thus the generally accepted model to work with. In their paper, Hartmanis and Stearns proposed and developed the notion of measuring the complexity of a problem in terms of the maximum number of steps, i.e., the time, needed to solve a particular instance of the problem, where the complexity of a given problem is a function of the size of the input. Besides time complexity classes, one has also introduced space complexity classes in the meantime, i.e., the size of the memory needed to compute a given problem.

The two most famous complexity classes are the classes **P** and **NP**. The class **P** is the class of decision problems *solvable* by a *deterministic* Turing machine, within a number of steps bounded by some fixed polynomial in the length of the input. Thus, for example, if a given problem can be solved in at most $c \cdot n^2$ steps, where c is a fixed constant and n is the size of the input, the problem belongs to **P**. The class **NP**, i.e., non-deterministic polynomial time, is the class of decision problems that can be solved by a *non-deterministic* Turing machine,[56] within a number of steps bounded by some fixed polynomial. Equivalently, **NP** is the class of decision problems that can be *verified* in polynomial time, i.e., it takes at most a polynomial number of steps to verify whether a solution to a given instance of the problem is correct. To explain this with an example, con-

statement of the P vs. NP problem is [Coo].

[56] Informally, a non-deterministic machine is a machine that has more than one possible move from a given configuration.

sider the *Travelling Salesman Problem*, the problem to determine the shortest route to visit a collection of cities and return to the starting point. While it can be *verified* in polynomial time whether a given number is the correct solution to an instance of the problem, given a certain number of cities in a certain configuration, it is far from clear whether one can *compute* the solution for every instance of the problem in polynomial time with a deterministic machine. If one would be able to prove that one can solve every instance of the Travelling Salesman in at most a polynomial number of steps with a deterministic machine, or, vice versa, that it is impossible to solve the Travelling Salesman problem in polynomial time with a deterministic machine, one would be one million dollar richer in having solved the **P** vs. **NP** problem. Nowadays, the general consensus is that **P** ≠ **NP**. Some, however, still believe that **P** = **NP**, while still others think that the problem is impossible to prove or disprove.

A basic concept in the context of research on the **P** vs. **NP** problem is the existence of a wide range of problems which are known to be **NP**-complete, i.e., problems which are in **NP** and are **NP**-hard. A problem is **NP**-hard if any other problem in **NP** can be reduced to it.[57] In his seminal 1971 paper, Stephen Cook proved that several "natural" problems are **NP**-complete. One year later, Karp [Kar72] used Cook's results to show that 20 other problems are **NP**-complete. Nowadays there are hundreds of problems known to be **NP**-complete, varying over many different fields. The Travelling Salesman problem is just one of the many more famous examples. In this sense, a solution to the **P** vs. **NP** problem would give us important information about problems stated in several different fields of science.

But this is not the only reason why a solution to this problem is considered so important. Stephen Cook summarized the consequences of a proof of **P** = **NP**, clearly echoing some of Gödel's idea about the significance of proving that the

[57] In this context a problem A is understood to be reducible to another problem B, if A is polynomial-time, many-one reducible to B. A set of natural numbers S_1 is said to be many-one reducible to a set S_2, if for each positive integer n in S_1 there is an effective method to determine a positive integer m, such that n is or is not in S_2 according as m is or is not in S_2. Then, if we have a method to determine whether m is in S_2 we would have a method to determine whether n is in S_1. The notion many-one reducibility is due to Post [Pos44].

problem he considered could be solved in polynomial time ([Coo71], p. 9):

> Although a practical algorithm for solving an **NP**-complete problem (showing **P** = **NP**) would have devastating consequences for cryptography, it would also have stunning practical consequences of a more positive nature, and not just because of the efficient solutions to the many **NP**-hard problems important to industry. For example, it would transform mathematics by allowing a computer to find a formal proof of any theorem which has a proof of reasonable length, since formal proofs can easily be recognized in polynomial time. Example theorems may well include all of the CMI prize problems. Although the formal proofs may not be initially intelligible to humans, the problem of finding intelligible proofs would be reduced to that of finding a recognition algorithm for intelligible proofs. Similar remarks apply to diverse creative human endeavors, such as designing airplane wings, creating physical theories, or even composing music. The question in each case is to what extent an efficient algorithm for recognizing a good result can be found. This is a fundamental problem in artificial intelligence, and one whose solution itself would be aided by the **NP**-solver by allowing easy testing of recognition theories.

There are several important analogies between the domain of mathematical logic and computational complexity theory, many of the concepts of the latter being inspired by the former. The nice thing about this analogy is that whereas the results from the previous chapter concern theoretical limits that can never be overcome by any effective procedure as long as the Church-Turing thesis remains valid, computational complexity theory studies limits that can be theoretically overcome but not necessarily in practice. I.e. many basic questions in computational complexity theory, concern “feasible” limits of computability. The notion of “feasibility” however is a vague and intuitive notion. It should thus not be surprising that, as is the case for “computability”, there have been attempts to capture to intuitive notion formally. Indeed, also in this domain there exists a thesis, known as the Cook-Karp-thesis:

> **The Cook-Karp thesis.** *A problem is considered feasibly computable iff. it is polynomial time computable.*

Contrary to the several equivalent theses discussed in the previous chapter, the Cook-Karp thesis is very debatable, since an algorithm that is in **P** may not actually be feasible in any meaningful sense. Indeed, as was e.g. pointed out by Martin Davis ([Dav82], p. 23):

> There has not been extensive activity seeking $O(n^{100})$ algorithms![58] Thus it seems entirely possible, in the present state of knowledge, that all NP-problems have polynomial time algorithms although none has an algorithm which is feasible in any practical sense.

Besides the fact that polynomially executable does not necessary imply feasibly executable, the domain of quantum computing seems to further challenge this thesis. For example, Shor [Sho97] has given a quantum computer algorithm for factorization of integers in polynomial time, while no algorithm is known to factor integers on a Turing machine in polynomial time. One of the problems involved with quantum computing is that for now, quantum computers are only capable to handle only very small numbers, the best quantum computer ever build (until now) to factor a number being only capable to factor up to 15 (whose prime factors are 3 and 5). Of course the field of quantum computing is only starting to develop and it seems a promising new domain for turning problems known to be unfeasible into feasible problems.
Quantum mechanics has also been mentioned in the context of what is now called *hypercomputability*. In this domain one is not interested in feasibly computable problems, but in the question of whether it is possible to feasibly "compute", where feasibly now means executable, the non Turing computable.

4.3.2 The land of Tor'bled-nam. To solve the unsolvable.

> Let us imagine that we have been travelling on a great journey to some far-off world. We shall call this world 'Tor'Bled-Nam'.
>
> Roger Penrose, 1989.[59]

[58] The notation $O(x)$ is used to denote how much time it takes at most for a given algorithm to solve an instance of a given problem, the n in the equation used by Davis indicating the length of the input.

[59] From [Pen89]

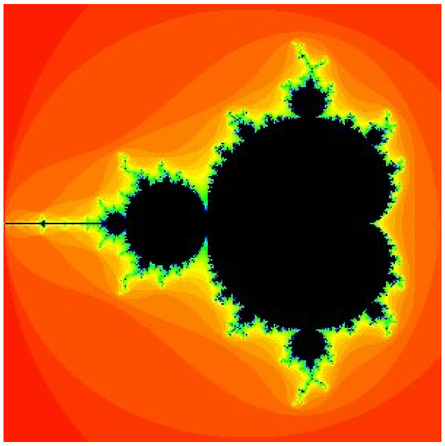


Figure 4.3: The Mandelbrot set.

In *The Emperor's New Mind* [Pen89] Roger Penrose argues, amongst other things, for the physical and Platonic existence of non-recursive phenomena, of which the land of Tor'Bled-Nam is suggested to be one such example. This 'land of Tor'Bled-Nam' points at the structure of one of the most famous fractals, the Mandelbrot set.

The Mandelbrot set is the set of complex numbers c for which the sequence c, $c^2 + c$, $(c^2 + c)^2 + c$, $((c^2 + c)^2 + c)^2 + c$,...remains bounded. Starting with an arbitrary point $c \in \mathbb{C}$ the question posed by Penrose is whether it is computable for any c if c does or does not belong to the Mandelbrot set. Figure 4.3 shows a picture of the Mandelbrot set. Of course, since the Mandelbrot set is defined over $\mathbb{C}$ the question of it being "uncomputable", is ill-posed, as is also noted by Penrose.[60] Nonetheless, the only way to enter the 'land of Tor'Bled-Nam' is through

[60] In 1989 Blum, Shub and Smale presented a model of real computation [BSS89], i.e., computing over the reals, and showed that the Mandelbrot set is indeed "uncomputable" *within the framework of their theory of real computations*. In fact the so-called BSS-model is a model of analogue computing, and assumes that the real numbers are represented exactly, and that each arithmetic operation can be performed in one step. One of the problems, to our mind, with the BSS-model is the fact that besides the Mandelbrot set, very simple fractals like the Koch snowflake are also "uncomputable" in their model. See for example [BC05] for an alternative model, in which the simpler fractals are known to be "computable" while the "uncomputabil-

the computer. Still, Penrose suggests that the "existence" of the Mandelbrot set implies that there "exist" non-recursive phenomena, and this illustrates that it might be possible to go beyond the Turing limit.[61]

Penrose's way of approaching the Mandelbrot set is just one example of how one often reasons in the context of *hypercomputability*: one first defines a non-computable function, or an abstract device that is, *theoretically*, capable to solve e.g. the halting problem, and then uses these theoretically defined functions and devices as arguments for the possibility of developing devices that "compute" the non Turing computable.

This idea of building a "hypercomputer" is closely connected with the computer. When Church, Post and Turing formulated their theses, this was done in a purely theoretical context. With the rise of the computer however, computations have been given a physical form, i.e., as was argued in Sec.4.1, the computer is in a certain way the physical, finite realization of the identifications proposed by Church, Post and Turing. Furthermore, through the computer, it became clear that computability should not remain restricted to "pure" calculability of numbers. Both Zuse and Turing very quickly understood that computers can do much more than computing numbers. Nowadays, computers are involved with almost every aspect of our society.

As a physical realization of the Church-Turing thesis, that has illustrated how general computability actually is, the computer makes clear that the Church-Turing thesis also has a physical side, i.e., the *physical Church-Turing thesis* [Cot03]. It is this thesis, identifying machines with Turing computability, that has become the main target of the advocates of hypercomputability. In the previous chapter, we already mentioned Gandy's paper [Gan80], which contains one of the statements of this physical Church-Turing thesis, indicated as *thesis M* [Gan80]:[62]

ity" of the Mandelbrot set is still an open problem.

[61] Penrose's books [Pen89, Pen94] are of course most well-known for the use of Gödel's incompleteness results to show that there must be something non-computable going on in our mind. Without wanting to oppose this last claim, it should be noted that Gödel's results cannot be used in this context as is e.g. argued in [Dav90, Dav93, Fef88].

[62] It should be noted however that Gandy did not propose this thesis in a context of hypercomputability. Rather, Gandy proposed this thesis, because to his mind, there are crucial steps

> **Thesis M (Gandy).** *Anything that can be calculated by a machine is Turing computable.*

It should be noted that Gandy identifies "machine" with "discrete deterministic mechanical device". According to Jack Copeland, who introduced the now fashionable term "hypercomputability" [Cop98, CP99], one has been too careless in not separating Thesis M from the Church-Turing thesis:

> A myth seems to have arisen concerning Turing's paper of 1936, namely that he there gave a treatment of the limits of mechanism and established a fundamental result to the effect that the universal Turing machine can simulate the behaviour of any machine.

Copeland considers Gandy as one of the few who has carefully distinguished the "stronger" statement expressed in thesis M, that any finitely realizable system can be simulated by a Turing machine, from the "weaker" Church-Turing thesis. Indeed, it seems that Copeland does not want to attack the Church-Turing thesis, but rather Thesis M, although this is not completely unambiguous in his work. By using phrases such as, "computing the uncomputable" [CP99] he seems to actually contradict the Church-Turing thesis, and with it, Thesis M. A similar, to our mind rather ambiguous, attitude can also be found in [EGW04] and [Sta04]. In [EGW04], the "normal" Turing thesis is separated from the so-called strong Turing Thesis, i.e., the claim that a Turing machine can do anything a computer can do. Stannet [Sta04] claims that serious misunderstandings are involved with respect to the Church-Turing thesis, in that many people have interpreted it wrongly as "*a statement to the effect that anything that can be computed by any means whatsoever can be computed by a Turing machine*" ([Sta04], p. 136). According to Stannet, Turing did not intend to identify "effective computation" with "computations by any means whatsoever", *[r]ather, [effective computation] is essentially the highly constrained form*

in Turing's analysis where he appeals to the fact that the computation is carried out by a human being, steps which cannot simply be applied to machines. In other words, Gandy proposed (and argued for) his Thesis M, because he wanted to generalize Turing's arguments from computing human beings to machines.

of behavior effected by human clerks engaged in the production of books of tables;" ([Sta04], p. 137).

It is indeed true that Turing considered a man in the process of computing a number, as the kind of process that can be captured by a Turing machine. In this sense, one can indeed argue, as Gandy did, that other models, *equivalent to Turing machines*, might be more suitable to make the identification with machines. However, the idea of constructing other formalisms because they are more suitable to e.g. capture the machine notion, does not imply that such identifications are "stronger" than the Church-Turing thesis.[63]

Notwithstanding the fact that other formalisms might be more suitable to capture certain notions, one can seriously doubt whether Turing would not have identified his Turing machines with "computations by any means whatsoever" or with what a machine or a computer can do. Besides, it should also be noted that Turing was in fact the only one who started from human computers in order to formulate his thesis. Church started from the vague notion "effective calculability" as used in mathematics, and later clearly stated that there are no fundamental problems to be overcome in identifying "computing machine" with a "human calculator".[64] Post in his turn considered the notion of generated set, and stated that in order for his thesis to gain a more general character, it would be necessary to analyze *all the possible process* that can be set-up by the human mind to generate a set, an analysis which resulted in his formulation 1. However, as is here illustrated, in the context of hypercomputability, one only rarely takes into account anything else but Turing machines. As a consequence, one does not consider all the other formalisms that are covered by the "traditional" Turing thesis and neglects that there is nothing new to the idea of developing other formalism *equivalent to Turing machines*, because they are more suitable for other purposes.[65]

[63] In the previous chapter we also mentioned Markov's thesis, who developed a formalism very different from Turing machines, because he believed that the formalisms already considered by Church, Post and Turing were not suitable for capturing the notion of an algorithm.

[64] See the first part of Church's review [Chu37b] of Turing's *On Computable Numbers*, quoted in the previous chapter.

[65] Another example in this context are cellular automata, which are known to be capable to simulate any Turing machine, but are more well-suited to study self-reproduction in a compu-

But why would some of the researchers in the domain of hypercomputability make such a fuss about the fact that Turing himself would not have identified Turing machines with "machines", thus being able to claim that they are not really opposing Turing's thesis, but rather the physical version of it?[66] The answer is clear: Turing had a much wider concept of machines since he introduced *oracle machines* in his seminal [Tur39], i.e. abstract devices that are indeed capable *in theory* to provide answers to non-computable questions. Copeland and Proudfoot have identified these oracles as *Alan Turing's forgotten ideas in Computer Science* [CP99], the title of one of their papers. Also Wegner et al. make a similar statement, one that also illustrates the ambiguities involved with their attitude towards the "'weak" Church-Turing thesis and the "strong" Turing thesis ([EGW04], p. 160):

> It is little known that Turing had proposed other, non-algorithmic models of computation, and would have disagreed with the strong Turing Thesis [i.e. a Turing machine can do anything a computer can do]. He did not regard the Turing machine model as encompassing all others. As with many other of his ideas, Turing was far ahead of his time. Only now, with the development of new powerful applications, is it becoming evident to the wider computer science community that algorithms and Turing Machines do not provide a complete model for computational problem solving.[67]

tational framework.

[66] To further illustrate the rather ambiguous attitude of some of the researchers in this context, we should mention that Stannet announces in the introduction that he "*shall try to give an engineering solution to the question "Can computation be non-recusive?" Can machines be built – or, at the very least, might natural systems exist? – that perform actions that cannot be simulated by a Turing machine?*" ([Sta04], p. 136). As is clear from this quote, Stannet is convinced that it is possible to build machines that "compute" the non-recursive. Still, he seems to suggest that he is not opposing the Church-Turing thesis, but rather the fact that anything what can be computed by any possible means would be Turing computable. As far as I am concerned, I can only place question marks with these kind of statements. Was it not the main purpose to cover anything we consider (intuitively) computable with e.g. Turing machines?

[67] It should be noted that Wegner et al. not only intend Turing's oracles, but also the choice machines mentioned by Turing in his [Tur37], as the kind of other non-algorithmic models of computation (note the ambiguity in this statement!) Turing would have considered. This leads

Now, we have every respect for researchers trying to overcome what seems, for now, still impossible. However, we believe it very problematic that some of the advocates of hypercomputability, (ab)use Turing's work to strengthen their "arguments". First of all, it should be noted that Turing himself stated very explicitly that oracles cannot be regarded as machines ([Tur39], p. 166–167):

> Let us suppose that we are supplied with some unspecified means of solving number-theoretic problems; a kind of oracle as it were. *We shall not go any further into the nature of this oracle, apart from saying that it cannot be a machine.*

To state that many researchers misunderstand Turing's thesis, because it does not apply to machines, given the fact that Turing himself introduced oracles that can solve the halting problem, can only sound very wrong in the light of this quote. As is also argued in [Dav04, Hod04], there is no clue in Turing's work that he intended to really build an oracle, and it can thus not be claimed that Turing anticipated the agenda of hypercomputability.

Besides this fact, the claim that the idea of an oracle is one of Turing's "forgotten ideas" is a further illustration of this abuse: oracles are far from being forgotten in the literature and are in fact one of the fundamental concepts of recursion theory. Oracles can be used to define what it means for a problem to be computable relative to another problem. In this sense, oracles are a very useful tool in recursion theory and have been basic for research on *degrees of unsolvability*. One can for example use oracles to express Post's problem introduced in [Pos44] and solved, independently, by Friedberg [Fri57] and Muchnik [Muc56]: the problem of whether there can be two recursively enumerable sets that are non-recursive, such that first is recursive relative to the other, but not vice versa, i.e., an oracle for solving the decision of the first would not result in a solution

Wegner et al. to the conclusion that Turing would have disagreed with the so-called strong Turing thesis. Choice machines were described by Turing as follows: "*For some purpose we might use machines [...] whose motion is only partially determined by the configuration [...] When such a machine reaches one of these ambiguous configurations, it cannot go on until some arbitrary choice has been made by an external operator.*" ([Tur37], p. 118) While these choice machines indeed work in a different manner than Turing machines, it has been proven in 1956 [LMSS56] that Turing machines provided with a random number generator can compute only those functions which are Turing computable.

for the second, while the oracle for the second would give solutions for both problems. To state that oracles have been forgotten is plainly wrong, since they are a standard part of recursion (or computability) theory. Furthermore, they are also used in the context of computational complexity theory, although they are not introduced to answer non-computable questions, but rather to determine complexity classes relative to other classes.[68]

The basic difference between the use of oracles in the context of hypercomputability and recursion theory, is the fact that in the former they are considered as machines that could maybe be effectively build, while in the latter, they function as theoretical devices that have shown very fruitful to advance the theory.

As is clear, the "agenda" of hypercomputability is to show that there exist, or that one can construct, (physically realizable) processes or devices that function as an oracle, i.e., they are capable to give answers to uncomputable problems like the halting problem. Several different proposals of such devices have been made, in several different domains. Cotogno [Cot03] makes the following classification: physical supertasks and infinite computations; interactive systems; analog computations and quantum computations. For a more detailed description of the "devices" considered in each of these classes the reader is referred to Cotogno's paper.

Although one can describe these devices theoretically, none has been built until now and it seems rather improbable that, as far as these devices are concerned, one will ever be able to really build one of them. Several arguments, showing that there are serious problems involved with the idea of building a hypercomputer, have already been formulated, so we will not discuss these in any detail here, but merely mention the most basic ones[69].

It has been pointed out by Martin Davis [Dav06b] that in discussing these matters, one should differentiate between the theoretical and the practical in this context, as is the case for "theoretical" and "practical" computability. A Turing machine is a theoretical device, used in theoretical computer science. A

[68] For example, the seminal paper by Cook [Coo71] mentioned above uses the notion of an oracle.

[69] Some papers giving several different arguments are [Cot03, Dav04, Dav06a, Dav06b].

"real" computer cannot be identified with a Turing machine since it is finite. The same goes for "hypercomputers". Although one can invent many theoretical devices that are capable to solve unsolvable problems, like Turing's oracles, one should take into account physical constraints if one really wants to build one.

The first major problem related to this, is the fact that we humans are finite.[70] Suppose that someone would claim that he has build an apparatus that "computes" a non Turing computable sequence. How will we humans be capable to check this? Indeed, given our finiteness we can only perceive but a finite amount of data, and no finite amount suffices to distinguish the computable from the non-computable sequences.

Right now, the most obvious basis for building a hypercomputer is through physical theory and most of the models proposed are indeed based on physics.[71] But here there are some basic problems. One of the biggest problems here, is that the theory should be able to predict (and compute with) a non-computable real number to infinite precision in finite time! For now, no physical theory is capable to do this. As is remarked by Davis, "*[u]ntil now, all physical theory has been content with predictions that can be verified to within less than, say 50 digits*" ([Dav04], p. 207). Even if we would be guaranteed that the physical theory is 100% precise, resulting in infinitely precise number, another question is: How will one build a device that is infinitely precise, following the theory, and, how will we humans be able to check this? Is it not basic to physics that one should be able to experimentally verify that the results are correct? Indeed, what kind of measuring device would one have to develop, that measures in finite time the correctness of an infinite number, and, foremost, how can we be sure about

[70] The arguments provided here, come from [Cot03, Dav04, Dav06b].

[71] Some other models are: infinite computation and interactive computation. However, for infinite computation, like e.g. so-called Zeno machines, one assumes that it is possible to perform infinite computation in finite time. Although one cannot exclude this possibility in principle, this can only work under highly idealized conditions, as is argued by Cotogno [Cot03], not taking into account certain quantum and thermodynamical constraints. As for conversational computation (see e.g. [EGW04]), Cotogno has argued that it can be simulated by a Turing machine, and has no hypercomputing aspect, unless viewed in a non-effective way, i.e., through infinite computation.

that? Would it not be frustrating if we would have a theory, and a device based on that theory, that solves e.g. the halting problem, but that we would not be capable to check and know the answers? In other words, is it not the case that, if it would be possible to develop a physical theory that solves the unsolvable, one would also have a theory that outruns us humans? Copeland [Cop98] has proposed the hypothesis that the human mind itself is an oracle. Well then, I ask, as I did in the previous section, if Dr. Copeland wants to take this hypothesis serious, I would like to see his mind solve the decision problem for the very simple tag system $v = 3, 1 \rightarrow 1101, 0 \rightarrow 00$. Of course, I am not sure whether this tag system is solvable or not, but the very simple fact that it is at least very hard to prove this, is reason enough for me to be convinced that my mind is not an oracle, except if I would suppose that this oracle part of my mind would only work for very specific cases, in a way that it is impossible for me to solve the halting problem, because I cannot control the oracle that is my mind. But what would be the point then?
A last objection, pointed out by Davis [Dav04], is the fact that for now one does not take into account questions of computational complexity ([Dav04], p. 209):

> Copeland's supposed oracles not only store information regarding unsolvable problems, but apparently spew out the information with no significant delay. Of course, in reality, even if [...] an actual oracle materializes, it will be quite useless if, for example, the time needed for the answer to a query to the oracle is an exponential function of the size of the query.

Clearly, this objection stands in sharp contrast with the following, to our mind, rather careless remark by Wegner et al. ([EGW04], p. 190):

> We think that the $P = NP$ question will lose its significance in the context of super-Turing computation.

Does all this critique imply that we completely oppose the possibility of a physical theory that is capable to give answers to non-computable questions? No. However, and here we follow Davis, the current approaches to hypercomputability cannot serve this goal, since they rely on current physical theory, which seems not suitable for what one needs in order to hypercompute.

In the end, one cannot but conclude that, their claim amounts to the following: *if* it were possible to build an oracle, it would be possible to solve unsolvable questions. This is a quite trivial remark, especially in the light of the developments based on Turing's so-called forgotten ideas in recursion (or computability) theory. We certainly do not want to exclude the possibility of a physical theory that is capable to cope with the infinite in finite time, however, such theory would require a revolution in physics itself. In supposing that one day some genius would develop such a theory, and, on the basis of the theory, build a device that "solves" the unsolvable, this would be a very exciting and fundamental philosophical discovery. Indeed, man would be confronted with something that not only goes beyond the Turing limit, but also beyond man's limit, an aspect of hypercomputability that is hardly taken into account by its advocates.

4.3.3 Conclusion

I was first confronted with the idea of "hypercomputability" at a time I was reading about chaos theory, wondering how chaotical systems might be connected to unsolvability. It was in this context that I read the paper by Da Costa and Doria [dCD90], in which they prove that there is no method to determine whether a given set of equations is chaotic or not, a paper published at a time that the domain of hypercomputability was not yet as fashionable.[72] In that same paper, the authors consider the possibility of developing devices that "compute" the non Turing computable on the level of the *Gedanken* experiment, and admit that a real implementation might be "*tricky from a practical point if view.*". Being unaware at that time of the domain of hypercomputability, and the fact that a large number of researchers situate themselves in this domain, I was rather surprised by their proposal, but even more by the following statement [dCD90]:

> We cautiously suggest that the trouble may lie not in some inner weakness or flaw

[72] Besides Da Costa and Doria, there were several others who constructed uncomputable problems in the domain of physics, that are now discussed in a context of the possibility of "computing" the non Turing computable. See e.g. also [PER79, Sca63].

> of mathematical reasoning, but in a too narrow, too limited concept of formal system and of mathematical proof. There is a strong mechanical, machinery-like archetype behind our current formalizations for the idea of algorithmicity that seems to stem from an outdated 17th century vision à la Descartes [...] Also, a first-order language such as the one for Zermelo-Fraenkel theory is too weak: *even if we can prove all of classical mathematics within it, it is marred by the plethora of undecidability and incompleteness results that we can prove about it, and which affect interesting questions* [m.i.] that are also relevant for mathematically-based theories such as physics. The authors certainly do not know how to, let us say, safely go beyond the limits of the presently available concepts of computability, algorithmicity, and formal systems, but they feel that if there are so many quite commonplace things that 'should' somehow be provable or decidable within a sensible mathematical structure, and which, however, turn out to be algorithmically undecidable or unprovable, then one cannot blame the whole of mathematics for that. Mathematics is not at fault here. The problem lies in our current ideas about formalized mathematics. They are too weak. We must look beyond them.

As is clear from this quote, the reason for Da Costa and Doria to ask for another "hypercomputable" kind of mathematics is not the idea of mechanization itself, but rather the fact that the formalisms underlying it have shortcomings – they are marred with the plethora of unsolvability and incompleteness. But why is this a problem? Their answer: because there are "so many" commonplace things which *should* not be marred with this plethora. We already pointed out that some of the supporters of hypercomputability have a rather ambiguous attitude towards the Church-Turing thesis. On the one hand, they seem to suggest that they do not want to oppose it, but rather the "stronger" physical version of it. On the other hand, in using phrases like "computing the uncomputable" ([CP99]) or asking questions like "Can computations be non-recursive?" ([Sta04], p. 136) they do oppose the Church-Turing thesis, and with it thesis M. A same attitude seems to underly the quote by Da Costa and Doria, and is rather revealing: their basic problem is not mechanization itself, but the fact that it gives rise to unsolvable decision problems not only in mathematics,

but also in physics.
I have always been attracted to the impossible, and especially its implications for me as a human being. In my mind, the fact that there are limits for us humans, is something very fascinating. This is in fact one of the reasons why I chose to study unsolvability and the question of how the theoretical results that follow from the Church-Turing thesis and diagonalization, are connected to specific formal systems and their execution. In Sec. 3.3 we argued, on the basis of our more historical results, that studying, on the one hand, specific (classes of) cases, and, on the other hand, formal systems that do not necessarily have a direct appeal to intuition, is very important from a philosophical point of view. For me, as a philosopher, exploring variant systems of computability has been basic to accept the Church-Turing thesis and it has been our study of tag systems that really confronted us with our own limitations. From this point of view it is indeed very important to consider other models of computability. However, this does not imply a hypercomputationalist perspective.
To our mind, the supporters of hypercomputability too much neglect this other side of the Turing limit. They do not take into account particular systems, nor do they consider anything else but the Turing limit, and only, in a very general sense. The fact that Copeland considers the hypothesis that the mind is an oracle only illustrates how far he is removed from understanding the true power of "computability" and the Church-Turing thesis. Again, we would like to emphasize, that this statement from our side, *does not imply a computationalist point of view*, assuming that the mind itself is computable. rather, we want to emphasize that *as far as problems are concerned that are situated in the context of computability and unsolvability*, the mind is at least as limited as a universal Turing machine, if one accepts the Church-Turing thesis.

As was argued, both computational complexity theory and hypercomputability are very closely connected to, on the one hand, theoretical computability and unsolvability, and, on the other hand, the computer itself. These two developments stand in a very sharp contrast with each other. Computational complexity theory investigates the practical feasibility of the computable, hypercomputability studies the possibility to make feasible the non-computable. In both domains, theses have been proposed (or opposed) that are very closely connected to the Church-Turing thesis. However, the Cook-Karp thesis is clearly not as strong and well-supported as the Church-Turing thesis, while the advocates of hypercomputability have a rather ambiguous attitude towards the Church-Turing thesis. They do not explicitly oppose it, but do not give very clear nor convincing arguments for differentiating it from the thesis they do want to oppose, i.e., the physical Church-Turing thesis. It is remarkable to see that what they actually oppose is not really the Church-Turing limit, but indeed the idea that the computer is the actual physical limit of computability. This position can only be called "strange" in the light of the previous chapters. Indeed, since the computer can be regarded as a finite version of a universal Turing machine, given its physical form, and is thus in fact a "weaker" form of the Turing limit, it would seem more logical if one would oppose the Church-Turing thesis itself, rather than its physical version.

Relating the two developments described here to the previous chapters, it is clear that the theoretical developments in the twenties and thirties, when combined with the physical form of these developments, have given rise to a variety of new fundamental problems, problems closely connected to the limit imposed by the Church-Turing thesis. In both domains however, Turing machines and the Turing limit are the dominant paradigm. While this is not a real problem with respect to computational complexity theory – it is clear that Hartmanis and Stearns preferred Turing machines because they are close to real computers – we believe this is a problem in the context of hypercomputability. In not taking into account the several other theses as well as the proper context in which Turing wrote his 1936 paper, some of the advocates of hypercomputability make fundamental mistakes and are unclear about what they are actually opposing. Indeed, by stating that they do not want to oppose the Church-Turing thesis, because Turing himself did not identify physical machines with his Turing machines, but rather human calculators, they first of all, neglect Church's and Post's statements in this respect, and, secondly, do not see that the true purpose of the theses was in effect to capture computability and effectivity in all its possible forms. If e.g. Turing would have believed that his thesis is merely a theoretical construct, while being convinced that it is possible to construct physical machines that are not subject to the Turing limit, I don't think that he would have claimed to have proven that the halting problem is unsolvable!

4.4 Conclusion.

In this chapter we have shown in how far the computer can be considered as a physical realization of computability (Sec. 4.1). It was argued that as such a physical realization, that can be effectively used to compute very quickly, the computer itself has generalized the notion of computability. It is no longer restricted to "pure" computations. Important here, is that the computer did not solely arise from a formal-theoretical analysis of what we exactly mean with a computation. Rather, it resulted from the experiences *and* abstract thinking of both the logicians and the engineers, or the logician-engineers (like Turing and Zuse) when developing this device.

As a physical realization of computability, the computer has not only shown us the rich variety of "tasks" covered by computability – in the end, anything that can be computed by a computer *as we know it today* can be computed by a Turing machine – but has also made available the universe of discourse to an extent that was not possible before (Sec. 4.2). In this sense, the computer has given rise to fundamental new results in several different domains. Equally important in this respect, however, is also the fact that as such, the computer has made it possible to study the formalizations it is the physical realization of. In part II, we will give some examples where the computer has played an important role in the establishment of certain results.

The computer also resulted in developments that are closely connected to the question of computability and its (theoretical and practical) limits (Sec. 4.3). Computational complexity theory studies the feasibility of the computable, while hypercomputability takes into account the question of the physical realizability of a device that "computes" the theoretically uncomputable.

To summarize, the computer's role in the context of computability and unsolvability can hardly be underestimated. Even though it is but a finite realization of the formalisms considered in the previous chapters, it has lead to developments closely connected to the Church-Turing thesis, some of them arising from the actual use of the computer, some from theoretical considerations connected to computers.

In part II of this dissertation we will study tag systems. Our purpose here is not to go beyond the Turing limit. Rather we would like to show that there are enough problems remained unsolved on the level of a class of formalisms that are known to be capable to compute what any Turing machine can compute. Although part II is far less historical as compared to part I, there are several connections between the two parts. In fact, our research on tag systems was very much inspired by our way of thinking on computers, our more historical study of the work by Church, Post and Turing and our more or less philosophical conclusions on the basis of this study. First of all, tag systems are in a way a "class example" of a class of formal systems that are far removed from our intuitions of computability. In the end, one should not forget that they were developed by Post in order to find abstract forms of mathematics, far removed from interpretation or meaningful concepts. Part II is based on the assumption that studying systems further removed from intuition, allows to more easy focus on more abstract properties of those systems. Secondly, most of our results are based on a study of specific (classes of) systems of tag. Finally, as will become clear, the computer is considered as an important tool to not only study tag systems, but also to get new results, results which can be heuristic as well as theoretical.

Part II

Tagging

> [...] [...] time after time I found that *because* of my ignorance of these antecedents, *I had not, nor could have, really understood those ideas*. All the logical analysis n the world will not reveal the intentions behind ideas, and without these intentions one all too easily misunderstands and misjudges the ideas and theories of a writer no longer living. [...] one also finds that current ideas and results can illuminate older and crustier ideas. The lesson seems to be this: we cannot fully understand our own conceptual scheme without plumbing its historical roots, but in order to appreciate those roots, we may well have to filter them back through our own ideas.
>
> Judson C. Webb, 1980.[73]

In Part I we discussed the early beginnings of unsolvable decision problems, and showed how, through the computer, new problems and possibilities have arisen in this context. Although this first part could be understood as a research in itself, the main recurring theme will now be further investigated from a completely different point of view. This main theme is the significance of the actual use, discourse and physical implementation of the several different formal systems considered by Church, Post and Turing for the abstract results of unsolvability and the closely connected theses of identifying the intuitive notion of computability with one of these formalisms. In this second part we will investigate solvability and unsolvability by starting from the form that first led Emil Post to the idea that, contrary to what he believed previously, there might not exist a positive solution to the decision problem for *Principia Mathematica*. Here, we will study Post's form of Tag.

[73] From [Web80], p. xii.

Chapter 5

Introduction. Why tag systems?

> All important fields of human endeavor start with a personal commitment based on faith rather than on result. [...] Only a sour, crabbed and unadventurous spirit will hold it against us.
>
> David Marr, 1976.[1]

Since Post spent nine months of his research during his Procter fellowship at Princeton (November 1920 – July 1921) on tag systems there has not been very much research on this class of systems relative to some of the other formalisms significant in the context of computability and unsolvability. As a consequence their influence on computer science and mathematical logic is nihil as compared to the influence of e.g. Turing machines and λ-calculus.[2] This does not imply that they are not known to a wider audience – they are frequently mentioned in e.g. surveys of or introductions to mathematical logic or theoretical computer science. It only means that not many researchers really got involved with tag systems. As far as the author knows there are about 15 researchers, excluding Post, who have researched and published on tag systems: Marvin Minsky [Min61, Min62, Min62b, CM63, Min67], Yuri Maslov [Mas4a, Mas4b], Hao

[1] [Mar76]

[2] The fact that the entries on Wikipedia for λ-calculus, Turing machines, primitive and partial recursive functions are significantly longer as compared to that for tag systems (of which the major part is reserved for cyclic tag systems) can be considered as a kind of superficial indication of the infamousness of tag systems relative to these other formalisms.

Wang [Wan3a], Shigeru Watanabe [Wat63], Philip K. Hooper [Hoo6a], Stal Aandera and Dag Belsnes [AB71], Charles E. Hughes [Hug73], David Pager [Pag70], Turlough Neary and Damien Woods [NW06a], Frank Rubin [Rub88], Brian Hayes [Hay86, Hay6a] and Stephen Wolfram [Wol02].[3]

The limitedness of this research on tag systems naturally leads to the following question: given the fact that tag systems have hardly been investigated, why would one be interested at all in these systems? Of course one could claim that the meagerness itself of this research is reason enough but one could also argue to the contrary and claim that tag systems simply haven't got much attention because they are not as interesting as, e.g., Turing machines or that this research would add nothing new to the existing literature.

While it was at first not "part of the plan" to devote more than 1/2 of this dissertation to tag systems, I got more and more attracted by these fascinating systems. In this sense, it was intuition rather than theoretical considerations that led me to this research, and our choice for tag systems can thus be called a subjective choice, one however that is completely in line with some of the conclusions from part I. We hope that through the results and comments to be given in the chapters to follow, in the spirit of the idea of "Legitimation durch Verfahren,"[4] we will be capable to convince the reader that tag systems were and are in need of more research. In the remainder of this short introductory chapter I would like to point out some of the typical general features of tag systems that make them interesting systems to study, at least for me.

As was pointed out in 2.2.5, one of the reasons why Post wanted his *Account of an anticipation* to be published was the difference in method as compared to Church, Gödel or Turing: he was more interested in the outward forms of symbolic logic, rather than in the logical concepts expressed through it. Tag systems were one of the forms that resulted from this method. As a consequence it is very hard to attach any concrete interpretation to them since they are far more abstract. This is to our mind a very important feature of tag systems. First of all, because of this abstract character, they are far less intuitively appealing

[3] In Section 6.1, the research that has already been done on tag systems, will be discussed in more details.

[4] "Justification by doing"

than Turing machines and do not arise from an analysis of the intuitive notion of computability. In this sense, they have seriously challenged the idea I had of a computation before I investigated these systems and in fact generalized it. Secondly, we believe that because of their abstract character, tag systems are very suitable to confront one with his/her own limitations on a very low level. Indeed, as was already explained, even the very simple example of a tag system mentioned by Post is very hard if not impossible to predict. For me personally, tag systems have played the role of showing me how general the limit implied by the theses discussed in the previous chapters actually are.
Another feature of tag systems connected to their formal simplicity is that they are very easy to implement on a computer and allow easily for a more "experimental" approach. In chapter 8 we will use this approach to study tag systems.
A further consequence of their abstractness is that it seems rather straightforward to find a large number of very small tag systems that are very hard to get a grip on and might be unsolvable.
A very interesting opportunity offered by tag systems, that cannot be fully explored in the present research, is to study the connection between number theory and formalized systems of logic. Post repeatedly mentions that there is a close connection between tag systems and number theory, through the *regularity of always removing v letters*. In chapter 9, Sec. 9.4.1 we will prove that tag systems are closely connected to an intricate problem of number theory, and further discuss these matters.

Although we have pointed out here some features of tag systems, they are merely general aspects that have yet to prove their merits in the following chapters. It is only in actually studying tag systems, that they show their true character. Still, we believe it is important to give the reader at least one clue from the existing literature that indicates where tag systems might play a significant role. There is indeed such a clue: tag systems' role in the research on small universal machines and limits of solvability and unsolvability. In the fifties and sixties of the 20th century, there seemed to have been a small hype for finding the smallest possible Universal machine, the size of a machine being measured by the product of the number of states and symbols.[5] The winner of this "competition" was, for a very long time, Marvin Minsky's 4-symbol, 7-state machine [Min62, Min62b]. Without going into the question of why searching for (smallest) universal systems might be interesting (see Ch. 9, Sec. 9.1 and 9.2), it is significant to note that Minsky's 4x7 machine was proven to be universal because it can compute what any tag system with $v = 2$ can compute. Since it was already proven by Minsky that there are universal tag systems [Min61, Min62, CM63] with $v = 2$, tag systems that are able to represent any Turing machine, this proof is valid. For the last twenty years, searching for smallest universal machines has gained new attention, research now being explicitly situated in the context of studying the limits of solvability and unsolvability. Remarkable here is that tag systems lie at the basis of many of the small universal systems known, including cellular automata [Coo04], Turing machines (See e.g. [Rog96]) and circular Post machines (See e.g., [KR01]).[6] Despite the significance of tag systems in this context, any extensive research on the limits of solvability and unsolvability on tag systems is still lacking. In chapter 9, Sec. 9.4 we will tackle this specific problem, and study limits of solvability and unsolvability in

[5] In his [Sha56], Shannon argued for the interchangeability between number of states and symbols, since the product of both has a certain invariance. He thus suggested this product as a measure for the size of a Turing machine. Minsky took this over in his [Min62b] and nowadays it is the conventional measure. For more detailed information see Ch. 9.

[6] It should be pointed out here that tag systems are of course not the sole means for constructing (small) universal systems in the context of determining limits of solvability and unsolvability. For example, in the domain of diophantine equations one has also done some serious research on limits of solvability and unsolvability, where universal Diophantine equations are constructed by other means (e.g. through simulation of partial recursive functions, Turing machines and register machines). Since Matiyasevich [Mat70] has proven that Hilbert's tenth problem is recursively unsolvable, one has searched for small universal diophantine equations, while there has also been research on solvable classes. The paper by Jones [Jon82] gives an overview of the smallest known universal diophantine equations (in terms of the number of unknowns or the total degree) and gives several examples. It should be noted that, as is the case in the context of Turing machines, it is still not known what the minimum degree or the minimum number of unknowns is to obtain a universal diophantine equation. It is known however that the cases with degree 2 are solvable, while one can construct universal equations for any degree equal to or greater than 4.

tag systems.

Part II is divided into several chapters. We will first discuss most of the existing research results on tag systems and describe the general classes of behaviour in tag systems and their connection to the two forms of the problem of "tag". In this way, the first chapter (Ch. 6) can be considered as a kind of introduction to tag systems. In the next chapter (Ch. 7), we will consider several features of tag systems, called constraints, that have been implemented in an algorithm for generating tag systems that might be considered intractable on a more practical level, i.e., for now it is very challenging if not impossible to develop methods for predicting the behaviour of these tag systems. One such class of tag systems generated was used in several computer experiments, the results of which will be described in Chapter 8. In the final chapter 9 we will situate tag systems in the research on limits of solvability and unsolvability. It should be pointed out here that, besides tag systems, our main focus will be on research on limits of solvability and unsolvability in Turing machines. Other computational systems in which finding minimum limits for unsolvability or upper limits for solvability will not be taken into account.[7]

[7] But see footnote 6 in this respect.

Chapter 6

Preliminaries: Some basic aspects of Tag systems

In this chapter we want to give the reader an introduction to tag systems. In a first section, we will discuss most of the existing literature and results on tag systems. This section not only serves the purpose of giving an overview but can help to make the reader more familiar with the inner workings of tag systems. The last subsections of this section, will make clear that tag systems are very 'stubborn' systems, easy at first sight but particularly hard to structurally unravel. In a second section, we will discuss the several classes of general behaviour in tag systems, as first described by Post, and furthermore link these classes with the two forms of the problem of "Tag". After having given an example of how one might proceed to prove specific instances of tag systems solvable, we will prove a small theorem concerning the reducibility of the decision problems for specific classes of tag systems to other classes of tag systems, through decomposition.

6.1 Discussion of published results on tag systems.

> A number of [...] investigators [...] have worked on the problem [...], and in recent years they have brought the power of the computer to bear on it. Nevertheless, we live in a state of almost perfect ignorance about the nature of Post's tag system[s].
>
> Brian Hayes, 1986.[1]

The existing results by on tag systems can be subdivided into two categories. On the one hand there are theoretical results concerning the whole class of tag systems, related to general mathematical properties such as unsolvability and universality. On the other hand, there are some publications that try out a more concrete approach to tag systems, mostly starting from one and the same specific case the tag system mentioned by Post, defined in Sec. 2.2.3.

6.1.1 General Theoretical results

Tag systems, Universality and unsolvability

Tag systems were first proven to be unsolvable (relative to Turing machines) by Marvin Minsky in 1961 [Min61], after the problem was suggested to him by Martin Davis. Some months after the publication of this first proof, Minsky, in collaboration with Cocke, provided an alternative proof, published as [Min62, Min62b, CM63]. Hao Wang somewhat modified this second proof in order to limit the length of the words to be tagged at the end of a string.
The first proof by Minsky will be described here, but not in all its details. It is added here because it was the first proof of the general unsolvability of tag systems. The second proof will be described in all its details, since it will be discussed later on in Chapter 9. Wang's proof will not be included, since it basically relies on the same methods used by Minsky in his second proof.

§1. Minsky's first proof To prove the decision problem for tag systems unsolvable, Minsky showed that tag systems can "compute" anything computable

[1] [Hay86], p. 21

by a Turing machine. While the construction in his [Min61] is rather ingenious, it is also rather complex, contrary to the proof he found some months later.
The proof is not a direct reduction of Turing machines to tag systems. It is first shown that any Turing machine can be reduced to a 2-tape non-writing machine. Then, it is proven that any 2-tape non-writing machine can be reduced to a tag system. The reduction of Turing machines to 2-tape non-writing machines uses a kind of Gödel numbering, relying on a factoring method.
A 2-tape non-writing machine, consists of two semi-infinite tapes, each of which is completely blank, except for a single mark indicating the location of the square at the left end of the tape. The only thing such machines can do is move left or right and "recognize" that they are at the left end of one of their tapes. Now how does the encoding of a Turing machine into this 2-tape non-writing machines work?
Suppose that the Turing machine we want to simulate is T, T being a 2-symbolic Turing machine,[2] and let us indicate the 2-tape non-writing machine to which T is reduced by T^*.
As we already know from Part I, a Turing machine can be represented by a set of quintuples:

$$q_i s_j : s_{ij} d_{ij} q_{ij}$$

where the q's represent internal states, the s's symbols (0 or 1), and d the direction of the motion of the head (left or right). Using this representation, the operations of T are:

R_{q_i}: Read the tape. If the current symbol is 0, go to W_{i_0}. If the current symbol is 1, then goto W_{i_1}.

W_{i_0}: write s_{i_0}, go to M_{i_0}

M_{i_0}: Move in direction d_{i_0}, go to $R_{q_{i_0}}$

W_{i_1}: write s_{i_1}, go to M_{i_1}

M_{i_1}: Move in direction d_{i_1}, go to $R_{q_{i_1}}$

[2] Since Shannon has proven in [Sha56] that any n-symbolic Turing machine can be reduced to a 2-symbolic Turing machine, this does not lead to any problems with respect to the proof.

If for some i and j there is no quintuple $q_i s_j$, we can set W_{ij} equal to the halting instruction. For each instruction of T, T^* will have a corresponding set of instructions, and whenever T executes an instruction, T^* will execute the instructions from the corresponding set. In order for T to be reducible to T^*, T^* must be able to represent the complete state of T, each time T has completed an instruction. The *complete state* is given by:

(1) the content of the tape

(2) the location of the reading head on the tape

(3) the machine's internal state

All three tapes of these machines are semi-infinite, ending left. In counting from the left, the number x denotes the current square of T. At any moment in the operation of T its tape contains a finite amount of data, represented by the integer k. The number k is the decimal value of the binary number that represents the contents of the tape, where the content of the 0-th square contains the least significant letter of k. The content of the square x scanned at a given time, is always equal to the x-th digit of k's decimal expansion. Now, at the beginning of each instruction of T, the two tapes T_1^* and T_2^* of T^* will be in the following state. Tape T_2^* will be positioned at its left end. Tape T_1^* will have been moved, such that its reading head is $2^k 3^{2^x}$ squares to the right. In this way, T^* indeed represents the entire content of T (k) as well as the location x of the current square T is scanning. After each read, write or move instruction of T, T^* will be restored in this state, x and k being changed in the appropriate way. Now how will T^* do this? We will not go through the whole proof here, but merely sketch the global idea behind it. T^* is able to "simulate" each of the instructions of T's program through its own instruction table, such that, if T has moved to the right, the reading head of T_1^* will be moved from the $2^k 3^{2^x}$-th square to the $2^k 3^{2^{x+1}}$-th square. Similarly, if T has moved to the left, the reading head of T_1^* will be moved from the $2^k 3^{2^x}$-th square to the $2^k 3^{2^{x-1}}$-th square. If the x-th digit in the binary expansion of k has been changed from a 0 into a 1, T has printed a 1 in its x-th square, the reading head of T_1^* will be moved from the $2^k 3^{2^x}$-th square to the $2^{k+2^x} 3^{2^x}$-th square. Similarly, if T has changed a 1 into 0, the reading head of T_1^* will be moved from the $2^k3^{2^x}$-th square to the $2^{k-2^x}3^{2^x}$-th square. Now, T^* must also be able to determine whether the x-th digit of the binary expansion of k is a 0 or a 1. This is done by repeatedly subtracting 2^x from k, until k is exhausted, keeping track of the parity of the number of times 2^x has been subtracted from k. If the parity is even, x must have been 0, if not, x must have been 1.

Now that we know how the tape must be changed, the question of course becomes, how will this change be executed? Instead of going through all the operations, the general principle behind them can be explained by showing how T^* is able to perform the right actions depending as to whether T reads a 1 or a 0. The first step is to make a copy of k so that T_1^* is restored i.e. the head is moved from the $2^k3^{2^x}$-th to the $2^k5^k3^{2^x}$-th square. This is done by an iterative method in which $2^k3^{2^x}$ is first changed to $5^k7^k3^{2^x}$ and then to $2^k5^k3^{2^x}$. The first step of this iterative process is accomplished by a subroutine called $C(2,35)$ which will:

(1) divide the length of the T_1^* tape by 2; then

(2) multiply its length by 35, goto (1).

This process is repeated until (1) fails and the length of the tape is no longer divisible by 2, $2^k3^{2^x}$ being changed to $5^k7^k3^{2^x}$. Then, $5^k7^k3^{2^x}$ is changed to $2^k5^k3^{2^x}$ by a similar subroutine, $C(7,2)$, consecutively dividing by 7 and multiplying by 2, until we can no longer divide by 7. In general $C(S,T)$ indicates a loop, of consecutively dividing by S and multiplying by T, until we can no longer divide by S. Now we still have to repeatedly subtract 2^x until k is exhausted. This is done as follows:

$$2^k5^k3^{2^x} \xrightarrow{C(15,7)} 2^k5^{k-2^x}7^{2^x} \xrightarrow{C(35,3)} 2^k5^{k-2\cdot 2^x}3^{2^x} \xrightarrow{C(15,7)}$$

$$2^k5^{k-3\cdot 2^x}7^{2^x} \xrightarrow{C(35,3)} \ldots$$

This cycle is repeated until the substraction is completed, keeping track of the parity i.e. whether the last operation was a $C(35,3)$ or a $C(15,7)$ loop. After this is done, the original tape is restored by a simple $C(7,3)$ operation (if the last

routine performed was $C(15,7)$). T^* then starts with the set of instructions corresponding with W_{i_0} or W_{i_1} of T depending as to whether it ended after having performed $C(35,3)$ or $C(15,7)$ respectively.
This same kind of reasoning can be applied to the operations of moving to the left or right, or write 0 or 1. For example, to move to the right, changing $2^k3^{2^x}$ to $2^k3^{2^{x+1}}$, the following process is performed:

$$2^k3^{2^x} \xrightarrow{C(3,5)} 2^k5^{2^x} \xrightarrow{C(5,9)} 2^k9^{2^x} = 2^k3^{2^{x+1}}$$

Now that the global encoding scheme from T into T^* has been described, we still have to further specify the routines $C(S,T)$ as lists of instructions for T^*. Since multiplication and division are the operations T^* has to perform in order to execute a subroutine $C(S,T)$, Minsky shows how one can encode these operations over an arbitrary number (called MPY(T) and DIV(S)) in T^*. We will not describe in detail how this can be done, since this is, with a little bit of thinking, rather straightforward to program (T_2^*'s role is crucial here, functioning as a kind of calculation sheet for MPY and DIV) .
In assembling MPY(T) and DIV(S) in the right way, using the right indices, T^* is then able to represent any complete state of T, emulating the instructions of T. It should however be noted that in order for the solution for the Post tag problem to work, the program for $C(S,T)$ only uses subprograms MPY and DIV, with prime parameters. Suppose that $S = p_{i_1}p_{i_2}...p_{i_m}$ and $T = p_{j_1}p_{j_2}...p_{j_n}$, where no p_i is one of the p_j's. Then if S or T are not prime, one does not directly multiply resp. divide by S or T, but by the prime factors p of S and T. For example if S = 6, MPY(S) is done by consecutively performing the following two operations: MPY(2), MPY(3). In doing the right assembling, one can then simulate any Turing machine T by a 2-tape non-writing machine T^*. Minsky has thus proven the following theorem:

Theorem 1 *T^* represents the machine T in the following sense. Suppose that the machine T is started in state q_i at the x-th square of its tape, with the binary number k written on its tape. Suppose also that the machine T^* is started at instruction R_{q_i} with its tape T_1^* at its $2^k3^{2^x}$-th square, and its tape T_2^* at its left end square. Then if T ultimately halts on its y-th square with the binary number*

N on its tape, T^* will ultimately halt with tape T_1^* at its $2^N 3^{2^y}$-th square. Thus one can conclude that if T is a universal machine, then so will T^*.

Now, using this theorem, Minsky showed that any Turing machine can be reduced to a tag system, thus obtaining the general unsolvability of tag systems. We already know that every Turing machine T can be represented by a 2-tape non-writing machine with a program consisting of a certain number of instructions I, each instruction being one of the two following forms:

(I_j) MPY(K_j). Go to I_{j1}

(I_j) DIV(K_j). If division is exact go to I_{j_1} else go to I_{j_2}

In each case K_j will be one of four primes 2, 3, 5, 7.[3] So "all" we still have to show is that one can always encode these operations into a tag system, maintaining the right transfer-of-control. The shift number v is always equal to the product of the primes one needs in the encoding.[4].

Now if the j-th instruction is MPY(K_j), we need v+3 letters: $A_j, a_j, B_{j_1}, ..., B_{j_v}, b_j$, for the tag system to be able to "simulate" the j-th instruction. The state of the program at the start of instruction I_j is always represented by a string in the tag system that has the following form: $A_j a_j^n$.[5] This must be converted into $A_j a_j^{nK_j}$, if we want to encode MPY(K$_j$). This can be done by the following production rules:

$$\begin{cases} A_j \rightarrow B_{j_1} B_{j_2} ... B_{j_v} \\ a_j \rightarrow b_j^{v^2} \\ B_{j_i} \rightarrow b_j^{(v-i+1)(v-1)} A_{j_1} \\ b_j \rightarrow a_{j_1}^{K_j} \end{cases}$$

The string $A_j a_j^n$ will indeed be converted into $A_j a_j^{nK_j}$ by repeated application of the production rules.

The encoding for DIV(K$_j$) into tag production rules is a bit more complicated. If

[3] All the subroutines work with numbers for which the prime factors or never larger than 7.

[4] It should be noted that, later on in the paper, Minsky shows that the values of K_j can be reduced to 2 and 3 so v becomes 6

[5] Note that a_j^n means that a_j is repeated n times.

the j-th instruction is DIV(K$_j$), we now need $2v$+2 letters: $A_j, a_j, B_{j_1}, ..., B_{j_v}, b_{j_1}, ..., b_{j_v}$, for the tag system to be able to "simulate" the j-th instruction. We then need the following production rules:

$$\begin{cases} A_j \to B_{j_1} B_{j_2} ... B_{j_v} & \\ a_j \to b_{j_1} b_{j_2} ... b_{j_v} & \\ B_{j_i} \to A^i_{j_1} a^{(v-i)/K_j}_{j_1} & If K_j | i \\ b_{j_i} \to a^{v/K_j}_{j_1} & If K_j | i \\ B_{j_i} \to A^i_{j_2} a^{(v-i)}_{j_2} & If K_j \nmid i \\ b_{j_i} \to a^v_{j_2} & If K_j \nmid i \end{cases}$$

Applying these production rules to the string $A_j a^n_j$, it will be converted to $A_{j_1} a^{n/K_j}_{j_1}$ or to $A_{j_2} a^n_{j_2}$ depending as to whether n is divisible by K_j. Both the production rules representing DIV and MPY can be checked manually. The details are skipped here.

It should be noted that in this encoding, the number of symbols needed for a tag system "simulating" the operations of a given Turing machine, let alone a universal Turing machine, is very large. We can make a rough estimate of this number of symbols in the following way. As was said before, the operations of a Turing machine T can be represented by 5 instructions: $R_{q_i}, W_{i_0}, M_{i_0}, W_{i_1}, M_{i_1}$. Now in order to read a symbol (R_{q_i}), we need 5 subroutines of the form $C(S, T)$. Encoding these as MPY and DIV, we need 6 multiplications and 5 divisions. This implies that, supposing $v = 2 \cdot 3 = 6$, we need $6 \cdot 9$ productions for the multiplications, and $5 \cdot 14$ productions for the divisions. This results in a total of 124 productions for one single reading operation. For the tape to move to the left, we need 2 subroutines of the form $C(S, T)$. Encoding these with MPY and DIV, we get 3 multiplications and 2 divisions, resulting in a total of $28 + 27 = 55$ instructions. Doing the same calculations for moving to the right we get 60 instructions. For writing a 0 we need 55 instructions, for writing a 1 we need 60 instructions. Making a kind of "best-situation" estimate, supposing the Turing machine one wants to represent only moves to the left, and always prints 0's, never 1's, we need at least $124 + (4 \cdot 55) = 344$ productions in the tag system for

each state.[6] Now, since the shortest 2-symbolic universal Turing machine has 18 states, a representation of this Turing machine into tag systems – thus constructing a universal tag system with an unsolvable decision problem – would lead to a tag system of at least 18*344 = 6192 productions (and thus symbols). Besides the gigantic proportions of the tag system, the time needed by the tag system – the number of productions to be performed – is equally gigantic since we are working with very large numbers, resp. strings.
Although Minsky must have been satisfied to have solved Post's problem of tag, he understood that improvements could be made ([Min61], p. 450):

> We have been unable to reduce P [= v] further, and the prospects seem gloomy. (We have been unable, as was Post, to prove a negative result for $P = 2$.) It would be desirable to reduce the exponentiation level in this representation but the "Tag" systems seem intractable in regard to lower level manipulations. We have been unable even to find productions which can reduce the length $n > P$ of a string to $n - 1$, for arbitrary n.

Some months later, Minsky, in collaboration with Cocke, managed to prove that any Turing machine can indeed be reduced to a tag system, with a shift number $v = 2$. This encoding leads to the possibility of constructing shorter universal tag systems.

§2. Minsky's second proof The first proof of the general unsolvability of tag systems is rather complicated and impractical (given the level of exponentiation). This motivated Minsky to search for a more "elegant proof" of this result ([CM63], p. 1):

> [The previous proof] is very complicated and uses lemmas concerned with a variety of two-tape non-writing Turing machines. Our proof here avoids these otherwise interesting machines and strengthens the main result, obtaining the theorem with a best possible "deletion number" P = 2. Also the representation of

[6] Of course the calculation was based on the original encoding of Minksy using four prime numbers. An encoding with two prime numbers might result in a shorter list of productions. Still, even if we would be able to halve this number, it remains a fact that we need relatively large tag systems to do the simulation properly.

> the Turing machine in the present system has a lower degree of exponentiation, which may be of significance in applications.

In this second proof, it is shown how any two-way infinite Turing machine T can be directly reduced to a tag system (without the intermediary reduction to two-tape machines).

In [CM63], the usual formalization of the operations of a Turing machine – the machine is in state q_i, depending on the symbol it is scanning it writes $s_{i,0}$ or $s_{i,1}$, performs move $d_{i,0}$ or $d_{i,1}$ and goes to state $q_{i,0}$ or $q_{i,1}$ – is replaced by a slightly different, though completely equivalent one: The machine is in state q_i, it writes s_i, performs move d_i. Only then it reads the tape, and depending on whether a 0 or a 1 was scanned, it goes to state $q_{i,0}$ or $q_{i,1}$. In converting the first formalization of a Turing machine to this second form, of course the number of states has to be doubled.[7] Within this formalization, a Turing machine is represented by a set of quintuples of the following form: $\{q_i : s_i, d_i, q_{i,0}, Q_{i,1}\}$.

The contents of the tape can then be represented as follows:

...	a_6	a_5	a_4	a_3	a_2	a_1	a_0	α	b_0	b_1	b_2	b_3	b_4	b_5	b_6	...

where the machine is in state q_i and α is the digit on the scanned quare. The complete state (contents of the tape, the machines present location on the tape and its internal state) can now be represented by 3 numbers: q_i (the state the machine is in after having read α), $m = \sum a_i 2^i$ and $n = \sum a_i 2^i$, α not being included because it has become redundant within the present formalization). It is assumed that the machine is binary, the a's and b's thus always equal to 0 or 1. All but a finite number of a's and b's are zero, so the summation is defined.

Now, since the complete state of a machine is given by the triple $\{q_i, m, n\}$, the effect of applying a quintuple $\{q_i : s_i, d_i, q_{i,0}, q_{i,1}\}$ to such triples – the way the triples are transformed from each moment to the next – can be very eas-

[7] Minsky finds this formalization of a Turing machine a bit more honest than the usual one. As is stated in [CM63]: "*In a way, however, this new formalism is a little more honest, because in the usual formalism the machine needs an extra memory in which to store the symbol just read, while it writes and then moves. Here, the reading operation causes an immediate state-change, and no implicit symbol-memory is required.*"

ily described. For example if our machine were to move to the right, the triple $\{q_i, m, n\}$ will be changed as follows:

- Change m to $2m+0$ or $2m+1$ (depending on whether $s_i = 0$ or 1)
- Change n to $\lfloor \frac{n}{2} \rfloor$, i.e. the largest integer $\leq n/2$
- Change the new state to $q_{j,0}$ or $q_{j,1}$, depending on whether n was even or odd.

In other words, $\{q_i, m, n\}$ is changed to $\{q_{i,(n \bmod 2)}, 2m+s_i, \lfloor \frac{n}{2} \rfloor\}$. If the machine has moved to the left, we only have to reverse the roles of m and n. Now, how will we construct a tag system out of this interpretation?

Given a binary Turing machine, with states $q_1, ..., q_r$ we define a tag system with symbols $x_i, A_i, \alpha_i, B_i, \beta_i, D_{i,0}, D_{i,1}, d_{i,0}, d_{i,1}, S, s, T_{i,0}, t_{i,1}$, for $i = 1, ..., r$.

The complete description (q_i, m, n) of a Turing machine at a given time will be represented by the following letter string in our tag system:

$$A_i x_i (\alpha_i x_i)^m B_i x_i (\beta_i x_i)^n$$

where the superscripts m and n mean that the bracketed strings are repeated m and n times respectively. The indices determine the state the machine is in. From now on, the state-subscripts, that appear on each letter, will be dropped.

To simulate the operation of moving to the right, the following production rules in the tag system will be sufficient.

The tag words associated with A_i and α_i are:

$$\begin{cases} A \to Cx & (s_i = 0) \\ A \to Cxcx & (s_i = 1) \\ \alpha \to cxcx & \end{cases}$$

Applying these rules to $Ax(\alpha x)^m Bx(\beta x)^n$ we get:

$$Bx(\beta x)^n Cx(cx)^{m'} \tag{6.1}$$

where m' is 2m or 2m + 1. The rules for B and β are:

$$\begin{cases} B \to S \\ \beta \to s \end{cases}$$

Applying these to (6.1), we get:

$$Cx(cx)^{m'}S(s)^{n} \tag{6.2}$$

Then, applying

$$\begin{cases} C \to D_1 D_0 \\ c \to d_1 d_0 \end{cases}$$

to (6.2) we get:

$$S(s)^{n} D_1 D_0 (d_1 d_0)^{m'} \tag{6.3}$$

The rules for S and s are:

$$\begin{cases} S \to T_1 T_0 \\ s \to t_1 t_0 \end{cases}$$

If n is odd, the above given rules result in:

$$D_1 D_0 (d_1 d_0)^{m'} T_1 T_0 (t_1 t_0)^{\frac{n-1}{2}} \tag{6.4}$$

If n on the other hand is even, (6.3) is changed to:

$$D_0 (d_1 d_0)^{m'} T_1 T_0 (t_1 t_0)^{\frac{n}{2}} \tag{6.5}$$

Let us proceed first with the case n odd. The rules for D_1 and d_1 are:

$$\begin{cases} D_1 \to A_1 x_1 \\ d_1 \to \alpha_1 x_1 \end{cases}$$

Applying this to (6.4) we get:

$$T_1 T_0 (t_1 t_0)^{\frac{n-1}{2}} A_1 x_1 (\alpha_1 x_1)^{m'} \tag{6.6}$$

The rules for T_1 and t_1 are:

$$\begin{cases} T_1 \to B_1 x_1 \\ t_1 \to \beta_1 x_1 \end{cases}$$

and result in:

$$A_1 x_1 (\alpha_1 x_1)^{m'} B_1 x_1 (\beta_1 x_1)^{\frac{n-1}{2}} \tag{6.7}$$

from (6.6). Since $\frac{n-1}{2}$ is equal to $\lfloor\frac{n}{2}\rfloor$, we have arrived at the next complete state of the Turing machine, having gone from $\{q_i, m, n\}$ to $\{q_{i,n \bmod 2}, m', \lfloor\frac{n}{2}\rfloor\}$.
Now, let us look at the case n even. The rules for D_0 and d_0 are:

$$\begin{cases} D_0 \to xA_0x_0 \\ d_0 \to \alpha_0 x_0 \end{cases}$$

Applying these to (6.5) we get:

$$T_0(t_1t_0)^{\frac{n}{2}}xA_0x_0(\alpha_0x_0)^{m'} \tag{6.8}$$

Next, we have:

$$\begin{cases} T_0 \to B_0x_0 \\ t_0 \to \beta_0 x_0 \end{cases}$$

resulting in:

$$A_0x_0(\alpha_0x_0)^{m'}B_0x_0(\beta_0x_0)^{\frac{n}{2}} \tag{6.9}$$

From this it follows that also in case n even, we have arrived at the next complete state of the Turing machine, $\{q_i, m, n\}$ being changed to $\{q_{i,n \bmod 2}, m', \lfloor\frac{n}{2}\rfloor\}$.
Both (6.7) and (6.9) have the same form as the string we started from initially, except for the indices being changed. In this respect the production rules given here, are exactly those needed for simulating one execution of the operations that have to be performed in a given state, the Turing machine having moved to the right. The rules for moving to the right, can be applied in a similar way. These are:

$$\begin{cases} A \to S & \alpha \to s \\ B \to Cx & B \to Cxcx(s_i = 0\text{or}1) \\ \beta \to cxcx & \\ S \to T_1T_0 & s \to t_1t_0 \\ C \to D_1D_0 & c \to d_1d_0 \\ T_1 \to A_1x_1 & T_0 \to xA_0x_0 \\ t_1 \to \alpha_1x_1 & t_0 \to \alpha_0x_0 \\ D_1 \to B_1x_1 & D_0 \to B_0x_0 \\ d_1 \to \beta_1x_1 & d_0 \to \beta_0x_0 \end{cases}$$

As is clear from these production rules everything depends on the parity of n (when moving to the right) or m (when moving to the left). This can be clarified with an example. Suppose that we want to simulate the machine going from complete state $\{q_i, m, n\}$ to $\{q_i, 2m+1, \lfloor\frac{n}{2}\rfloor\}$. Further suppose that n = 3 and m = 5 (...0000011x_α101000000...). Then starting from the string, $Ax\alpha x\alpha x\alpha x\alpha x\alpha xBx\beta x\beta x\beta x$, which is the encoding of the content of the tape of the Turing machine, applying the above production rules, we get the following sequence of strings (remember that $v = 2$):

$$
\begin{gathered}
Ax\underbrace{\alpha x\alpha x\alpha x\alpha x\alpha x}_{m=5}Bx\underbrace{\beta x\beta x\beta x}_{n=3}\\
\alpha x\alpha x\alpha x\alpha x\alpha xBx\beta x\beta x\beta xCxcx\\
\alpha x\alpha x\alpha x\alpha xBx\beta x\beta x\beta xCxcxcx\\
\alpha x\alpha x\alpha xBx\beta x\beta x\beta xCxcxcxcxcxcx\\
\alpha x\alpha xBx\beta x\beta x\beta xCxcxcxcxcxcxcxcx\\
\alpha xBx\beta x\beta x\beta xCxcxcxcxcxcxcxcxcxcx\\
Bx\beta x\beta x\beta xCxcxcxcxcxcxcxcxcxcxcxcx\\
\beta x\beta x\beta xCxcxcxcxcxcxcxcxcxcxcxcxS\\
\beta x\beta xCxcxcxcxcxcxcxcxcxcxcxcxSs\\
\beta xCxcxcxcxcxcxcxcxcxcxcxcxSss\\
CxcxcxcxcxcxcxcxcxcxcxcxSsss\\
cxcxcxcxcxcxcxcxcxcxcxSsssD_1D_0\\
cxcxcxcxcxcxcxcxcxcxSsssD_1D_0d_1d_0\\
cxcxcxcxcxcxcxcxcxSsssD_1D_0d_1d_0d_1d_0\\
cxcxcxcxcxcxcxcxSsssD_1D_0d_1d_0d_1d_0d_1d_0\\
cxcxcxcxcxcxcxSsssD_1D_0d_1d_0d_1d_0d_1d_0d_1d_0\\
cxcxcxcxcxcxSsssD_1D_0d_1d_0d_1d_0d_1d_0d_1d_0d_1d_0\\
cxcxcxcxcxSsssD_1D_0d_1d_0d_1d_0d_1d_0d_1d_0d_1d_0d_1d_0\\
cxcxcxcxSsssD_1D_0d_1d_0d_1d_0d_1d_0d_1d_0d_1d_0d_1d_0d_1d_0\\
cxcxcxSsssD_1D_0d_1d_0d_1d_0d_1d_0d_1d_0d_1d_0d_1d_0d_1d_0d_1d_0\\
cxcxSsssD_1D_0d_1d_0d_1d_0d_1d_0d_1d_0d_1d_0d_1d_0d_1d_0d_1d_0d_1d_0\\
cxSsssD_1D_0d_1d_0d_1d_0d_1d_0d_1d_0d_1d_0d_1d_0d_1d_0d_1d_0d_1d_0d_1d_0\\
SsssD_1D_0d_1d_0d_1d_0d_1d_0d_1d_0d_1d_0d_1d_0d_1d_0d_1d_0d_1d_0d_1d_0d_1d_0\\
ssD_1D_0d_1d_0d_1d_0d_1d_0d_1d_0d_1d_0d_1d_0d_1d_0d_1d_0d_1d_0d_1d_0d_1d_0T_1T_0\\
D_1D_0d_1d_0d_1d_0d_1d_0d_1d_0d_1d_0d_1d_0d_1d_0d_1d_0d_1d_0d_1d_0d_1d_0T_1T_0t_1t_0\\
d_1d_0d_1d_0d_1d_0d_1d_0d_1d_0d_1d_0d_1d_0d_1d_0d_1d_0d_1d_0d_1d_0T_1T_0t_1t_0A_1x_1
\end{gathered}
$$

$$d_1 d_0 d_1 d_0 d_1 d_0 d_1 d_0 d_1 d_0 d_1 d_0 d_1 d_0 d_1 d_0 d_1 d_0 d_1 d_0 T_1 T_0 t_1 t_0 A_1 x_1 \alpha_1 x_1$$

$$d_1 d_0 d_1 d_0 d_1 d_0 d_1 d_0 d_1 d_0 d_1 d_0 d_1 d_0 d_1 d_0 d_1 d_0 T_1 T_0 t_1 t_0 A_1 x_1 \alpha_1 x_1 \alpha_1 x_1$$

$$d_1 d_0 d_1 d_0 d_1 d_0 d_1 d_0 d_1 d_0 d_1 d_0 d_1 d_0 d_1 d_0 T_1 T_0 t_1 t_0 A_1 x_1 \alpha_1 x_1 \alpha_1 x_1 \alpha_1 x_1$$

$$d_1 d_0 d_1 d_0 d_1 d_0 d_1 d_0 d_1 d_0 d_1 d_0 d_1 d_0 T_1 T_0 t_1 t_0 A_1 x_1 \alpha_1 x_1 \alpha_1 x_1 \alpha_1 x_1 \alpha_1 x_1$$

$$d_1 d_0 d_1 d_0 d_1 d_0 d_1 d_0 d_1 d_0 d_1 d_0 T_1 T_0 t_1 t_0 A_1 x_1 \alpha_1 x_1 \alpha_1 x_1 \alpha_1 x_1 \alpha_1 x_1 \alpha_1 x_1$$

$$d_1 d_0 d_1 d_0 d_1 d_0 d_1 d_0 d_1 d_0 T_1 T_0 t_1 t_0 A_1 x_1 \alpha_1 x_1 \alpha_1 x_1 \alpha_1 x_1 \alpha_1 x_1 \alpha_1 x_1 \alpha_1 x_1$$

$$d_1 d_0 d_1 d_0 d_1 d_0 d_1 d_0 T_1 T_0 t_1 t_0 A_1 x_1 \alpha_1 x_1 \alpha_1 x_1 \alpha_1 x_1 \alpha_1 x_1 \alpha_1 x_1 \alpha_1 x_1 \alpha_1 x_1$$

$$d_1 d_0 d_1 d_0 d_1 d_0 T_1 T_0 t_1 t_0 A_1 x_1 \alpha_1 x_1 \alpha_1 x_1 \alpha_1 x_1 \alpha_1 x_1 \alpha_1 x_1 \alpha_1 x_1 \alpha_1 x_1 \alpha_1 x_1$$

$$d_1 d_0 d_1 d_0 T_1 T_0 t_1 t_0 A_1 x_1 \alpha_1 x_1 \alpha_1 x_1 \alpha_1 x_1 \alpha_1 x_1 \alpha_1 x_1 \alpha_1 x_1 \alpha_1 x_1 \alpha_1 x_1 \alpha_1 x_1$$

$$d_1 d_0 T_1 T_0 t_1 t_0 A_1 x_1 \alpha_1 x_1 \alpha_1 x_1 \alpha_1 x_1 \alpha_1 x_1 \alpha_1 x_1 \alpha_1 x_1 \alpha_1 x_1 \alpha_1 x_1 \alpha_1 x_1 \alpha_1 x_1$$

$$T_1 T_0 t_1 t_0 A_1 x_1 \alpha_1 x_1 \alpha_1 x_1 \alpha_1 x_1 \alpha_1 x_1 \alpha_1 x_1 \alpha_1 x_1 \alpha_1 x_1 \alpha_1 x_1 \alpha_1 x_1 \alpha_1 x_1 \alpha_1 x_1$$

$$t_1 t_0 A_1 x_1 \alpha_1 x_1 \alpha_1 x_1 \alpha_1 x_1 \alpha_1 x_1 \alpha_1 x_1 \alpha_1 x_1 \alpha_1 x_1 \alpha_1 x_1 \alpha_1 x_1 \alpha_1 x_1 \alpha_1 x_1 B_1 x_1$$

$$A_1 x_1 \underbrace{\alpha_1 x_1 \alpha_1 x_1 \alpha_1 x_1 \alpha_1 x_1 \alpha_1 x_1 \alpha_1 x_1 \alpha_1 x_1 \alpha_1 x_1 \alpha_1 x_1 \alpha_1 x_1 \alpha_1 x_1}_{2m+1=11} B_1 x_1 \underbrace{\beta_1 x_1}_{\frac{n-1}{2}=1}$$

As is clear from the examples, the production rules indeed do the job for n uneven: m is changed to $2m+1$ (the string αx is repeated 11 times now, instead of 5); n is changed to 1 (it is halved): the string of the form βx is repeated only once. Moreover, the tag system has "recognized" that n is uneven (look at the indices). Now let us have a look at an example for which the machine prints a zero, moves to the right, with $n = 4, m = 2$:

$$Ax \underbrace{\alpha x \alpha x}_{m=2} Bx \underbrace{\beta x \beta x \beta x \beta x}_{n=4}$$

$$\alpha x \alpha x B x \beta x \beta x \beta x \beta x C x$$

$$\alpha x B x \beta x \beta x \beta x \beta x C x c x c x$$

$$B x \beta x \beta x \beta x \beta x C x c x c x c x c x$$

$$\beta x \beta x \beta x \beta x C x c x c x c x c x S$$

$$\beta x \beta x \beta x C x c x c x c x c x S s$$

$$\beta x \beta x C x c x c x c x c x S s s$$

$$\beta x C x c x c x c x c x S s s s$$

$$C x c x c x c x c x S s s s s$$

$$c x c x c x c x S s s s s D_1 D_0$$

$$c x c x c x S s s s s D_1 D_0 d_1 d_0$$

$$\begin{array}{l}
cxcxSssssD_1D_0d_1d_0d_1d_0\\
cxSssssD_1D_0d_1d_0d_1d_0d_1d_0\\
SssssD_1D_0d_1d_0d_1d_0d_1d_0d_1d_0\\
sssD_1D_0d_1d_0d_1d_0d_1d_0d_1d_0T_1T_0\\
sD_1D_0d_1d_0d_1d_0d_1d_0d_1d_0T_1T_0t_1t_0\\
D_0d_1d_0d_1d_0d_1d_0d_1d_0T_1T_0t_1t_0t_1t_0\\
d_0d_1d_0d_1d_0d_1d_0T_1T_0t_1t_0t_1t_0xA_0x_0\\
d_0d_1d_0d_1d_0T_1T_0t_1t_0t_1t_0xA_0x_0\alpha_0x_0\\
d_0d_1d_0T_1T_0t_1t_0t_1t_0xA_0x_0\alpha_0x_0\alpha_0x_0\\
d_0T_1T_0t_1t_0t_1t_0xA_0x_0\alpha_0x_0\alpha_0x_0\alpha_0x_0\\
T_0t_1t_0t_1t_0xA_0x_0\alpha_0x_0\alpha_0x_0\alpha_0x_0\alpha_0x_0\\
t_0t_1t_0xA_0x_0\alpha_0x_0\alpha_0x_0\alpha_0x_0\alpha_0x_0B_0x_0\\
t_0xA_0x_0\alpha_0x_0\alpha_0x_0\alpha_0x_0\alpha_0x_0B_0x_0\beta_0x_0\\
A_0x_0\underbrace{\alpha_0x_0\alpha_0x_0\alpha_0x_0\alpha_0x_0}_{2m=4}B_0x_0\underbrace{\beta_0x_0\beta_0x_0}_{\frac{n}{2}=2}
\end{array}$$

This example shows that the rules work for n even too: m is changed to $2m$, n is halved (from 4 to 2), and the index is changed to 0. It has been 'recognized' that n is even. As is clear, the recognition of whether n is even or odd is accomplished through an intelligent use of the shift number: evenness or oddness lead respectively to a kind of de-synchronization and synchronization (this happens the first time with D_1D_0) and a re-synchronization when n is even through the addition of three symbols instead of two ($D_0 \rightarrow xA_0x_0$).

While this proof by Minsky is less complicated and in a way more elegant than his first proof of the unsolvability of tag systems, it might perhaps already be clear that Turing machine simulating tag systems are not quite practical: 2-symbolic Turing machines with m states and 2 symbols can be reduced to a tag system with $v = 2$ $\mu = 16m$, i.e., to still large tag systems. This will be discussed in more detail in Chapter 12.

Decidability criteria in tag systems

To prove that the whole class of tag systems has an unsolvable decision problem has been an important result – Post probably would have been relieved to hear that the intractability he had experienced when working with tag systems is actually inherent to these systems. This of course does not mean that every tag system has an unsolvable decision problem. To exclude specific classes of tag systems that have an unsolvable decision problem, some mathematicians have proved the existence of criteria that can mark the difference between solvability and unsolvability. [8]

In his *Account of an anticipation*, Post mentions some results concerning solvable cases of tag systems, thus determining a kind of lower bound for unsolvability with respect to the number of symbols and v, which is still the best known so far. Anything below this limit is known to be solvable. After having discussed the different types of behaviour, Post writes ([Pos65], p. 362):

> [...] the problem of "tag" was made the major project of the writer's tenure of a Procter fellowship in mathematics at Princeton during the Academic year 1920–1921. Indeed, the reduction of the last section, effected early in that year, sealed this determination. And the major success of that project was the complete solution of the problem for all bases in which μ and v were both 2.

In the footnote to this quote he added:

> When either μ or v is 1 the problem becomes trivial. By contrast, even this special case $\mu = v = 2$ involved considerable labor.

Post thus proved the following theorem:

Theorem 6.1.1 *For any given tag system T, if $\mu = 1$ or $v = 1$ or $\mu = v = 2$ then the two forms of the decision problem for T are solvable.*

The proof of the theorem however, was never published.[9] It is indeed a fact that the case for which $\mu = 1$ is very trivial. Clearly, the word w_{a_0} corresponding to this one symbol a_0, will be a concatenation of $l_{w_{a_0}}$ times a_0. Then, if $l_{w_{a_0}} < v$

[8] More will be said about such criteria in Chapter 7, where a more exact definition of such criteria will be given, following Margenstern [Mar00].

[9] This result or notes relating to this result might still be present in the archive of Emil Post in Philadelphia (American Philosophical Society). The present author plans a visit to this archive for exactly this reason.

the tag system will halt, whatever the initial condition might be, it will become periodic when $v = l_{w_{a_0}}$ and show unbounded growth when $l_{w_{a_0}} > v$. The case with $v = 1$ seems not that trivial, since Wang took it seriously enough to present it in his [Wan3a]. The case $v = \mu = 2$, will be proven to be solvable in chapter 9. As will become clear then, the proof indeed involves "considerable labor".

Both μ and v can be regarded as a *decidability criteria* [Mar00] for tag systems, since their solvability depends on the size of these parameters. The number of symbols μ is a decidability criterion since tag systems with $\mu = 1$ are solvable, while there also exist tag systems with a given number of symbols $\mu > 1$ that are unsolvable.[10] Cocke and Minsky proved that any Turing machine can be simulated by a tag system for which $v = 2$ (See [CM63], [Min62]). Maslov generalized this result and proved that for any $v > 1$ there exists at least one tag system with an unsolvable decision problem and, independent of Wang, proved that any tag system for which $v = 1$ is solvable [Mas4b].

Another such criterion for tag systems, is the length of the words. Let l_{min} denote the length of the smallest word of a tag system and l_{max} the length of the lengthiest word. Pager studied classes of tag systems with shift number v, l_{min} and l_{max}, that contain tag systems with an unsolvable decision problem. Wang proved that any tag system for which $l_{min} \geq v$ or $l_{max} \leq v$ is solvable [Wan3a]. It should be added here that Maslov proved that the tag systems with an unsolvable decision problem that can be constructed using his method, for each $v > 1$ all satisfy the following condition: $l_{min} = v-1$, $l_{max} = v+1$ [Mas4b]. Taking into account Wang's result, he describes this condition as a kind of minimal condition for unsolvability in tag systems. This result was independently proven by Wang for a tag system with $v = 2$ [Wan3a].

Except for Post, nobody takes into account the number of symbols μ, determining the number of production rules, in order to study solvable and unsolvable classes of tag systems.[11] Still, it is clear that the significance of μ for determin-

[10] The exact value for μ to mark the difference between solvability and unsolvability will be further discussed in Chapter 9, where we will tackle the question of the unsolvability of tag systems with $\mu = 2$

[11] In the chapters to follow the significance of μ will be made more explicit. It should also be noted that besides including μ to determine solvable classes of tag systems, Post also explicitly

ing solvable and unsolvable classes of tag systems cannot be underestimated and can in fact be used as a decidability criterium, which is independent of v. It is thus only natural to include μ in the definition of a measure for the size of tag systems. In this respect we would like to propose the following measure for the size of tag systems:

Definition 6.1.1 *The size of a tag system is defined as the product of μ and v, where TS(μ, v) denotes the class of tag systems with μ symbols and a shift number v.*

The length of the words is not taken into account, since the decidability criterion with respect to l_{min} and l_{max} is defined relative to v.

Decision problems for tag systems and degrees of unsolvability

Some more results on tag systems, which are only mentioned here for completeness, concern degrees of unsolvability in tag systems and the proof of the unsolvability of yet another decision problem for tag systems.

In 1966 it was proven by Philip Hooper that the immortality problem for 2-symbolic Turing machines is recursively unsolvable [Hoo6a]. What was previously called a *complete state* of a Turing machine T, is called the *instantaneous description* (ID) of T by Hooper. If an ID has a corresponding quintuple – describing the internal state of the machine and the operations that have to be performed when the machine enters this state – the result of applying it is another ID. If this is not the case, the ID is called terminal. Those ID's that ultimately lead to a terminal ID are called *mortal* ID's, those that do not are called *immortal.* The immortality problem now is the problem to decide for a given T whether or not there exists an immortal ID. After having proven that the immortality problem is indeed unsolvable for Turing machines, Hooper added a whole list of related problems and results, one of them being the proof of the unsolvability of polygenic Post normal systems, through reduction of the tiling

refers to μ in describing the behaviour of other classes of tag systems. I.e. the case with $\mu = 2$, $v > 2$ is called intractable, while he terms the cases $\mu > 2, v = 2$ as being of "*bewildering complexity*". See Sec. 2.2.5, p. 51.

problem to polygenic normal systems.[12] A polygenic normal system is the opposite of a monogenic system. A monogenic normal system is a system for which one can never apply more than one production on a given string. Furthermore he mentioned the unsolvability of the immortality problem for tag systems and monogenic systems in normal form as an open problem.

Some years later, in 1971 Stal Aanderaa and Dag Belsnes published a paper [AB71] called *Decision problems for tag systems*, in which the authors prove two results for tag systems. First of all they settled the questions posed by Hooper concerning the immortality problem for tag systems and monogenic systems in normal form, by proving that the immortality problem for tag systems is recursively unsolvable of degree **0”**. Furthermore they proved that for each recursively enumerable degree **d**, there is a tag system whose halting problem is of degree **d**.[13]

While this field of recursion theory is of course a very fascinating one, Post's [Pos44] being one of the founding papers, it will not be considered here in any details.

6.1.2 “Tag – you are it”: Some concrete research on tag systems.

Using the computer with tag systems.

In his introduction to *Theory of finite and infinite machines* [Min67], Marvin Minsky spends several pages on tag systems, including his less exponential construction of a universal tag system. In discussing the only tag system mentioned by Post ($v = 3, 1 \rightarrow 1101, 0 \rightarrow 00$) Minsky remarks ([Min67], pp. 267–268):

> Post found this (00, 1101) problem “intractable”, and so did I, even with the help of a computer. Of course unless one has a theory, one cannot expect much help from a computer (unless *it* has a theory) except for clerical aids in studying examples; but if the reader tries to study the behaviour of 100100100100100100 without such aid, he will be sorry.

[12] The proof was announced in [Hoo65], and proven in [Hoo6b].

[13] A more formal proof of this result is given in [Ove71]. The result was generalized to the immortality problem in [Hug73].

This quote is actually a perfect characterization of how one feels after having worked some time on this exemplar of tag system: you can't seem to do without a computer. It makes one respect even more Post's research on tag systems and illustrates that his study of tag systems had to involve certain "experimental" work.[14]

The example mentioned by Post is a remarkable example of a very small system, with only two production rules, giving rise to complex behaviour. Despite its simplicity, it is still not known whether this tag system is solvable or not. By formulating a kind of statistical argument one would think that the system would always come to a halt or would become periodic after some time. As Minsky remarks ([Min67], p. 270):

> [...] Post was interested in the decidability of the question: Does the reading head, which is advancing at the constant rate of *P* squares per unit time, ever catch up with the write head, which is advancing irregularly. [...] Note, in the (00, 1101) problem, that the read head advances three units at each step, while the write head advances by two or four units. Statistically, one can see, the latter has the same average speed as the former. Therefore, one would expect the string to vanish, or become periodic. One would suppose this for most initial strings, because if the chances are equal of getting longer or shorter, then it is almost certain to get short, from time to time. Each time the string gets short, there is a significant chance of repeating a previously written string, and repeating once means repeating forever, in a monogenic process. Is there an initial string that grows forever, in spite of this statistical obstacle? No one knows. All the strings I have studied (by computer) either became periodic or vanished, but some only after many millions of iterations!

Indeed, given the fact that the total number of 1's and 0's in the two words to be tagged (w_0 and w_1) are equal, one would expect that the system will always come to a halt or become periodic after a certain number of steps. This so-called "statistical obstacle" presupposes, however, that the *relevant letters* have

[14] As was explained in Sec. 2.2.5, the double quotes surrounding "experimental" have a clear intention, and indicate that "experimental" should be understood here in a specific way.

a random distribution.

> The relevant letters are those letters in a given string that will be scanned by the tag system, i.e. every $3n+1$-th letter in the case of Post's tag system. For example, in the following string,
>
> **1**01**0**11**0**10**1**01**0**10**1**00**1**01**0**10**1**01**0**10**1**01**0**10**1**10**1**10**1**10
>
> the bold letters, are the ones that really matter, the rest of them are erased as a consequence of the shift number.

Also Brian Hayes understands that this kind of reasoning is "tempting". After having described the same kind of argument, he compares this tag process with a random walk ([Hay86], p. 22):

> The variations in the length of the string should describe a one-dimensional random walk centered on the average length. Any one-dimensional random walk, if it is continued long enough, can be expected to visit all the sites available to it; in this case the string should at some point become arbitrarily short, drop below the three-digit treshold, and dwindle away. (The blind man, stumbling about at random on top of the cliff, eventually falls over the precipice.) Even before that happens, the system may light upon a pattern that leads into a cycle. The problem with this analysis is that the pattern of 0s and 1s is not random; on the contrary, it is generated by a fully deterministic process. Moreover, even if most of the strings have the statistical properties of random patterns, there may well be exceptional strings that behave very differently. At best, a probabilistic argument can predict the total outcome, but what is of greatest interest is the singular case.

Neither Minsky nor Hayes seem to be completely convinced by this kind of statistical argument. While Minsky does not give very clear reasons why this should be the case, Hayes does. He is completely right in pointing out the fallacy in the argument by saying that the sequence of letters formed by the relevant letters is not necessarily random. However, his reasons for saying so, are not necessarily correct. First of all, randomness can at least be called a problematical concept. Besides the fact that there are different usages of the notion,[15] one is also confronted with the fact pointed out by Marsaglia – the developer of DIEHARD, a battery of random tests – that deterministic random number generators usually do better as compared to physical devices according to statistical tests for randomness. In other words, the first argument by Hayes is problematic. As far as the second argument is concerned, one should remark that one does not necessarily need "exceptional strings" in order for this sequence of relevant letters not to be random. In chapter 8 this argument will be further explored through some experimental results.

Hayes's paper mainly focusses on Post's exemplar of a tag system. Besides the description of an algorithm to run tag systems, Hayes raises several questions based on his observations of the output of several "computer experiments". In a first set-up he tested 50 initial conditions of length 24. Starting from an arbitrary 24-bit string, the 49 consecutive strings were tested measuring the number of steps it takes for each string to halt or enter a period. For those strings that became periodic he also measured the length of the period and the length of the first repeated string. He found that about 2/3 of these strings become periodic, the rest halting after no more than 400 steps. Most of those that become periodic, did so after 100 steps, although there are exceptions. Furthermore about 50% of the strings that become periodic have a period 6, while about 30% has a period 10. In looking at a larger sample of strings, 75% of the strings become periodic, the rest halt. Based on this observation, Brain Hayes poses the question of whether, as the initial strings get longer, the proportion of strings that become periodic relative to the strings that vanish, grows. The findings concerning the dominance of period 6 and 10 were affirmed by the larger sample, Hayes consequently asking the question of whether there is an explanation for this dominance. A further observation is the fact that the length of the first string repeated is always an odd number. At first Hayes thought that this observation might lead to an explanation. However, in another sample of 1000 intial conditions, the conjecture was contradicted. A further conjecture was the idea

[15] Compare for example the definition of randomness in algorithmic information theory, with the way Stephen Wolfram uses this notion.

that as the number of relevant 1's increases, the average number of steps before periodicity is entered would also increase, given the fact that if a 1 is scanned, the length of the string increases. But after having tested a sample of all first 50 strings for which all the relevant letters are 1's, he concluded that, while there are clearly some long runs, it is clear from the data that neither the length of the initial string, nor the number of significant 1's can be used as reliable predictors of run length.

As is clear from these results, one must be very careful in making conclusions based on data from computer experiments as far as tag systems are concerned. Of course, one can always make conjectures, but one should always perform more tests before being less critical about the conjecture. E.g. some of Hayes's initial conjectures were falsified by the systems themselves while testing the conjecture ([Hay86], pp. 26–27):

> In much of life, perhaps, the exception proves the rule, but in mathematics a law valid in 99 and 44/100% of the cases is an abomination.

Hayes ask two further questions. First of all, given the intractable character of Post's tag system, and the difficulty of getting a grip on it even "experimentally" Hayes poses the question of whether there might exist a connection between the $3n+1$-problem and tag systems.

The $3n+1$-problem is the following: Given an arbitrary number. If it is even, divide it by 2, if not, multiply by 3 and add 1. The same procedure is applied to any next number produced this way. The question now is, whether for every number, this iterative process ultimately leads to 1 and consequently periodicity. This problem is still open, although most believe that the mapping will always lead to a cycle, and is one of those mathematical problems for which the computer has become an indispensable instrument. The $3n+1$- problem will be discussed in more detail in the last chapter 9, linking it both to the Busy Beaver game and tag systems. In fact, it will be shown that the answer to Hayes' question is affirmative.

Another question posed by Hayes concerns the periods he found in Post's tag system, viz.: 2, 4, 6, 8, 10, 12, 16, 28, 40, 52. Based on this piece of data, Hayes wondered whether the intervening even numbers in this sequence will ever turn up. In a column published 10 years later in the American Scientist, Hayes further lingered upon this question [Hay6a]. Hayes now submitted the sequence of periods he found to Sloane's sequences@research.att.com.

Sloane has made a wonderful instrument available on the internet, called *The On-Line Encyclopedia of Integer Sequences*. In this encyclopedia millions of integer sequences can be looked up. If there is a match, you immediately get a list of functions which result in the same sequence, or another sequence of which the one submitted is merely a subset. One of the interesting aspects of this encyclopedia is that it can function as a kind of bridge between all the specialized fields of mathematics. While it may be the case that the mathematician working in one field cannot understand the work of the mathematician working in another field, they have one thing in common: numbers. Indeed, whatever field you are working in, numbers are always present (explicitly or implicitly). In submitting a sequence of integers to the encyclopedia, one sees that it can happen that one sequence pops up in a variety of domains, thus linking them up.

The answer of the algorithm behind the e-mail address Hayes submitted his sequence to was a "no match". There is, however, also a more sophisticated list of algorithms developed by Sloane, called *Superseeker*. It can be addressed though the following address: superseeker@research.att.com. After Superseeker examined the sequence submitted by Hayes, "it" concluded that the last four terms are separated by three equal intervals of 12, which led Superseeker to the prediction that the four following terms should be 64, 76, 88, 100. Hayes went back to his computer and now tested 650000 different initial conditions which resulted in the following sequence of periods: 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 32, 34, 36, 40, 46, 52, 56, 282. Superseeker now suggested three different generating functions, all resulting in the even numbers. Hayes wondered in his column whether Post's tag system is indeed able to generate all even numbers, or whether the sequence is just a finite and arbitrary collection of numbers. The problem was settled by James B. Shearer in a letter to the editors [She96]. He showed that Post's tag system can indeed generate all even numbers, the proof being very straightforward. Let $A = 001101$, and $B = 110111010000$. It is clear that A has a period 2, while B has a period 4. Then it follows that any even number $n > 4$ is generated by $A^{(n-4)/2}B$. For example, a period 14 is generated by $(001101)^5 110111010000$.

Hayes's research on tag systems is a very good example of how research on tag systems can be like: Doing computer experiments with tag systems often leads to a variety of questions, that can be specified by performing yet more experiments. Sometimes one gets a clearer answer, making it possible to formulate a conjecture. More often however, doing more experiments only leads to more puzzling questions.[16]

Periodicity in Post's tag system: an attempt to get formal control on tag systems.

The paper that seemed – to the author – the most promising as far as hard mathematical facts and tag systems are concerned is Watanabe's [Wat63], called *Periodicity of Post's normal Process of Tag*, published in 1963. It was presented at the *Symposium on Mathematical Theory* where many other important mathematicians and computer scientists gave a talk, including Martin Davis, Tibor Rádo, John McCarthy and Hao Wang.

It was only by the end of August 2006 that the author was able to read this paper, and our expectancies were high. However, although the first pages seemed at first sight rather promising, a first "scroll" through the paper showed that what-

[16] There is one researcher whose "work" on tag systems should be mentioned here in a footnote: Stephen Wolfram. In his rather arrogant *A new kind of science* Wolfram [Wol02] discusses several classes of computational systems in one of the first chapters of his book, including "tag systems", in order to add strength to his conjecture – which flows like a kind of mantra throughout the whole book – that small systems not only give rise to complex behaviour, but that the threshold for universality is very low. While there is actually nothing more to be said about the few pages Wolfram spends on "tag systems", we mention it, to point out an error. One of the examples Wolfram gives of a "tag system" leading to complex behaviour, is the following: $v = 2, 11 \rightarrow 100, 10 \rightarrow 100, 01 \rightarrow 0, 00 \rightarrow 01$. As should be clear from this set of rules, this system is not a tag system, but is a member of the more general class of systems tag systems belong to i.e. monogenic systems in normal form.

ever the proofs might prove, there had to be an error somewhere. To explain why this is the case, we have to explain a bit more about periods in Post's tag systems. As was said, this tag system is able to produce all the even numbers. In Chapter 8 we will study in more detail the several possible periodic structures in several tag systems, including Post's. The "structure" of a periodic string should be understood here as the sequence of letters formed by the relevant letters in a periodic sequence. For example, given the following periodic string (period 6):

101**1**10**1**11**0**10**0**00**0**0

Its structure, formed by the bold letters is:

111000

After having tested thousands of initial conditions of Post's tag system, we found 4 basic structures, all the other structures (except for two) being compositions of these structures. These basic structures are:[17]

$$\begin{cases} 10 & \text{Period} = 2 \quad [2] \\ 1100 & \text{Period} = 4 \quad [4] \\ 111000 & \text{Period} = 6 \quad [6] \\ 1100011100 & \text{Period} = 10 \quad [10] \end{cases}$$

An example of a composed structure is given by the following string, which has a period 30:

101**1**10**1**11**0**10**0**00**0**01**1**01**0**01**1**01**1**10**1**11**0**10**0**00**0**01**1**0
1**1**10**1**11**0**10**0**00**0**01**1**01**1**10**1**11**0**10**0**01**1**01**1**11**0**10**0**00**0**0

Assembling the relevant letters we get:

$$111000101110001110001110011000 = 3 \cdot [6] + 1 \cdot [2] + 1 \cdot [10] = [30]$$

While the majority of periodic strings analyzed this way always have a composite form, or one of the basic forms, two exceptions to this rule were found, viz.

[17] Of course these are just one possible form of each of the structures. E.g. 001110 is a possible variant of the structure of period 6.

a period 40 and a period 66. While their structure contains some of the basic structures, it is not composite. Furthermore, contrary to the composite periods, the length of the structure does not equal the length of the period. The two exceptions found (until now) are:[18]

$$\begin{cases} \text{000001101011100} & \text{Period} = 40 \quad [40] \\ \text{111101001101000100011111} & \text{Period} = 66 \quad [66] \end{cases}$$

Now, what is the significance of these findings for Watanabe's paper? The paper was written with the intention of getting a better understanding or even explanation for the periodic structures in Post's tag system – as should be clear from the title. At the end of the paper, Watanabe concludes, on the basis of *formal* arguments, that the following periodic strings (or any variant form of them):

(1) **0**00**0**11**0**11**1**01**1**10**1**(00**0**00**0**11**0**11**1**01**1**10**1**)^n

(2) **0**01**1**01

(3) **1**10**1**11**0**10**0**00

(4) Any concatenation of (2) and (3)

are the only periodic strings possible in Post's tag system ([Wat63], p. 97):

> We can [...] conclude that any periodic string is only ab, b^2a^2 or a concatenation of ab's and b^2a^2, or $a^2b^3(a^3b^3)^n$

where a = 1101, b = 00. These strings, are in fact three of the basic periodic structures mentioned above, namely those for period 2, 4 and 6. As should be clear by now, however, there must be something wrong here, since neither [10] nor [40] and [66] are concatenations of this form.

First of all, on the basis of the proofs, Watanabe excludes the possibility of periodic strings being a concatenation of (1) with (2) and/or (3). This cannot be true. The composite structure of a string with period 30 (given above), contains both (1) and (2) and was effectively produced by Post's tag system. Secondly,

[18] Of course this is just one form of the structure

the fourth basic periodic structure found, period 10, is not included. Furthermore, the “special” structures found for strings with a period 40 or 66 are not considered. So, what is wrong here? There are several mistakes in the proof, but we will only discuss one more fundamental mistake.

At some points Watanabe presupposes certain properties for periodicity. These assumptions exclude the possibility of structures such as that for the period 40 string. On p. 92 he claims to have proven the following “theorem”:

> The string L = PQ defined above always produces the same form [QP] as considered as a circular permutation. We call L a loop.

Now, the string PQ is defined as consisting of a periodic string (P) and some other string (Q), the length of PQ being divisible by 3. In defining a certain, rather strange, operation[19] – without proving that this operation “works” in tag systems – Watanabe proves that when we have PQ it will always be converted into QP, after applying another operation defined earlier in the paper, indicated by a short arrow. This short arrow, basically means that a string X is produced from a string Y after all the relevant letters of X have been processed, where X has a length divisible by $v = 3$. As should be clear, scanning all the relevant letters of PQ, with the length of PQ divisible by 3, and P periodic, can never result in QP since after this is done, the first string in whatever string results from PQ after application of the short arrow operation, must be P, if periods are considered such as the period 4 and 6 in Post’s tag system, which are the only periodic strings considered by Watanabe. Thus, there must be something wrong here. It would make more sense if the short arrow is – in the context of the proof – understood as the result of scanning the relevant letters of P. Then it is indeed possible for PQ to produce QP. E.g. if P has length 6, PQ will be converted into QP after two iterations in Post’s tag system.

This is indeed the case if P is a periodic string of the types considered by Watan-

[19] This operation, indicated as “ is defined as follows: $A\text{“}W_iB \rightarrow AW_i\text{“}BX_i$, where W_i is a string of a given form, and X_i a specific kind of string produced by W_i. Watanabe never shows that it is possible to define this kind of operation for Post’s tag system for any string in this form. This kind of introduction of special operations and special derivations is typical for Watanabe’s whole proof, and one of its weakest points.

abe, or is it not? Let us look at an example:

$$\underbrace{001101}_{P}\underbrace{000000}_{Q}$$

Applying the production rules of Post's tag system to this string for two times, having gone through all the significant letters in P, leads to the following string:

$$\underbrace{000000}_{Q}\underbrace{001101}_{P}$$

As far as this example is concerned, everything is OK. However what about the following example, using a period 6?

$$\underbrace{101110111010000000}_{P'}\underbrace{1101}_{Q'}$$

After 6 iterations, having gone through all the relevant letters in P, we get the following string:

$$\underbrace{1011}_{\neq Q}\underbrace{101110111010000000}_{P'}$$

As is clear from this example, it is not always the case that $PQ \to QP$, if $PQ \equiv 0 \bmod 3$, with P periodic. Now, based on the "proof" that PQ will always be converted into QP after all the significant letters of P have been processed, taking into account the conditions on P and PQ, Watanabe naturally concludes that if both P and Q are periodic the following symmetry is achieved:

$$PQ \to QP \to PQ \to ....$$

It is this "fact", together with several other "facts", that finally lead him to the proof that there are only 3 basic periods (together with the concatenations of (2) and (3)) possible in Post's tag system. Besides the fact that the proof of the convertibility from PQ into QP is wrong, taking into account the conditions imposed on PQ and P Watanabe does not see the possibility of a structure such as that of a period 40. This is the case, because he presupposes that any periodic string is of the form PQ as described above, with both P and Q periodic. This leads him to the further assumption that *if* a string is periodic, it must reproduce itself after all its significant letters have been scanned. This is indeed the case for the majority of the periods produced in Post's tag system. E.g.:

$$001101 \rightarrow 10100 \rightarrow 001101$$

However, in the case of the period 40 given above this *cannot* be the case. If all the relevant letters of a string with this kind of periodic structure are "processed" (15 steps in the case of the structure given above) the resulting string is not the same as the one started from. The same reasoning goes for the structure of the period 66 string mentioned above. It is in making certain assumptions about periodicity, possibly linked to a theoretical knowledge of periodicity, that first of all, the basic proof on loops is wrong, and that, secondly, the possibility of having a periodic string, which, in a way, exceeds its own length, is not taken into account.

In general, although we have enjoyed reading Watanabe's paper in a certain way, it is clear that what he proves is contradicted by Post's tag system. Some comments are in place here. First of all, we have the impression that Watanabe has started from a too limited set of observations of Post's tag systems, and, on the basis of these observations, constructed a proof that is not correct. Secondly, as was shown in the analysis of one subproof, he starts from certain assumptions that have not been proven. Thirdly, the whole proof is characterized by a kind of reversal of the normal structure of a proof. I.e., Watanabe's lemma's are often of the form "if this kind of periodic structure occurs, the involved strings are of this particular form", however, when trying to follow his proofs, it appears that what actually is the case, is the opposite: "if the involved string is of this particular form, this kind of periodic structure occurs". Although this is not a problem in itself, it does obscure the actual assumptions of the proof. Finally, as a consequence of these previous points, the proofs themselves contain basic errors. For example, without going into the details, the proof starts by constructing words of several orders, 00 and 1101 are for example words of the first step, derived from substrings 0– and 1–. He considers words of the first, second and third step, and derived words for each of these steps. Basic to the proof is that words of the first step can always be "rewritten" as words of the second step, and that words of the second step can always be "rewritten" as words of the third step, where both the words themselves as well as the derived words have a length divisible by v. The problem with the proof is that nothing guarantees that this will actually happen. Furthermore, Watanabe does not take into account, that the reverse might happen, words of the third step e.g. being "rewritten" as words of the second step.

Summarizing, Watanabe's proof is too much directed towards a given goal, proving that Post's tag system allows but a certain kind of periodic strings, and, as a consequence, neglects certain aspects of Post's tag system, and tag systems in general.

Despite our critiques towards Watanabe's paper, we still believe that it is a very interesting paper. It is the only paper I know of that tries to find a more formal approach towards specific cases of tag systems. Furthermore, Watanabe's failure to get a more formal grip on tag systems, only shows that even this simple case is far from trivial.

6.1.3 Conclusion

In this section we have seen that there are some basic results on tag systems, the most important being Minsky's proof of the general unsolvability of tag systems. Still, the number of results remains limited. From the previous, it is also clear that once researchers focus on more specific cases of tag systems, there arise more problems than there are solutions. Both Minsky, Hayes as well as Watanabe concluded for the intractability of Post's tag system. This serves as an indication of the difficulties even such simple tag system gives rise to.

In the next section we will take a closer look at the several classes of behaviour one can differentiate in tag systems, and connect these classes to the two forms of the problem of tag (as was done by Post).

6.2 General classes of behaviour in tag systems

In Post's description of his research on tag systems, he discusses the relation between solvable and unsolvable classes of tag systems and the general classes of behaviour for tag systems: termination, periodicity and unbounded growth. It is the purpose of this section to discuss these general classes and their link with unsolvable decision problems in tag systems. In this way, we want to give the reader yet another impression of the behaviour of tag systems.

6.2.1 Description of classes of behaviour

There are two global classes of behaviour for tag systems: *termination* and *non-termination.* A tag system terminates, when the empty string ϵ is produced. In Fig. 6.1, the following tag system:

$$\begin{cases} v = 3 \\ 0 \to 0 \\ 1 \to 0110 \end{cases}$$

was started with a random initial condition of length 200. After about 700 steps it terminates.

If a tag system does not terminate, the resulting infinite sequence of strings produced either converges or diverges, i.e. either the tag system becomes periodic or grows without bound. In Fig. 6.2 an example is given of a tag system that becomes periodic. The tag system used here is:

$$\begin{cases} v = 4 \\ 0 \to 01 \\ 1 \to 011010 \end{cases}$$

The period of a tag system is determined by the number of iteration steps needed for a string to reproduce itself. In the example, the period = 2. The string:

0101101001011010010110100101101001011010010110100
1011010010110100101101001011010010110100101101001
0110100101101001011010010110100101101001011010010
11010010110100101101001011010010110100101101001011010

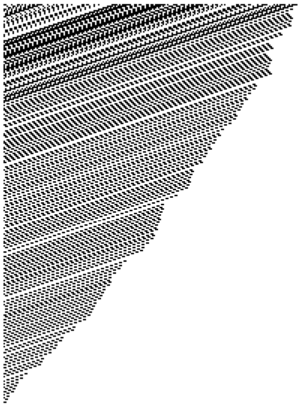


Figure 6.1: Example of a tag system that terminates.

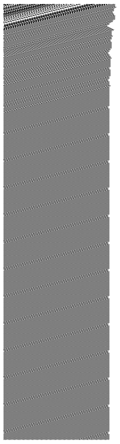


Figure 6.2: Example of a tag system that becomes periodic.

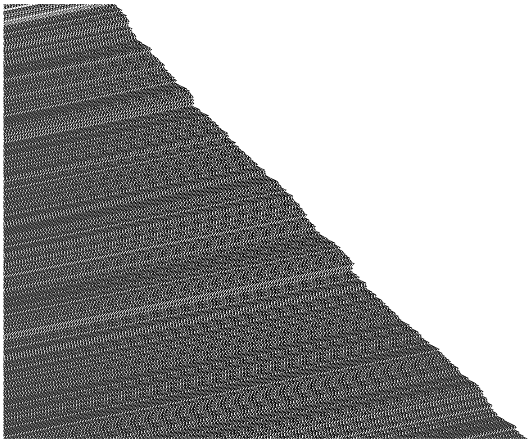


Figure 6.3: Example of a tag system that grows ad infinitum

being repeated after two steps, when applying the production rules. A tag system becomes periodic if any string appears more than once in the evolution of the system. This is due to the fact that tag systems are monogenic: in applying the iterative tag process to a given string, one and only one string can be produced. Consequently if a sequence of strings $A \rightarrow B \rightarrow ... \rightarrow A$ is produced, the second time A occurs it must enter the same cycle.

Besides periodicity and termination, a tag system can also grow without the lengths of the strings produced being bounded. In Fig.6.3 an example is given of a tag system showing this behaviour. The tag system used here is:

$$\begin{cases} v = 5 \\ 0 \rightarrow 1001 \\ 1 \rightarrow 0110111 \end{cases}$$

For this specific example, it can be shown that the tag system will keep on growing i.e. it can be proven that whatever initial condition it is started with, except for the initial condition 0, for any number n the tag system will produce a string S_i of length $l_{S_i} > n$ after a finite number of iterations i, such that no string S_j, $j > i$, will ever be produced again for which $l_{S_j} \leq n$. In other words, the tag system will show unbounded growth.

In looking at the examples of termination, periodicity and unbounded growth one might begin to wonder as to why adjectives such as “uncontrollable” and “intractable” were used in relation to tag systems. As far as the examples given above are concerned, it is indeed all rather simple. The tag systems were constructed such that I knew in advance what kind of behaviour they would give display. So what kind of behaviour is it precisely that characterizes the more difficult tag systems?

6.2.2 “Unpredictable iterations.”

In Fig. 6.4 an example is shown of what we call *fluctuating behaviour*.[20] The tag system used is that mentioned by Post:

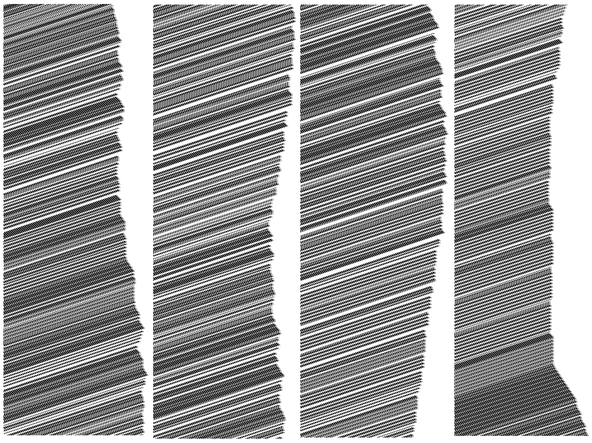


Figure 6.4: Example of fluctuating behaviour in Post’s tag system.

$$\begin{cases} v = 3 \\ 0 \rightarrow 00 \\ 1 \rightarrow 1101 \end{cases}$$

As can be seen, there seems to be no indication whatsoever in this kind of behaviour to predict what will happen in the end. Will this process terminate,

[20] The title of this subsection, *Unpredictable Iterations* is the title of a paper by Conway [Con72] in which he gave a proof of the unsolvability of Collatz-like problem, cfr. Sec. 9.3.2.

become periodic or lead to unbounded growth? It is a typical feature of "fluctuating behaviour" that there seems to be no regularity in the way the lengths of the strings evolve with the number of iterations.

As was already discussed in the previous chapter, for this specific tag system, one would at first expect that despite this fluctuating behaviour, the process will in the end become periodic or terminate, presupposing that the distribution of relevant letters is statistically random. For the tag process shown in Fig. 6.4 this is indeed the case. It will become periodic after about 16500 iterations as is shown in Fig. 6.2.2. However, neither the rules of the tag system nor the first, say, 10000 iterations of the tag process (See Fig. 6.4) seem to give us any clue that this will in fact happen. In general, until now, no one has found a method to predict the behaviour of Post's tag system, when started with an arbitrary initial condition. All one seems to be able to do is to wait and see. It is exactly this kind of behaviour that characterizes the more difficult tag systems, such as the one mentioned by Post. Whenever we find a tag system that seems to show this kind of behaviour, such that there seems to be no clear way to predict the behaviour of the tag system, we will call this tag system *intractable*, allowing for the possibility that one or both forms of the problem of tag are un-

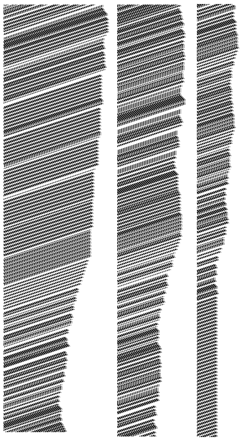


Figure 6.5: Further evolution of the tag process from from Fig. 6.4, leading to periodicity.

solvable for the specific tag system considered, and thus also allowing for the possibility of universality.

6.2.3 Classes of behaviour and the two forms of the problem of "tag".

For Post [Pos65], the two forms of the problem of tag and the general classes of behaviour were clearly connected questions. The problem in its first form is the problem to determine for a given tag system T, when given an arbitrary initial condition I, whether it will terminate yes or no. In the second form of the problem, the initial condition is considered part of the description of the tag system. Then the problem is to determine for any string A whether a given tag system T will ever produce A. How can these problems become solvable?
As far as the first problem is concerned, the problem is solvable for a given (class of) tag systems if it can be determined for every possible string over the alphabet, from the recursively enumerable list of all possible strings, whether it belongs to the class of terminating initial conditions or to the class of non-terminating initial conditions for the (class of) tag systems considered. In the second form, the problem is immediately solved if it can be shown that the tag system will terminate. If the tag process does not terminate, we must differentiate between periodic or non-periodic behaviour. If one is able to show that the lengths of the strings produced will remain bounded – they will never become longer than a given number n – then the system will become periodic (or halt) and one thus merely has to compare the string for which one wants to know whether it will be produced by the tag system, with a finite number of strings. If this is not the case, the problem is solved if one can find a method to show that the tag process will show unbounded growth, i.e. [Pos65], p. 362: "*if a method were also found for determining for any given length of sequence a point in the process beyond which all derived sequences were of length greater than that given length.*" (cfr. the example of unbounded growth). Indeed, if you know in advance, for every possible length n, how many iteration steps i it takes before the tag system will never again produce a string S_j, $j > i$, with length $l_{S_j} \leq n$ the problem in its second form can be solved.

In the remainder of this text, the first form of the problem of tag will also be identified as the *halting problem* for tag systems, the problem in its second form will be called the *reachability problem* for tag systems.

With respect to the solvability of the two forms of the problem of tag, it is interesting to have a closer look at Minsky's small 4-symbols, 7-state universal Turing machine (cfr. in Ch. 5). This machine was proven to be universal by showing that it can simulate every tag system with $v = 2$. Since Minsky had already shown that there exist tag systems with $v = 2$ that can represent any Turing machine, the small universal Turing machine constructed by Minsky is indeed universal.
The tape of this universal machine is subdivided into three regions:

...	0	0	**Rules Region**	**Erased Region**	**Sequence Region**	0	0	...

The rules of the tag system to be simulated are represented in the rules region of the tape. Every time the simulation of an iteration step has been finished, the encoding of the production rules will be restored in its original form. The sequence region is used to encode the actual productions of the tag system, and are represented by an arrangement of X's and B's.[21] For example, encoding the words of the tag system mentioned by Post, a 1 will be encoded as XXXXB and 0 as XXXXXXXB. If the initial condition of the tag system to be represented is 000110, the sequence region will contain XXXXBXXXXBXXXXBXXXXXXXBXXXXXXXBXXXX.[22] One of the features of this universal machine $U_{\mathbf{T}}$ is that every time the first 2 symbols are erased in the tag system simulated, the 2 first encoded symbols of the string contained in the sequence region are all set to 0. This portion of the tape thus becomes the erased region. E.g., if we suppose that the tag system simulated is Post's tag system, the shift number v however being equal to 2, instead of 3, the erased and sequence region together will contain the following sequence of symbols: 0000000000XXXXBXXXXXXXBXXXXXXXBXXXXBXXXXBXXXX after the machine has done what it has to do to simulate one iteration step of the tag process, applied to the above sequence of symbols. In repeating this process, the number

[21] The details of the exact encoding can be found in [Min62, CM63].

[22] The encoding of the last symbol of the string, is always the original encoding without the B.

> of 0's between the encoding of the string actually produced by the tag system and the encoding of the production rules will never shrink.
>
> Given the universality of this machine, it can be used to construct a universal tag system $U_{\mathbf{tag}}$, applying Minsky's coding discussed in 6.1.1. Given the encoding of $U_{\mathbf{T}}$, the length of the erased region can only grow. As a consequence, given Minsky's encoding from Turing machines to tag systems, the problem of tag in its second form is solvable for $U_{\mathbf{tag}}$. In this sense it is useful to talk about the *modified reachability problem*, following Margenstern [Mar00] who uses this terminology in the context of Turing machines. The modified reachability problem for tag systems is then the problem to determine for every arbitrary string S, and a given tag system T, the initial condition fixed, whether T will ever produce a string of which the last (or the first) l_S letters are equal to S.

Summarizing, the halting problem for tag systems is solvable for a given tag system or class of tag systems if it can be shown for each string over the alphabet of the tag system that it will lead to termination or non-termination. The problem in its second form is solved for a given tag system if one can prove that it will terminate, become periodic, or lead to unbounded growth, each of these possible classes of behaviour being illustrated through Figs. 6.1 - 6.3.

6.2.4 Conclusion

Post had already pointed out that one can identify three general classes of behaviour for tag systems: termination, periodicity or unbounded growth, the two forms of the problem of tag each being closely connected to each of these classes. If it can be shown for a given tag system that it will lead to one of these classes of behaviour after a finite number of steps, these two problems are solvable for that tag system. It is the possibility of what we have called fluctuating behaviour that makes these problems so hard to solve for specific instances of tag systems such as the one mentioned by Post. It is in this respect that we have called tag systems showing this kind of behaviour intractable. Of course, contrary to the other three classes of behaviour, fluctuating behaviour is ill-defined and the notion of intractability is only used here heuristically.

In the next, rather short, section we will give an example of a solvable tag system to give a better idea of how one might proceed to solve a given tag system and how this is connected to the three classes of behaviour. On the basis of this proof, we will also be able to prove a certain theorem about the possibility of reducing specific classes of tag systems to a certain number of other classes of tag systems.

6.3 Shifting through tags. Decomposition of tag systems.

6.3.1 An example of a solvable tag system.

The following tag system can be shown to be solvable, by decomposing it first in 6 different tag systems:

$$\begin{cases} v = 6 \\ 0 \to 001 \\ 1 \to 010011101 \end{cases}$$

If the tag system is started with a random initial condition of length 200, the string produced after $\lfloor \frac{200}{6} \rfloor = 33$ iterations, will have one of the 4 following forms:

$$\begin{cases} 00x_1x_2x_3..........x_{33} \\ 01x_1x_2x_3..........x_{33} \\ 10x_1x_2x_3..........x_{33} \\ 11x_1x_2x_3..........x_{33} \quad x_i \in \{001, 010011101\} \end{cases}$$

The first two letters (whatever they may be) determine a shift throughout $x_1x_2x_3$ ....x_{33}, i.e. if $x_1 = 010011101$, the fifth letter of 010011101 will be processed, if $x_1 = 001$, the second letter of x_2 will be processed. This tag system differs from, e.g., Post's exemplar, because the shift induced by the length of the initial condition remains constant in a certain way. This is due to the fact that the length of the respective words and the shift number v are not relatively prime. Based on this property we can prove that this tag system can be decomposed in 6 different solvable cases. A general proof that tag systems where the lengths of the words and v are not relatively prime are reducible to v different tag systems, will be given in the following subsection (6.3.2). For now, we will develop the example to prepare for the proof.

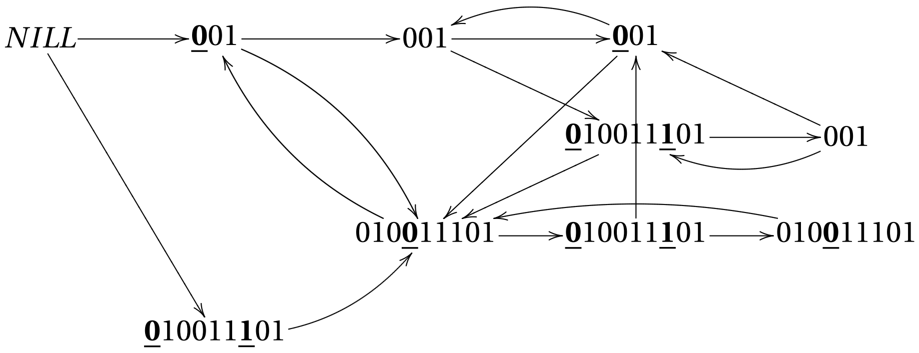
In order to understand the possibility of such decomposition, it should first be noted that for each initial condition with a length l, there are exactly $2^{l \equiv n \bmod 6}$ possible forms of the string $S_{\lfloor \frac{l}{6} \rfloor}$ produced after $\lfloor \frac{l}{6} \rfloor$ iteration steps. For example, for $l = 203$, there are 32 possible forms, i.e., the five last letters of the initial condition plus the words tagged at the end of the initial condition. Thus, the shift $S_{\lfloor \frac{l}{6} \rfloor}$ is "entered" with, is determined by l mod 6. Now let us look at each such possible shift $l \equiv n$ mod 6.

Case 1: $l \equiv 0$ mod 6**.** If the length l of the initial condition is divisible by 6, the string $S_{\lfloor \frac{l}{6} \rfloor}$ produced after $\lfloor \frac{l}{6} \rfloor$ iteration steps will be of the following form:

$$x_1 x_2 x_3 .... x_{\lfloor \frac{l}{6} \rfloor} \quad x_i \in \{001, 010011101\}$$

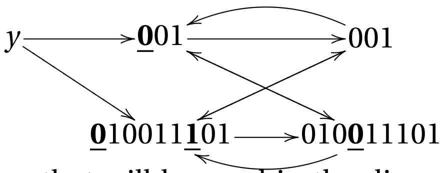
In the following iteration the first letter of x_1 will be processed, which is always equal to 0. If $x_1 = 010011101$ the next letter scanned is 1, the 7th letter of 010011101. If $x_1 = 001$ the next letter scanned is always 0.[23] This reasoning can be generalized, by looking at every possible combination of words 001 and 010011101, and all the possible letters of 001 and 010011101 that *can* be scanned during the tag process, for every such possible combination. This is done in the following transition scheme, where an arrow going from a word x to a word y is to be interpreted as x is followed by y. A letter is put in bold and underlined when it will be processed by the tag system.

[23] If $x_2 = 001$, the first letter of x_3 will be processed, which is always equal to 0. If $x_2 = 010011101$, the 4th letter of x_2 will be processed, which is also equal to 0.



As is clear from this scheme, it is indeed true that the shift determined by the length of the initial condition can, in a way, never be changed again. I.e., once started with a shift 0, it can be predicted that certain letters of the words will never be "scanned" by the tag process. If the letters of a word w_i of a given tag system are indexed as $a_0...a_{l_{w_i}-1}$, with l_{w_i} being the length of w_i, then letters 1 and 2 of 001 and letters 1, 2, 4, 5, 7 and 8 of 010011101 will never be scanned by the tag process if the initial condition has length $l \equiv 0 \bmod v$. The letters that will be scanned are: letter 0 of 001 and letters 0, 3 and 6 of 010011101, i.e. 0, 0, 0 and 1. Now, given the production rules of the tag system, we know that if the first letter of a given string S with length l is 0, the length of the string produced from S will equal $l-3$. If the first letter of a given string S with length l is 1, the length of the resulting string is $l+3$. From this it follows that if this tag system is started with an initial string of length l, such that $0 \equiv (l \bmod 6)$ it will either halt or become periodic. Why is this the case? Since only the first letter of 001 can ever be processed, every time 001 is encountered during the tag process, the resulting string must get 3 letters shorter. If 010011101 is processed there are two possibilities: the letter indexed 3 is scanned, or letters 0 and 6 are scanned. In the first case the resulting string gets shorter, in the second case the length of the resulting string remains invariant. In other words, whatever word 001 or 010011101 is processed, the resulting string can never grow, and the two forms of the problem of tag become solvable.

Before looking at the other cases, it should be noted that the above given scheme, while being more explicit, can be simplified to the following scheme:

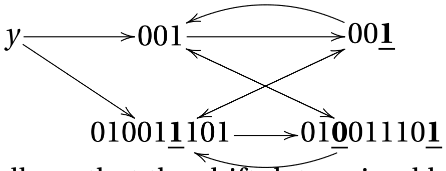


It is this simplified form that will be used in the discussion of the remaining cases.

Case 2: $l \equiv 1 \bmod 6$ If the length of the initial condition $l \equiv 1 \bmod 6$ the string $S_{\lfloor \frac{l}{6} \rfloor}$ produced after $\lfloor \frac{l}{6} \rfloor$ iteration will be of the following form:

$$yx_1x_2x_3..........x_{\lfloor \frac{l}{6} \rfloor} \quad x_i \in \{001, 010011101\}, y \in \{0,1\}$$

Applying the same reasoning as in the previous case, the following scheme shows which letters of 001 and 010011101 *can* be scanned in this case:



From this scheme, it follows that the shift determined by the length of the initial condition determines which letters of 010011101 and 001 will and will not be scanned. The letters never scanned are: letters indexed 0 and 1 of 001 and letters indexed 0, 1, 3, 4, 6 and 7 of 010011101. The letters that *can* be scanned are: letter 2 of 001 and letters 2, 5 and 8 of 010011101, i.e. 1, 1, 0 and 1. From this it follows that if the tag system is started with an initial condition with length $l \equiv 1 \bmod 6$ the strings produced can never get shorter. If 001 is processed, only its last letter can be scanned, resulting in growth. If 010011101 is processed, we have two possibilities: if letter 5 is scanned, the resulting string will grow, if letters 2 and 8 are scanned, the length remains invariant. From this it follows that in this case, the tag system can never become shorter, and the two forms of the problem of tag become solvable.

Case 3: $l \equiv 2 \bmod 6$ If the length of the initial condition $l \equiv 2 \bmod v$, the string $S_{\lfloor \frac{l}{6} \rfloor}$ produced after $\lfloor \frac{l}{6} \rfloor$ iteration will be of the following form:

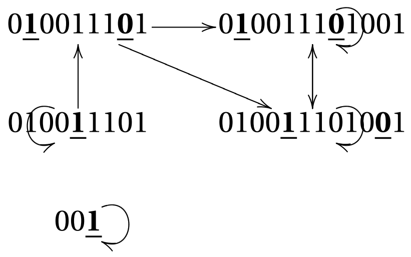
$$y_1y_2x_1x_2x_3..........x_{\lfloor \frac{l}{6} \rfloor} \quad x_i \in \{001, 010011101\}, y_n \in \{0,1\}$$

The following scheme shows which letters of 001 and 010011101 *can* and cannot be scanned in this case:

$$\begin{tikzcd}
y_1 y_2 \arrow[r] \arrow[dr] & 001 \arrow[r] \arrow[dr, leftrightarrow] & 0\underline{\mathbf{0}}1 \arrow[l, bend right] \arrow[dl, leftrightarrow] \\
& 0100\underline{\mathbf{1}}1101 \arrow[r] & 0\underline{\mathbf{1}}00111\underline{\mathbf{0}}1 \arrow[l, bend left]
\end{tikzcd}$$

This scheme proves that this tag system, when started with an initial condition with length $l \equiv 2 \bmod 6$ the following letters can never be scanned: letters indexed 0 and 2 of 001 and letters 0, 2, 3, 5, 6 and 8 of 010011101. The letters that can be scanned are: the 2nd letter of 001 and letters indexed 1, 4 and 7 of 010011101, i.e. 0, 1, 1, 0. Now one must be careful in concluding that also in this case the tag system is solvable. As was already argued with respect to the tag system mentioned by Post, it is not because one expects that on the average equally as many 0's as 1's might be processed that the system must become periodic or halt, since one then presupposes a random distribution of the sequence of relevant letters scanned. Still, it can be shown that even in the case the length of the initial condition $l \equiv 2 \bmod v$ the tag system discussed here is solvable: it will either become periodic, or, for a special subclass of initial conditions, halt. This can be easily proven by applying a certain method, which will be discussed in more detail in Sec. 7.3. This method looks at all the theoretically possible productions starting from individual words as follows. Take a word w_i, and look at all the possible strings that can be produced from w_i, by entering it with all the possible shifts, going from 0 to $v-1$. Thus from each word, $v-1$ strings are produced. If one of these strings is identical to w_i or any other string already produced, it is marked, since all possible productions from this string, are the same as those from the string it is identical to. Then the same procedure is applied to the unmarked strings produced from w_i, marking those strings that are identical to one of the strings already produced. If at a given time all the strings produced are marked, one applies the same procedure to the next word. If it is the case that this procedure applied to each of the words comes to an end – all the strings produced at a given time have been marked – one must conclude that the tag system is solvable, since the length of

any possible substring produced in the tag system is bounded.
In case of the tag system considered here, we do not have to take into account all the possible shifts, but only those allowed by the length of the initial condition. Since there are only two possible shifts with which 010011101 can be entered (0100**1**1101001 and 0**1**00111010**0**1), only two possible strings can be produced from 010011101: 010011101 and 010011101001. The first of these strings is again 010011101, so we only have to look at what happens with 010011101001. Again there are two possible shifts possible (0**1**00111010**0**1001 and 0100**1**11010**0**1). In either case the result is again 010011101001. As far as 001 is concerned, it can only lead to 001 (itself) or the nill string (when preceded by 001). The reasoning is summarized in the following scheme:



It follows that the tag system will always become periodic or halt if the initial condition is of length $l \equiv 2 \bmod v$. It will halt for any initial condition, with $l \equiv 2 \bmod 6$, for which all the relevant letters are 0. In all the other cases it will become periodic.

Cases 4–6: $\mathbf{l \equiv 4}$ mod 6, $\mathbf{l \equiv 5}$ mod 6, $\mathbf{l \equiv 6}$ mod 6. Each of the cases $l \equiv 4$ mod 6, $l \equiv 5$ mod 6, $l \equiv 6$ mod 6 can be reduced to cases 1–3 respectively.

It thus follows that both the halting problem as well as the reachability problem for tag systems is solvable for this one specific tag system. We have been able to prove this by reducing the decision problem of this tag systems to the decision problem of three other tag systems. Indeed, as should be clear from the example, the tag systems' solvability depends on the solvability of the following tag systems:

$$\begin{array}{lll} v=2 & 0\rightarrow 0 & 1\rightarrow 010 \\ v=2 & 0\rightarrow 1 & 1\rightarrow 011 \\ v=2 & 0\rightarrow 0 & 1\rightarrow 110 \end{array}$$

6.3.2 Generalization of the example.

The analysis given above clearly shows that the tag system $v = 6, 0 \rightarrow 001, 1 \rightarrow 010011101$, is solvable. Depending on the length of the initial condition, the tag process either leads to termination, periodicity or unbounded growth. The proof was based on the fact that the length of the initial condition determines which letters of the words will or will not be scanned, thus leading to the decomposition of the tag system in three other cases of tag systems, its decidability thus depending on the decidability of these three cases. This is possible only because the lengths of the respective words and v are not relatively prime. in this subsection we will prove the following theorem:

Theorem 6.3.1 *Given a tag system T with shift number v, $\Sigma = \{a_0, a_1, ..., a_{\mu-1}\}$ and words $w_{a_0}, w_{a_1}, ..., w_{a_{\mu-1}}$. Then, if the lengths l_{a_i} of the words and v are not relatively prime, the solvability of a given decision problem for T can be reduced to the solvability of the decision problem for λ different tag systems, λ being the greatest common divisor of v, $l_{w_{a_0}}, ..., l_{w_{a_{\mu-1}}}$, with shift number $v' = v/\lambda$.*

Given a tag system T with shift number v, μ letters and thus μ words $w_0, w_1, ..., w_{\mu-1}$ with respective lengths $l_0, l_1, ..., l_{\mu-1}$, with v and $l_0, l_1, ..., l_{\mu-1}$ having $\lambda > 1$ as their greatest common divisor (g.c.d.). Given this last feature, the following equation:

$$a_0 l_0 + a_1 l_1 + ... + a_{\mu-1} l_{\mu-1} + bv = 1 \tag{6.10}$$

with $a_0, a_1, ..., a_{\mu-1}, b \in \mathbb{Z}$, does not have integer solutions.[24] Then, given an initial condition I with length l_I over the alphabet Σ, the string produced after $\lfloor \frac{l}{v} \rfloor$ steps will always be of the following form:

$$\begin{array}{ll} y_1 y_2 ... y_n x_1 x_2 x_3 ... x_{\lfloor \frac{l_I}{v} \rfloor} & l_{y_1 y_2 ... y_n} \equiv l_I \bmod v \\ y_i \in \Sigma & x_j \in \{w_1, w_2, ..., w_\mu\} \end{array}$$

[24] See e.g., [Gau01], article 42 for this classical theorem.

If a string of this form has been produced, i.e. after $\lfloor \frac{l}{v} \rfloor$ steps, it is possible to determine what letters of any w_i will and will never become relevant – the letters of every word w_i being indexed from 0 to $l_{w_i} - 1$. This is possible by determining the solutions to the following equation:

$$a_0 l_0 + a_1 l_1 + ... + a_{\mu-1} l_{\mu-1} + bv + \overline{l_I \bmod v} = n \tag{6.11}$$

with $a_0, a_1, ..., a_{\mu-1}, b \in \mathbb{Z}$, and $\overline{x \bmod y}$ being the additive complement of x relative to y.[25]

Definition 6.3.1 *The additive complement $\overline{x \bmod y}$ of a given number x relative to a modulus y is defined as follows:*

$$\overline{x \bmod y} = \begin{cases} y - (x \bmod y) & \textit{if } x \neq 0 \bmod v \\ 0 & \textit{if } x \equiv 0 \bmod v \end{cases}$$

Eq. (6.11) can be used to determine the indices of all the letters that will and will never become relevant for each of the words w_i. For ease of explanation, suppose that from the point $y_1 y_2 ... y_n x_1 x_2 x_3 ... x_{\lfloor \frac{l_I}{v} \rfloor}$ is produced, the leading letters are never really erased by the tag system, but that you merely shift further and further to the right.[26] Then we are always working with a string of length $(l_I \bmod v) + a_0 l_0 + a_1 l_1 + ... + a_{\mu-1} l_{\mu-1}$. Instead of $l_I \bmod v$, $\overline{l_I \bmod v}$ is used in the equation, since we do not want to determine the "absolute index" of the relevant letters scanned, where absolute index means the x-th letter in the string produced at a given time. Rather we want to determine the indices of the relevant letters (to be) scanned, taking into account the shift $(l_I \bmod v)$ induced by l_I. The index of the letter scanned in a given word can be calculated by using additive complements, since given $l_I \bmod v$, $\overline{l_I \bmod v}$ letters will be erased in the first letters following $y_1 ... y_n$. The shift operation is represented in the equation by bv, moving $b \cdot v$ times to the right. As a consequence $a_0 + a_1 + ... + a_{\mu-1} = b$. Eq. (6.11) can be further simplified. Since $l_0, l_1, ..., l_{\mu-1}, v$ are not relatively prime,

[25] An alternate formulation would be: The absolute value of the least negative residue of x modulo y.

[26] This was actually the original conception of tag systems, where a *tag* moves along a string from left to right, cfr. Sec. 2.2.5.

they have a greatest common divisor (λ), thus $a_0\,l_0+a_1\,l_1+...+a_{\mu-1}\,l_{\mu-1}+bv$ must be divisible by λ and can be rewritten as $\lambda \cdot i$ and (6.11) can be rewritten as:

$$\lambda i + \overline{l_I \bmod v} = n \tag{6.12}$$

If $\overline{l_I \bmod v} > \lambda$, $\overline{l_I \bmod v}$ may be rewritten as $(\lambda \cdot m) + \overline{l_I \bmod \lambda}$, m being the quotient of l_I after division by λ, since v is divisible by λ. This rewriting operation implies that the cases for which $l_I \equiv x \bmod \lambda$ is valid, can be reduced to each other, even though different l_I, $\overline{l_I \bmod v}$ may give different solutions.[27] Putting $j = m + i$, the general form for (6.11) is:

$$\lambda j + \overline{l_I \bmod \lambda} = n \tag{6.13}$$

Given (6.13), we can determine the indices of all the letters of each of the words w_i that *can* be scanned by the tag system, given l_I, by the following sets of solutions:

$$\begin{cases} (\lambda j_0 + \overline{l_I \bmod \lambda}) & j_0 = 0, ..., \frac{l_0}{\lambda} - 1 \\ (\lambda j_1 + \overline{l_I \bmod \lambda}) & j_1 = 0, ..., \frac{l_1}{\lambda} - 1 \\ (\lambda j_2 + \overline{l_I \bmod \lambda}) & j_2 = 0, ..., \frac{l_2}{\lambda} - 1 \\ \vdots & \\ (\lambda j_{\mu-1} + \overline{l_I \bmod \lambda}) & j_{\mu-1} = 0, ..., \frac{l_{\mu-1}}{\lambda} - 1 \end{cases}$$

Applying this to the example of the tag system discussed above, e.g. supposing that $l_I = 199$, we get:

$$\begin{cases} (3j_0 + 2) = \{2\} & j_0 = \{0\} \\ (3j_1 + 2) = \{2, 5, 8\} & j_1 = \{0, 1, 2\} \end{cases}$$

As was shown the only letters which can be relevant for this case are the letter indexed 2 of 001 and the letters indexed 2, 5 and 8 of 010011101.

Now, since l_i is always divisible by λ, it follows that not every letter of each of the words can become relevant. Indeed, it follows from the above given set of equations that, given the length l_i of the word w_i corresponding with the i-th letter from the alphabet, only l_i/λ letters in w_i will ever be scanned once the tag process is started with a given initial condition I of length l_I. If the g.c.d. λ of

[27] In the example of sec. 6.3.1 the case $l_I \equiv 1 \bmod v$ is e.g. equivalent to the case $l_I \equiv 4 \bmod v$.

$v, l_0, l_1, ..., l_{\mu-1}$ is greater than 1, these numbers are not relatively prime, $l_i/\lambda < l_i$. From the method for calculating the indices of the letters that will be scanned and a given l_I, it follows that given a tag system T with g.c.d. $(v, l_0, l_1, ..., l_{\mu-1})$ $= \lambda > 1$, its decision problem can be reduced to λ cases. We have thus proven theorem 6.3.1.

Chapter 7

Constraints for intractable behaviour

7.1 Introduction

As was shown in the previous section, one can identify three general classes of behaviour with respect to tag systems: termination, periodicity and unbounded growth. It was furthermore shown how the two forms of the problem of tag can be shown to be solvable for a given tag system, if one can determine for that tag system and any initial condition that it will either terminate, become periodic or lead to unbounded growth after a finite number of steps. On the level of observing the behaviour of tag systems, the more "challenging" tag systems are those which show what was called fluctuating behaviour. As was said, while the mere observation of such behaviour does not guarantee anything about the future behaviour of the process, it can be considered as an indication of the intractable character of the tag system involved.

However, one must be very careful here. It might be the case that, even if one is working with a tag system that shows "fluctuating behaviour", having tested it for several initial conditions, there still exists a method or algorithm to solve its decision problems and thus predict its behaviour. This problem is a consequence of the *general* unsolvability of tag systems. Even if one has searched for years for a method to prove a specific tag system or a class of tag systems solvable, without finding anything pointing into the direction of such a solution, you can never be sure that there is not some (special kind of) method which predicts the behaviour of that (class of) tag system(s). Except of course, if one is able to prove that the decision problem of the tag system or class of tag systems considered is unsolvable.

This leads to the following problem. On the one hand, it is known that 2-symbolic tag systems, with $v \leq 2$ are solvable. The shortest universal tag system known so far, on the other hand, is rather gigantic: it needs 288 symbols and thus production rules, although $v = 2$. This leaves us with an infinite class of tag systems in between these two classes. Clearly, it would be very useful if we would find a way to make a kind of selection of tag systems which are intractable, to study the limits of solvability and unsolvability in tag systems from the perspective of the more "difficult" cases. "Fluctuating behaviour" seems to be a good indication of intractability. However, apart from the fact that this implies no theoretical guarantee, the question arises how one can use this rather intuitive description for making a selection of intractable tag systems in a computer search?

While the idea of fluctuating behaviour is helpful, it is only when one starts to search for an explanation of this behaviour that one is able to make such a selection. Of course such an explanation can never be completely formal, given the general unsolvability of tag systems. However, if one wants to study these in-between classes, it is better to have something instead of nothing, or, as they say, one bird in the hand is better than two in the bush. There are several features of tag systems that might contribute to an explanation for a tag system being predictable or unpredictable. While none of these features guarantee anything on a theoretical level, putting them together in an algorithm makes it possible to separate the more "difficult" tag systems from the easy ones. In this section, it will be shown how it is possible to make such a selection in discussing several of these features, putting them together as, what we have called, constraints for intractability. While it might have been more convenient to talk about features rather than constraints, we used this word because the "features" to be described were implemented in an algorithm for selecting "'difficult" tag systems from a class of randomly generated tag systems.

Some of these constraints have already been discussed in Sec. 6.1, others not. The last constraint to be described merely functions as a heuristic constraint. Constraint 3 is still in a conjectural stage and needs more research.

It should be emphasized that the notion of intractability is used heuristically here and should not be identified with the inherent intractability of certain tag systems, i.e. their unsolvability. It is only if we would be able to prove that the tag systems identified as being intractable have an unsolvable decision problem, that the two uses of the word 'intractable' coincide.

Some of the constraints discussed here are *decidability criteria* as defined by Margenstern in the context of Turing machines. Margenstern proposed the following definition for a decidability rsp. strong decidability criterion for Turing machines [Mar00]:

Definition 7.1.1 *Let c be an integer-valued function defined on a set M of Turing machines in with the following property: there is an integer f such that the halting problem is decidable for any machine $T \in M$ such that $c(T) < f$, and for any $k \geq f$ a machine $U \in M$ such that $c(U) = k$ and such that its halting problem is undecidable, respectively, such that U is universal, can always be constructed. In that case, c is called decidability (respectively, strong decidability) criterion and f is called its frontier value.*

The differentiation between decidability and strong decidability criteria is based on the fact that there exist Turing machines with an unsolvable halting problem that are not universal, i.e., Turing machines with an unsolvable decision problem of a lower degree of unsolvability. Based on this definition, a similar definition for decidability rsp. strong decidability criterion can be formulated for tag systems. However, not all the constraints discussed here can be regarded as decidability criteria. This will be discussed for each of the constraints individually.

In this chapter we will first describe the several constraints and then give two algorithms for generating tag systems, implementing all but one of the constraints. For each of the algorithms described, we will give a list of tag systems generated by them, and discuss them in the context of finding a method for proving certain tag systems equivalent. Before starting the description of the constraints, we will first add some notational conventions that will be used in the remaining chapters. Most of them are already in use, but we thought it useful for the reader to summarize them here.

7.2 Notational Conventions

An arbitrary tag system **T** is defined as follows. Given a positive integer v and μ letters over an alphabet $\Sigma = \{a_0, a_1, ..., a_{\mu-1}\}$. A set of μ words $w_0, w_1, ..., w_{\mu-1}$ is defined over the alphabet such that each letter a_i over the alphabet corresponds with one such word w_i:

$$\begin{array}{lll} a_0 & \rightarrow & a_{0,1}a_{0,2}...a_{0,t_0} \\ a_1 & \rightarrow & a_{1,1}a_{1,2}...a_{1,t_1} \\ ... & ... & ... \\ a_{\mu-1} & \rightarrow & a_{\mu-1,1}a_{\mu-1,2}...a_{\mu-1,t_{\mu-1}} \end{array}$$

We can then set up the following operation for obtaining from any given sequence A over the alphabet a uniquely derived sequence A' as follows. Depending on the first symbol a_i of A, tag the word w_i associated with a_i at the end of A. Then remove the first v symbols. This operation is repeated on A' resulting in A'',...

The length of a word w_i is indicated as l_{w_i}. The longest word in the set of words for a given tag system **T** is indicated as $l_{\mathbf{max}}$, the shortest as $l_{\mathbf{min}}$. In general l_x always means: "the length of x". The total number of times a symbol a_i from the alphabet appears in the set of words $\{w_0, w_1, ..., w_{\mu-1}\}$ is represented by $\#a_i$. For example in Post's tag system, #1 = 3, #0 = 3.

7.3 Theoretical, conjectural and heuristic constraints.

The main focus at the beginning of our research on tag systems was on the tag system mentioned by Post (which will be called **T1** here) and another tag system

T2 which seemed to be as difficult to predict as Post's tag system:

$$\mathbf{T2} = \begin{cases} v = 6 \\ 0 \rightarrow 00101 \\ 1 \rightarrow 1011010 \end{cases}$$

T2 was in a certain way developed by accident. I wanted to know whether in making the words of Post's tag system a bit longer, adapting v, this would also result in a tag system showing intractable behaviour. A "small" mistake was made here. Instead of 1101, 1011 was extended. It soon became clear that although trying to develop tag systems in a rather arbitrary way can help to improve ones intuition of tag systems, a more systematic method was needed. In this respect, some algorithms were programmed, able to generate tag systems for which there is a higher chance for intractability, combining several constraints a tag system must fulfill so that it does not become (trivially) solvable. The following constraints were taken into consideration. All but one were implemented in the algorithms.

7.3.1 Constraint 1: Post's condition

The first constraint was proven by Post, viz., that tag systems for which $\mu = 1$ or $v = 1$ or $\mu = v = 2$ are solvable (although the proofs were never published). In isolating both μ and v from each other it is clear that both function as decidability criteria. Indeed, since any tag system for which $v = 1$ is solvable, while, as was proven by Maslov [Mas4b], for each class of tag systems with $v \geq 2$ there exists at least one tag system that is universal, v is a strong decidability criterion. Although the frontier value for the number of symbols μ marking the difference between solvability and unsolvability is unknown, it is clear that it also functions as a decidability criterion, given the fact that the solvability or unsolvability of a tag system depends on its value.

7.3.2 Constraint 2: The Wang condition

As was shown in 6.1.1 Wang proved that the two forms of the problem of tag are solvable for any tag system **T**, for which $l_{\mathbf{max}} \leq v$ or $l_{\mathbf{min}} \geq v$ and constructed a universal tag system for which $l_{max} = v+1$ and $l_{min} = v-1$. A similar result was obtained by Maslov [Mas4b].[1] The values $\mathbf{l_{min}}$ and $\mathbf{l_{max}}$ when combined seem to function as (strong) decidability criteria, the frontier values f being equal to $v - \mathbf{l_{min}} = 1$ and $\mathbf{l_{max}} - v = 1$. For this condition to completely satisfy the definition for (strong) decidability criterion however, it should first be proven that for any class of tag systems with $v - \mathbf{l_{min}} \geq 1$ and $\mathbf{l_{max}} - v \geq 1$ there is at least one tag system that is universal.

7.3.3 Constraint 3. Proportions between $\#a_i$.

Another feature of tag systems that might be used to differentiate solvable from unsolvable tag systems is the proportion between the total number of times each of the letters appears in the respective words. To explain this idea, let us start from the class of 2-symbolic tag systems, and consider the following tag system:

$$\begin{cases} v = 3 \\ 0 \to 01 \\ 1 \to 1101 \end{cases}$$

This example is clearly based on **T1**, with the "minor" change that the second letter of w_0 has been changed from 0 into 1, thus setting $\#0 = 2$ and $\#1 = 4$. Now every time a 1 is scanned during the tag process, the resulting string will grow with one letter, erasing 3 letters, tagging 4. If a 0 is scanned, it will shrink with 1 letter, erasing 3 adding 2. Since $\#0 = 2$ and $\#1 = 4$, one is tempted to conclude that this tag system will always lead to unbounded growth, except for the initial condition 0. This is indeed the case. The system can only shrink when the first letter of w_0, or the 3th of w_1 is scanned. Now suppose that at a given time during the tag process, the first letter of w_0 has been scanned. How many iterations will it minimally take before the next 0 is scanned? If we would be working with **T1**, the answer is 0 steps, because one can have 00 followed by 0000. However within this tag system this is not the case. Once all the relevant letters in the initial condition have been scanned, the minimal number of iteration steps before the next 0 is scanned is equal to 1. Indeed, looking at all the possibilities of

[1] For more details, see Sec. 6.1.

words following 01, the quickest way to scan the next 0 is when 01 is followed by 3 times 01: **0**10**1**01**0**1. However, even this situation is rather "exceptional" since it is only possible if one has 000000000000 as a substring in the initial condition. Once 01010101 (produced from 000000000000) has been scanned, it can never happen again during the process that a string is produced that contains the substring 01010101. Looking at the other possibility of the system for scanning a 0, as the third letter of 1101, a similar reasoning can be applied, since the tail of 1101 is 01. This implies that every time a 0 is scanned, minimally one 1 is scanned, and one can thus conclude that the system can never shrink (of course after the relevant letters of the initial condition have been processed).

On the basis of this example, one might be *tempted* to conclude that if #1 > #0, a 2-symbolic tag system is always solvable. A same reasoning could be applied for the case where #0 > #1. This conclusion however can only be taken seriously if $v - w_0 = w_1 - v$. Indeed, the idea that #1 = #0 must be the case for a tag system not to be solvable, was based on the assumption that the amount of shrinking induced by scanning a letter is always equal to the amount of growth of a string. But what about those cases for which $v - w_0 \neq w_1 - v$? Consider e.g. the following example:

$$\begin{cases} v = 10 \\ 0 \rightarrow 0101100 \\ 1 \rightarrow 11000111100000 \end{cases}$$

Here, $v - w_0 = 3 < w_1 - v = 4$. This means that every time a 0 is scanned the system shrinks with 3 letters, and grows with 4 if a 1 is scanned. For the behaviour of this system not to be predictable, it seems important that *on the average*, for each 3 1's, 4 0's are scanned, since $4 \cdot (-3) + 3 \cdot 4 = 0$. For the above tag system, the proportion between #0 and #1 is indeed such that it seems not possible to conclude for the solvability of the tag system on the basis of the production rules. Since #0 = 12 and #1 = 9, we get $12 \cdot (-3) + 9 \cdot (4) = 0$. Due to this kind of "right" proportion between #0 and #1 one guarantees that one cannot predict the behaviour of the system on the basis of the proportion between #0 and #1. This kind of reasoning can be generalized to n-ary tag systems.

It seems reasonable to assume that there exists an infinite class of tag systems for which the halting and reachability problem are solvable if the proportion between all $\#a_i$ is such that the result of the following equation:

$$x = \#a_0(v - w_0) + \#a_1(v - w_1) + \ldots + \#a_{\mu-1}(v - w_{\mu-1}) \neq 0 \tag{7.1}$$

is a number $x \neq 0$. It is however far from trivial to prove this *in general*. Indeed, while it might be easy to prove for individual tag systems that they are solvable on the basis of this reasoning, finding a general method to prove this is not. In fact, such a proof is impossible since, although we are convinced that the constraint is valid for certain classes of tag systems, it is not valid for all classes of tag systems. For the constraint to be valid in general it needs further refinement. To show this, we will discuss three problems with respect to the constraint.

Problems 1 & 2: Minsky's universal tag system

A first problem related to the constraint occurred in checking its validity with respect to Minsky's second encoding of Turing machines into tag systems. In Sec. 9.2 a table is given of a universal tag system constructed using this encoding. In calculating the proportions between the symbols that lead to an increase and those that lead to a decrease, the result of (7.1) is 59, after replacement of the variables for the proper values. This implies that, in general, one suspects that this tag system should always lead to unbounded growth. If our reasoning with respect to constraint 3 would be generally valid, we should conclude that this tag system is solvable. Clearly this cannot be true since we are dealing with a universal tag system. Still, the prediction on the basis of the constraint, that the strings produced by this tag system – if it encodes this universal Turing machine in the proper way, i.e. indirectly encoding tag systems with a shift number $v = 2$ – will get longer and longer, thus showing unbounded growth, is valid.[2] Indeed, although the universal machine used to construct a univer-

[2] It is important that this getting longer of the strings produced can stop at a given time, if a halt takes place in the Turing machine, because the tag system simulated by the Turing machine scans a halting symbol. Still, the reachability problem for the universal tag system remains solvable, since one can compute a bound on the number of iteration steps for producing a string of given length n.

sal tag system differs from Minsky's 4-symbol 7-state machine, it is based on the same kind of encoding, its so-called erased region never getting shorter. As was stated in Sec. 6.2.1 we thus have to reformulate the reachability problem as the modified reachability problem for this kind of tag systems. Taking this consideration into account, we cannot but conclude that if we want to understand constraint 3 as a kind of decidability criterium, it can only be valid as a criterium with respect to the reachability problem for tag systems.

A further more intricate problem with respect to constraint 3, also appeared in the context of Minsky's second encoding of Turing machine into tag systems. This problem has to do with the use of the symbol x in this encoding scheme. This symbol functions merely as a kind of separator and is used to regulate the shift through the tag system. Since it is never supposed to be scanned, no corresponding word is attached to it. Now suppose that we would construct a tag system – representing a given Turing machine, using Minsky's encoding – for which, in replacing the variables in (7.1) by the proper values, the result of (7.1) equals 0, however, without taking into account the word corresponding to x. We could for example assign the word xx to x. If this tag system would also satisfy certain of the other constraints considered here, proving the tag system solvable might be very hard. The problem here is that as far as the representation of Turing machines in this tag system is concerned, the actual word assigned to x does not matter, since x is not supposed to be scanned anyway. This however is merely the case if we use specially encoded initial conditions. If we would work with arbitrary initial conditions, x plays a major role in the future behaviour of the tag system, since it is the symbol most frequently used in the production rules. If the word corresponding to x would be smaller than v, the tag system when started with arbitrary initial conditions will in the majority of cases, come to a halt. A similar reasoning can be applied for predicting the behaviour of this tag systems if $l_{w_x} = 2$ or $l_{w_x} > 2$.

This example shows that any proof of constraint 3 being a valid decidability criterium for a certain class of tag systems, must take into account the possibility of classes of initial conditions that are structured in a way that one or more symbols from the alphabet of the tag system under consideration, will never be scanned by the tag system and thus do not play a role in the development of the length of the strings produced by the tag system. If we would like to apply constraint 3 to a given tag system, we should first check whether it is possible to construct subclasses of such initial conditions. After this is done, constraint 3 should be applied to every such subset. I.e. for each such subset, one should take into account only those symbols that will be scanned by the tag system to compute the result of (7.1).

We have to do more research on this and other related problems with respect to constraint 3. For example, it could also be possible that although a given symbol will be scanned for a certain number of times during the tag process, that from a given point on, it will never be scanned again. In the short time we have spent on searching for refinements of constraint 3, we came to the conclusion that any proof of constraint 3 will most probably have to work with certain frontier values for the number of times each of the letters appears in the initial condition as a relevant letter.

Problem 3: The number of letters in small words, to which are assigned long words

The last problem, which showed up in connection to the proof of the solvability of the class of tag systems $\mu = v = 2$, has to do with the number of letters a_j in words w_i, $l_{w_i} < v$, to which are assigned words w_j, $l_{w_j} > v$. As will be discussed in Sec. 9.4.2, there exist certain frontier values with respect to #1 and #0 marking the difference between classes of tag systems that halt or become periodic, and tag systems that show unbounded growth. It will be shown there that for almost all cases (7.1) gives a very good estimate of this frontier value, except for those cases for which $w_0 = 1$; $l_{w_1} > 2$. For this case, the frontier value that can be calculated on the basis of constraint 3 is higher than the actual frontier value. This has to do with the fact that although #1 = 1, the word w_0 that leads to a decrease in the length of the string produced will, indirectly, always lead to an increase if the letter 1 produced by scanning 0, is scanned by the tag system. This case clearly illustrates that the equation for constraint 3 (7.1) should be further refined, taking into account the distribution of each of the letters over the different words. Again, more research is needed. We will not enter any further details concerning this last problem, since we will discuss it in more detail in Sec. 9.4.2.

Despite these three problems that arise in connection with constraint 3, we are convinced that further research on constraint 3 as it has been discussed here, could be used to differentiate solvable from unsolvable tag systems relative to the reachability problem. It should be noted though, that we are not sure whether we will be able to formulate it in terms of a decidability criterium in the sense of Margenstern, since it might not involve the existence of one absolute frontier value but rather one constant solution to a given equation or the existence of frontier values relative to the number of symbols and the length of the several words.

We also think that given a tag system, the classes of initial conditions that exclude certain symbols as relevant symbols can be constructed and are in fact computable. This would allow for a method to determine for a given tag system whether the constraint has to be applied to only one or more than one class of initial conditions.

To conclude, although there are clear reasons why constraint 3, as defined through (7.1), cannot be valid for the whole class of tag systems, there are clear indications that it might be useful for certain classes of tag systems. More research on the ideas behind the constraint might lead to a more exact criterium for differentiating between solvable and solvable tag systems, which can be proven in general.

We should also point out here that, notwithstanding the problems connected to constraint 3, we still implemented it as a constraint in our algorithms for generating tag systems. We did this because it is our experience that the majority of tag systems generated by random means are decidable if constraint 3 is not satisfied, i.e. if the result of Eq.(7.1) does not equal 0.

7.3.4 Constraint 4. The table method.

As is clear from the previous section, the proportion between the total number of occurrences of each letter of the alphabet in the words to be tagged, *can* make the difference between solvable and unsolvable tag systems. Another aspect is the order in which words are tagged at the end of a string, an order which is determined by the position in which the different letters appear in the respective words. To explain this, consider the following example:

$$\begin{cases} v = 4 \\ 0 \rightarrow 010 \\ 1 \rightarrow 11010 \end{cases}$$

At first sight, there seems to be no clear reason as to why this tag system would be solvable, while Post's isn't. Contrary to Post's, this tag system is solvable. This follows from the following table:

Table 7.1: Proof of the solvability of the tag system

	010	11010	11010010	01011010
$S0$	010 ✓	11010010	11010010 ✓	01011010 ✓
$S1$	11010	11010 ✓	11010010 ✓	11010010 ✓
$S2$	010 ✓	010 ✓	01011010	01011010 ✓
$S3$	NILL	11010 ✓	11010010 ✓	11010010 ✓

This kind of table will play a fundamental role in the proof of the solvability of the class of tag systems $\mu = v = 2$ (Sec. 9.4.2) and is another, more efficient, representation of the kind of proof given in Sec. 6.3.1 for the solvability of a certain tag system. There we already explained that one way to prove a certain tag system solvable, is to look at all the possible theoretical productions, starting from the words of the tag system. If it follows from these production that the lengths of the strings that *can* be produced from the words are bounded, the tag system will always halt or become periodic. This is also the method represented in the table, which, from now one, will be called the *table method*. We will now give a more "exact" explanation of this table method.

As was said, what one basically does with this method is to look at a certain number of substrings that can be produced theoretically in a given tag system, by starting from the possible productions from the respective words $w_0, ..., w_{\mu-1}$. Given a tag system T with a shift number v, it is clear that for any word $w_i = a_{i,1}a_{i,2}...a_{i,l_{w_i}}$ produced by the tag system at a given time, some letters in w_i will be scanned, others not. The sequence of letters that is scanned is determined by the number n, $0 \leq n \leq v-1$, of leading letters of w_i that is erased but not scanned by the tag system, called the *shift*, and which leads to the concatenation or tagging of the words corresponding to the letters from the sequence at the tail of a given string. For example, if $v = 3$, there are three different sequences of letters in w_i that might be scanned by the tag system: $a_{i,1}a_{i,4}...a_{i,t_0}$, $a_{i,2}a_{i,5}...a_{i,t_1}$, $a_{i,3}a_{i,6}...a_{i,t_2}$, with:

$$t_j = l_{w_i} - [(l_{w_i} - j) \bmod 3]$$

Now, given a tag system T, with shift number v and μ letters. The table method is applied to the tag system by first looking at all the possible strings v that can be produced from each of the words w_i, $0 \leq i < \mu$, by concatenating the words corresponding to the letters of each of the different sequences in each of the w_i, determined as above. If one of these new strings produced is equal to one of the words w_i it is marked. If all the strings produced in this way are marked or equal to ϵ it follows that the tag system will always halt or become periodic, since the length of the strings that can be produced from the respective words is bounded. If this is not the case, the same procedure is applied to all the strings left unmarked and not equal to ϵ,... If we, for instance, apply this method to the two words 00 and 1101 of **T1**, the tag system mentioned by Post, we get the following strings: 00, 00, ϵ, 11011101, 1101, 00. As is clear only one (11011101) of the 6 possible strings produced will be left unmarked, and differs from ϵ. If we then apply the method to this one string, and then again to the 3 unmarked strings that can be produced from 11011101, it becomes clear very soon that the method will never come to a halt, i.e., there will always remain strings left unmarked.

Now to explain the table representation of this method, let us return to the tag system proven solvable through table 7.1. The row headed with S_0 gives, for each of the strings S represented in the first row, the string produced from that string when the first letter of the string S is scanned by the tag system. In general, a row headed Sx, gives the possible productions from a given string S when its first x letters are erased without being scanned, i.e., when S is entered with a shift x. In the example, columns 2 and 3 show the possible productions from each of the words 010 and 11010, by entering it with the four different shifts (since $v = 4$), $S0, S1, S2$, and $S3$. If a string is produced that has already been produced in the table, or that is equal to one of the words, then this string is marked off, since we only have to trace the productions of a given string once. If an unmarked string is produced, the same procedure is applied to this string, looking at all the possible strings that can be produced by entering it with the possible shifts. The procedure halts if no produced string leads to the production of new strings that are unmarked. It follows that in applying this method to the tag system considered here, it will always become periodic or halt after a finite number of iterations.

This specific tag system is just one example of an infinite class of tag systems that can be proven solvable through the table method. But why is the above given tag system solvable? As is clear from the scheme, it is impossible to scan two consecutive 1's in any of the strings produced through the words. As a consequence the lengths must remain bounded. This feature is in part due to the position of the letters in the words. Indeed in slightly changing the above given tag system, by e.g. exchanging the last two letters of w_1 it becomes far from trivial to solve it.

Now, given a tag system, it is always possible to determine in a *bounded number of steps*[3] using the table method, whether the respective words allow for the production of at least one string that contains a sequence of relevant letters in which the number of symbols inducing growth, "outbalances" the number of symbols that don't.[4] In general however, we have not find a method to deter-

[3] This bound is given by the total number of possible combinations of length $l \leq \lceil l_{\mathbf{max}}/v \rceil$.

[4] The use of the word "outbalance" should be understood here in the sense of Eq. 7.1. I.e. given a sequence of relevant letters $a_{i,1}a_{i,2}...a_{i,n}$ from a string, then if the result of the equation 7.1 – by substituting the variables $\#a_i$ for the number of times $a_{i,j}$ appears in the sequence of letters, and the l_{w_i} for the lengths of the words assigned to each $a_{i,j}$ – is a number equal to or smaller than 0, than we say that the number of symbols inducing growth does *not* outbalance

mine whether the table method when applied to a given tag system will come to a halt or not. This is due to the fact that there are also tag systems, for which, although there are at first some strings produced by the method that contain sequences of relevant letters for which the number of letters inducing growth does "outbalance" the number of letters that don't, in the end, no string will be produced for which this is the case and the procedure will halt. Examples of such more complicated cases will be given in the proof of the solvability of the class of tag systems with $\mu = v = 2$. Because of the existence of such cases, we do not see any clear method to compute *in general* for any given tag system the maximum number of steps it can take before the table method comes to a halt. Of course, if one has applied the table method for some iterations for a given tag system, and it has not come to a halt yet, it might be possible to prove for that individual case that the method will never come to a halt and can thus (possibly) not be used as a means to prove the tag system solvable. However, being capable of using it for individual cases does not lead to a general method for applying it.

The table method applied to the tag system defined at the beginning of this section is very simple, but yet a powerful instrument to study tag systems. As we will e.g. see in Sec. 9.4.2, even if the table method does not come to a halt, it can still be used to prove particular classes solvable. We feel that we have not exhausted the possibilities of this method yet and more research is needed here.

It is very straightforward to implement this procedure on the computer. As such it could be used as a kind of heuristic constraint to generate classes of tag systems with a higher chance for intractable behaviour, using a bound n on the number of strings that are produced through the procedure. If the procedure halts before n strings are produced one can exclude the tag system. At the time I wrote the algorithms, I did not have this method yet, hence it was not implemented.

the number of symbols that do not give rise to growth.

7.3.5 Constraint 5. On the number of iterations.

The last constraint to be discussed is the most experimental, and one could definitely doubt the use of the word constraint here. Still, since it was actually used as a constraint we will not deviate from our terminology. Given a tag system which fulfills constraints 1-4, one still cannot be sure that all the tag systems fulfilling these constraints, will give rise to intractable behaviour. An experimental way to reduce this class further is to select only those tag systems which are able to produce millions of strings, without leading to either termination, periodicity or unbounded growth. Several problems, however, are connected to this approach, due to its experimental character. We will describe here how the constraint was implemented, as well as some of the problems that are connected to it.

First of all, it should be noted that the fact that a given tag system is able to produce millions of productions without becoming predictable, does not guarantee anything about productions in the order of trillions, due to the general unsolvability of tag systems. This problem never vanishes: even if one has initial conditions for a given tag system that lead to the production of trillions of strings, this in its turn doesn't say anything on the level of trillions of trillions of trillions of productions. One thus has to impose a limit somewhere, and I have chosen for 10.000.000. If a given tag system satisfies constraint 1–4, the tag system is tested as to whether it is possible to run for 10.000.000 iterations, before leading to termination, periodicity or unbounded growth.

Now, how to find such initial conditions? In the implementation of the constraint, a given tag system was tested for 20 different initial conditions, generated using a pseudo-random number generator, of length 300. If none of these leads to unpredictable behaviour (after 10.000.000 iterations) the tag system is not selected. Random conditions were chosen, since it is my experience that in most of the cases a sample of random conditions lead a 'representative' sample of the behaviour[5], at least if one is using tag systems that have not been en-

[5] We thus used a kind of crude Monte Carlo method, assuming that randomly sampling the space of all initial conditions, will produce a more or less statistically representative picture of average behaviour of a tag system. Strictly spoken though, this assumption rests upon the

coded in some kind of special way, such as the universal tag system already discussed. We only tested 20 conditions to spare time and because one has to pose a limit somewhere. Related to this choice is the length of the initial conditions. Since we only maximally tested 20 initial conditions, it seemed reasonable to chose rather long initial conditions.
Now, in testing a certain tag system for 20 different random initial conditions, one also has to develop methods to test whether the system has led to one of the three general classes of behaviour. As far as termination is concerned, this is straightforward to check, since termination occurs when the empty string ϵ is produced. In order to check periodicity there is one ultimate method: store each string produced and compare it with all the preceding strings. This method is guaranteed to work, but, as Brian Hayes remarks in [Hay86], p. 23,

> The trouble with the brute-force method is that it requires too much brute force. For a pattern that runs through a million iterations, before entering a cycle, with an average string lengths of 1000 digits, the storage requirement would be at least a gigabyte. Furthermore, roughly 500 million string comparisons would be needed.

Given the time and memory this method takes, another approach was chosen, basically working with a kind of sampling.[6] The programs discussed below store every 1.000.000th string produced and compare every new string produced with the stored one, the reference string being replaced by a new one every 1.000.000th iterations. This method is very efficient as compared to the previous, but excludes the possibility of having periods longer than 1.000.000. We chose to prefer efficiency over correctness and thus used this less correct method. The tag systems that were selected to be used for further experiments finally, were then each tested individually. Every one of the tag systems to be used was rerun with initial conditions that were supposed to lead to the production of 10.000.000, using a more efficient, but still very slow, variant of the brute-force method. After this test, four tag systems were withdrawn, because they had periods larger than 1.000.000. The remaining tag systems are those we

"reasonable belief" this will be the case, not upon statistical derivations.

[6] Cfr. preceding footnote.

will use in the next chapter.
While termination and periodicity are rather easy to detect unbounded growth is far more challenging. This is the case because there are many different ways for a system to grow unboundedly. In the simplest case, a system grows in a more or less linear way, such that e.g. after every 100 new strings produced, the 101th will never be shorter than the 1st of these 100. However, since we do not know how fast or slow this growth proceeds, it is hard to determine some kind of measure here. It might, e.g., be the case that only every 1.000.000th string, lengths are produced which never become shorter than e.g. the 900.000th of the previous block of 1.000.000 strings produced. Furthermore, this kind of growth is not the only one possible. Growth can also proceed in a christmas-tree-like way: the process grows for a very long time, e.g., going from a string of length 100 to a string of length 10000, but then the strings become shorter and shorter until length 200, in its turn evolving until a string of say length 10100 is produced,....How will one detect this kind of growth? Still other kinds of growth are possible such as e.g. growth which is in a way the result of a combination of two christmas trees, e.g., going from 100 to 2000 to 500 to 1500 to 200 to 2100 to 600 to 1600,...
A first simple solution to this problem seems to be the following: store the length of every 10.000th string produced. If, after 10.000.000 steps, these lengths have never decreased, one concludes for a case of unbounded growth. Clearly this method cannot work. First of all, it is not because every 10.000th string is always longer as compared to the previous 10.000th string that the process shows unbounded growth. E.g. if the tag system has been fluctuating around the same length for 10.000 steps, this method will not work. Secondly, if the system grows very slow, it might be the case that while the 9.000.000th string is shorter than the 8.900.000th string, this does not exclude unbounded growth. Furthermore if one is having a christmas tree growth, this method must necessary fail if one does not know the right distances. There are several other problems involved here, and I finally decided to use the rather brute method of rejecting an initial condition as a possible condition if it does not lead to predictable behaviour after 10.000.000 iteration steps, if a string is produced the length of which is longer than 15.000. While this method can lead to the exclusion of tag sys-

tems on the wrong basis – it is not because a string of length 15.000 is produced that the system shows unbounded growth – it was considered as the least time and memory consuming, both on the level of the programming itself as well as on the level of the actual execution of the code. Indeed, developing a correct program for detecting such growth might have taken several weeks if not months from my research time, especially since one first has to find out about all the different ways in which a tag system can grow, and provide a good formalization for these different ways, such that they can be implemented into an algorithm.[7] The limit of 15.000 is considered reasonable, since the tag systems are all started with an initial condition of length 300, running maximally for 10.000.000 iterations. In this respect some limited space for growth is allowed for, while, if the system is indeed showing unbounded growth, the chance that it will have passed the limit of 15.000 after 10.000.000 is rather high. Of course, efficient though as this method might be, perfect it is not.

As is clear from the previous, there is a huge difference between constraints 1-4 and the more experimental constraint of checking whether a certain tag system can run for at least 10.000.000 iterations without becoming predictable: it is a far less elegant way to further restrict the class of possible intractable tag systems. Despite its non-elegance, it is in a certain way the most powerful. Indeed, in applying only one of the constraints 1-4, to restrict the class of possible intractable tag systems, one would still have a very large subclass of tag systems which are far from intractable. However, merely applying the more "experimental" constraint discussed here, to tag systems which are randomly generated, will result in a far more restricted class. Indeed, if one of the constraints is not valid for a given randomly generated tag system (except for certain cases where constraint 3 is invalid) there is a very high chance, if not certainty, that the constraint discussed here will lead to the rejection of the tag system. In a way constraint 1-4 are thus merely speed-ups for constraint 5: they lead to a

[7] In a very interesting paper [MS90], describing methods to calculate particular values for the Busy Beaver function, the identification of such different kinds of growth and writing decent algorithms to detect them, was fundamental to the authors' research. In fact, the design of these algorithms was based on the larger part of Machlin's Ph.D. dissertation of Machlin [Kop81], one of the co-authors.

restricted class of tag systems, on the basis of which a certain selection can be made using constraint 5. In other words, although this constraint will never guarantee anything about the tag systems selected, nor about the tag systems excluded, it offers us a very good method to select tag systems that might be called intractable.

7.4 Algorithms for generating possible intractable tag systems.

The constraints we discussed can help to find tag systems which might be intractable, and thus generate possible candidates for particular instances of tag systems for which neither the halting problem, nor the reachability problem are solvable. In this section we will describe two algorithms that were effectively implemented and run on my computer. The second algorithm is a generalized version of the first. We also implemented a third algorithm which is a further generalization of the second algorithm, generating n-symbolic tag systems. Since focus in the chapters to follow will be put on 2-symbolic tag systems, we will spare the reader the description of this last algorithm. The interested reader is referred to Appendix A.

7.4.1 Algorithm 1: Two-symbolic tag systems, $v - l_{w_0} = l_{w_1} - v$

The first tag generating algorithm I ever implemented takes into account constraints 1, 2 and 5, and generated 2-symbolic tag systems. A restricted form of constraint 3 was also implemented: only those tag systems for which $v - l_{w_0} = l_{w_1} - v$ were taken into account, thus setting #0 = #1.
The shift number v is determined at random, with maximal value $v = 15$. Of course the choice of 15 is in a certain way completely arbitrary, and can be easily adjusted in the algorithm, but posing a limit somewhere is necessary. This choice is motivated by the length of the initial conditions tested, the larger v becomes the less letters will be scanned in the initial condition. Since all further experiments to be discussed always start with initial conditions of length

300 this limit is indeed a good one, although one cannot exclude a certain degree of arbitrariness.

After v is determined, and after it has been checked that it is greater than 2 (constraint 1),[8] the algorithm randomly generates the length of w_0 again, using the method given above. This procedure is put into a conditional loop: it is repeated until the algorithm generates a number such that $0 < l_{w_0} < v$. After l_{w_0} has been generated, l_{w_1} is set to $v + (v - l_{w_0})$, thus assuring that $v - l_{w_0} = l_{w_1} - v$. After $\#0 = \#1$ are both set to v,[9], the two words w_0 and w_1 are generated in a random way as follows. First, w_0 is calculated. Its letters are determined by using a random number generator, resulting in numbers x between 0 and 1. If $0 < x \leq 0.5$, a 0 is concatenated to w_0, else 1 is concatenated. If l_{w_0} letters have been generated, w_1 is calculated. In using a counter, keeping track of the number of times a 0 or 1 has been concatenated to one of both words, the procedure prevents #1 or #0 becoming greater than v. Once #1 (or #0) is equal to v, no 1 (or 0) will be concatenated again. After v, w_0, w_1 have been generated, constraint 5 is applied, testing the tag system for maximally 20 randomly generated initial conditions. If none of these conditions leads to the production of 10000000 strings, the tag system is rejected else it is accepted. Some of the tag systems generated with this algorithm are given in the following table:

Table 7.2: Tag systems generated by Algorithm 1

Tag System	**Word$_0$**	**Word$_1$**	**Shift Number v**
A1	1000100	110111001	8
A2	0100100	10111	6
A3	11110010001	100111000	10
A4	1	01100	3
A5	10110110110111	101010000000	13
A6	01111	10000101100	8
A7	1011010011110	00100010011	12

Continued on next page

[8] If $v = 2$, a new v is generated.

[9] $\#0 = \#1 = v$, since it is always the case here that $l_{w_0} + l_{w_1} = 2v$ and $l_{w_0} + l_{w_1} = \#0 + \#1$

Table 7.2 – continued from previous page

Tag System	**Word$_0$**	**Word$_1$**	**Shift Number v**
A8	0001	001111	5
A9	1	01001	3
A10	100010110	10000011111	10
A11	0111	101000	5
A12	000001010111111	00110001110	13
A13	01111	100100010	7
A14	101110	11010000	7
A15	100	11100	4
A16	1101100	010000111	8
A17	001010111	01010011010	10
A18	00101110100000	10110111	11
A19	0	11010	3
A20	1	0001101	4
A21	010011001000000	01011111111	13
A22	010110	00101101	7
A23	011	1001100	5
A24	101	10001011100	7
A25	100	01101	4
A26	01	0110	3
A27	110	01010	4
A28	11011	1100000	6
A29	11	0010	3
A30	10	1100	3
A31	0000110100	100011101111	11
A32	1001	111000	5
A33	010	1010011	5
A34	100111010	01010100011	10
A35	10110	0101100	6
A36	1100100110	111101010000	11
A37	0110101	011000011	8

Continued on next page

Table 7.2 – continued from previous page

Tag System	**Word$_0$**	**Word$_1$**	**Shift Number v**
A38	101	11000	4
A39	001	00111	4
A40	110000011	11110100010	10
A41	1010111101	011110000000	11
A42	1101111000	111011000000	11
A43	1111111011	001000010000	11
A44	0001001001010	110111110	11
A45	11	0100	3
A46	11	0010	3
A47	0	10110	3
A48	1110	100001	5
A49	00	1101	3
A50	0100	110011	5
A51	1011010000100	11001111100	12
A52	0111	000110	5

The first thing to be noted is that in running the algorithm, **T1** (Post's tag system) was generated (**A49)**. This is interesting not only because it is an indication of the fact that the constraints can indeed be used to select intractable tag systems, but also because in comparing **T1** with the other tag systems generated, there are some that seem to be very similar to **T1** as is e.g. the case for **A45**. If we concatenate e.g. w_0 and w_1 of **T1**, we get a string 001101. We could then apply the following iterative process on this string: erase the first letter, and tag it at the end of that string. Apply this same operation on the resulting string,...until 011111100 is produced again. After two application of this process, we get 110100, which is exactly the string which would result from concatenating w_0 and w_1 of **A45**. Given this kind of convertibility of **T1** in **A45** (and vice versa), their shift numbers being equal, and the fact that they were both selected by the algorithm as possible intractable tag systems, might lead one to the conclusion that both systems might be reducible to each other, and one could thus find a way to define certain equivalence classes of tag systems. In this respect it is important to add the following small intermezzo on *rotated combinations*.

Rotated Combinations At one time during my research, I was convinced that in generalizing the kind of conversions sketched above, it would be possible to define whole classes of equivalent tag systems, even allowing a variation in the lengths of the respective words, as follows. Given a combination C of μ letters a_i from a finite alphabet Σ. Any combination which is the result of removing k letters from C ($k \leq l_C$) and tagging them at the end of C, is called a rotation. Now, given a tag system T, concatenate its words resulting in a combination C_T, and generate all the rotations of C_T. Then it is always possible to generate a certain number of *different* tag systems, for which v and μ remain constant, and the concatenation of the respective words are rotations of the same combination through the following procedure. For every rotation of C_T, a set of μ words w_i can be generated by splitting up C_T in μ pieces, such that the length of the first piece never exceeds $l_{C_T} - (\mu - 1)$. Applying this to **T1**, the following set of different tag systems, are such that their shift number and μ are equal to those of **T1**, the concatenation of their words all being rotations of the same combination:

$$\left\{ \begin{array}{ll}
0 \rightarrow 0 & 1 \rightarrow 01101 \\
0 \rightarrow 00 & 1 \rightarrow 1101 \\
0 \rightarrow 001 & 1 \rightarrow 101 \\
0 \rightarrow 0 & 1 \rightarrow 11010 \\
0 \rightarrow 01 & 1 \rightarrow 1010 \\
0 \rightarrow 011 & 1 \rightarrow 010 \\
0 \rightarrow 1 & 1 \rightarrow 10100 \\
0 \rightarrow 11 & 1 \rightarrow 0100 \\
0 \rightarrow 110 & 1 \rightarrow 100 \\
0 \rightarrow 1 & 1 \rightarrow 01001 \\
0 \rightarrow 10 & 1 \rightarrow 1001 \\
0 \rightarrow 0 & 1 \rightarrow 10011 \\
0 \rightarrow 01 & 1 \rightarrow 0011 \\
0 \rightarrow 1 & 1 \rightarrow 00110 \\
0 \rightarrow 10 & 1 \rightarrow 0110
\end{array} \right.$$

As is clear from this list, one cannot simply conclude that tag systems for which μ and v are the same, while the concatenation of their words are all rotations of

> the same combination, are reducible to each other, since the tag systems from this list for which $l_{w_0} = l_{w_1}$ are solvable in a trivial way. Still it seems reasonable to suppose that in starting from such a set of such tag systems, it is possible to find a subset in this set of tag systems which are reducible to each other. One might e.g. try to argue for the mutual reducibility of tag systems for which μ, v as well as the lengths of the words are all the same, while the concatenation of the words are rotations of the same combination. However, it seems far from trivial to prove that this is indeed the case. In fact, I am not convinced that, while it might be true for some individual cases, this can be proven in general. Indeed, as we already know, the order in which words are tagged, determined by the position of the letters and the shift (resulting from v and the lengths of the words) can be a determining factor for the behaviour of a given tag system. It is in this respect that there is a fundamental difference between tag systems, sharing the same constants v and μ, the concatenation of their words being rotations of the same combination. Still, in excluding the trivially solvable cases the possibility of finding whole classes of intractable tag systems, using this method of rotated combination, is worth more research. A start could be made, by generating more tag systems with the tag generating algorithms, in order to see whether the number of tag systems defined this way, grows.

The following sets of tag systems from Table 7.2, all have the same constants v and μ, the concatenations of their words being rotations of the same combination: {**A26**, **A29**, **A30**, **A46**, **A47**, **A4**},{**A19**, **A45**, **A49**, **A9**},{**A39**, **A15**},{**A25**, **A20**},{**A32**, **A8**}, {**A48**, **A11**}.

7.4.2 Algorithm 2: Two-symbolic tag systems, $v - l_{w_0} \neq l_{w_1} - v$

The second tag generating program, generates 2-symbolic tag systems for which it is not necessary the case that #0 = #1, i.e. tag systems for which it can be the case that $v - w_0 \neq w_1 - v$. It is thus a generalization of the previous algorithm. Here, v is determined in the same way as was done in the first algorithm. To determine the lengths, two numbers are generated at random. The length of w_0 is set to the smallest number, the length of w_1 is set to the largest. If $l_{w_0} \geq v$

or $w_1 \leq v$, new lengths are generated.

Since it is not necessarily the case that #0 = #1, an extra procedure is added to determine #0 and #1, such that constraint 3 is satisfied. The algorithm is based on the following equations:

$$\begin{aligned} \tfrac{a}{b} &= \tfrac{v-l_{w_0}}{l_{w_1}-v} \\ a \cdot x + b \cdot x &= l_{w_0} + l_{w_1} \end{aligned} \tag{7.2}$$

which, when worked out result in the following solution:

$$x = \frac{l_{w_0} + l_{w_1}}{a+b} \tag{7.3}$$

If $\frac{l_{w_0}+l_{w_1}}{a+b}$ is not an integer it is not possible to find the "right" proportion (constraint 3) for #0 and #1 for the specific values for l_{w_0}, l_{w_1} and v generated by the algorithm. Indeed, a and b give us the proportion between #0 and #1, on the basis of v, w_0, w_1: scanning 0 results in a decrease of a letters, scanning 1 results in a growth of b letters. Consequently, in order to avoid unbounded growth or termination, for every a 1's scanned, b 0's have to be scanned. To calculate the values of #0 and #1, respecting this proportion between a and b one merely has to determine how many times x, a and b can be multiplied, such that $ax + bx = l_{w_0} + l_{w_1}$. Once this solution has been calculated a and b are multiplied with x to find resp. the values for #1 and #0. If the result of this computation is not an integer, this means that it is not possible to satisfy constraint 3 with the specific values found for v, w_0 and w_1. If this is the case, the whole program is restarted, determining new values for v, w_0 and w_1.

Once v, l_{w_0}, l_{w_1},#0,#1 have been determined, the algorithm generates w_0 and w_1. This is done in a way similar to the previous algorithm, however now using a biased random number generator, taking into account the fact that it is not necessary the case that #1 = #0. This is done as follows. First the number $x = \frac{\#0}{l_{w_0}+l_{w_1}}$ is calculated. For each random generated number r that is generated, if $0 < r \leq x$, a 0 is concatenated (if the number of 0's generated is not equal to #0), else a 1 is concatenated (if the number of 1's generated is not equal to #1).

Once the tag system has been completely determined, it is checked whether constrained 5 is satisfied, as in the previous algorithm. In the following table, a list of tag systems generated with this algorithm is given:

Table 7.3: Tag systems generated by Algorithm 2

Tag System	**Word$_0$**	**Word$_1$**	**Shift Number** v
T3	111	01000	4
T4	11101	1100000	6
T5	010110	11100100	7
T6	0	01011	3
T7	101011	00011010	7
T8	011	111100	5
T9	101	0000111	5
T10	001	10110	4
T11	001	01110	4
T12	0	01011	3
T13	0110001	10000101111	9
T14	1010	110100	5
T15	111	0110000	5
T16	111000	11010110011000	10
T17	1001111	10100000011	9
T18	000110	101001010000	8
T19	110	001111	5
T20	1011000	111011000	8
T21	11011011	1110000000	9
T22	101001001	0101100110011	11
T23	001	010100	4
T24	11	00111000	5
T25	10000111	1000100111	9
T26	00111	0111000	6
T27	11011	0011000	6

Continued on next page

Table 7.3 – continued from previous page

Tag System	**Word$_0$**	**Word$_1$**	**Shift Number v**
T28	111000	11000110011100	10
T29	110	01001	4
T30	000111	11000011	7
T31	1	10100	3
T32	111010101110	00110101010000	13
T33	10001	1110010	6
T34	010	001001	4
T35	0010101	01010100100011	10
T36	1011	010100	5
T37	1111	010000	5
T38	000101	000000111	7
T39	00101	1001000110	7
T40	001	110000	4
T41	101	00001110011	7
T42	10111	0000011	6
T43	100	11001	4
T44	1111	00110000	6
T45	101	0011010	5
T46	1011	110000	5
T47	0	1001101	4
T48	11010011110	1111000010000	12
T49	001	100100	4
T50	110	11000	4
T51	1110010	00111110000	9
T52	01101	0111000	6

T8, **T19**, **T23**, **T34**, **T35**, **T38**, **T39**, **T40** and **T49** are examples of tag systems for which #0 ≠ #1. From those tag systems for which #0 ≠ #1, it should be noted that **T8** and **T19** have the same constants v and μ, the concatenation of their words being rotations of the same combination. Other such sets in Table 7.3 are: {**T12**, **T6**}, {**T50**, **T10**}, {**T37**, **T9**}, {**T26**, **T42**, **T4**}. In combining the results from tables 7.2 and 7.3, we get the following sets:{**A26**, **A29**, **A30**, **A46**, **A47**, **T6**, **T12**, **A4**}, {**A19**, **A45**, **A49**, **T35**, **A9**}, {**A39**, **T29**, **A15**}, {**A25**, **T10**, **T50**, **A20**}, {**T11**, **A38**}, {**A32**, **A8**}, {**A48**, **T9**, **T37**, **A11**}, {**T45**, **A33**}, {**T24**, **A52**}, {**T19**, **T8**}, {**T26**, **T42**, **T4**}, which means that 37 of the 102 tag systems already generated are part of a set of tag systems which are possibly intractable, having the same constants μ and v, the concatenation of their words being rotations of the same combination.

While tag systems for which the lengths of the words and v are relative prime can be further reduced to a number of tag systems equal to the g.c.d. of the lengths and v, such tag systems were not excluded by the program in its 1st version. In table 7.3 there are three such tag systems: **T16**, **T28** and **T45**. Since the lengths of the initial conditions used in testing whether the tag systems can run for 10000000 iterations without becoming predictable, are all of length 300, it is easy to deduce the tag systems contained in **T16**, **T28** and **T45** which were actually selected through the constraints. Indeed, since the respective shift numbers are 10, 10 and 6, it is clear that one should only take into account the letters from the respective words which *can* be scanned with a shift 0.[10] The actual tag systems, contained in **T16**, **T28** and **T45**, which passed all constraints are:

$$\begin{cases} \textbf{T17}, \text{Shift } 0 & v = 5 \quad 0 \rightarrow 110 \quad 1 \rightarrow 1001010 \\ \textbf{T31}, \text{Shift } 0 & v = 5 \quad 0 \rightarrow 110 \quad 1 \rightarrow 1001010 \\ \textbf{T49}, \text{Shift } 0 & v = 3 \quad 0 \rightarrow 11 \quad\;\;\, 1 \rightarrow 0100 \end{cases}$$

Since the tag systems that actually passed all tests contained in **T16** and **T28** are the same, it is not necessary to consider both of them in further experiments. It should be noted that the tag system resulting from **T45** should be added to the set of 2-symbolic tag systems deducible from **T1**, through the method of rotated combination. The tag system contained in **T16** and **T28** is deducible in this same way from **T40**.

[10] Since the length of the initial is divisible by all v's.

Chapter 8

Playing with tag systems. An experimental approach.

> What we have in mind is using *the computer as the mathematician's laboratory*, in which he can perform experiments in order to augment his intuitive understanding of a problem, or to search for conjectures, or to produce counterexamples, or to suggest a strategy for proving a conjecture. [...] *We shall not argue the philosophical merits of mathematical experiments* [m.i.]; we are more concerned with convincing mathematicians with different research orientations that it may be worth their while to try some experiments of their own on the computer. Obviously, there are many areas in mathematics where experiments would be of little or no help. In other, however, experiments may be really useful, sometimes in situations where this does not appear likely at first. Therefore the message is: Try it!
>
> Ulf Grenander, 1981.[1]

While one could say that any mathematician does some kind of "experimentation", think of some of the research by Euler or Gauss, working out several cases, in order to prove a more general result, it is with the rise of the computer that the idea of "experimentation" in mathematics has become much more explicit (See Sec. 4.2). In being confronted with certain objects or systems for which we have to rely on "computer experiments" in order to gain new results, one

[1] [Gre82], pp. xiii – xiv.

becomes much more aware of the fact that mathematics should not be contrasted too much with physics, biology or any other science depending on “des choses composées” (cfr. Descartes, quoted in the introduction). It is in this respect that we will use the word experiment here. Since we are facing systems the behaviour of which can be considered intractable, where *initially*, the only way to come to a better understanding of these systems, is to observe certain features of their behaviour, we indeed have to rely on some typical aspects of experimentation. We will have to set-up experiments, to answer certain questions, where none of the results from the experiments can *directly* lead to mathematically rigorous results. However, they can be used to build up an intuition of the systems involved, they can help to formulate and strengthen certain conjectures that have to be further investigated, they can lead to strategies to gain new results or, the further analysis (by hand or by computer) of the heuristic results can lead to rigorous ones. The element of surprise is always present. Although we often have certain expectations about the results from the experiments, it will become clear throughout the experiments, that while the results often more or less satisfy the expectation, the precise details of the results often come as a nice surprise.

In this chapter, we will approach tag systems from this experimental viewpoint, by using computer experiments to study and characterize their behaviour. To be more specific, we will use the tag systems generated by algorithm 2 from Sec. 7.4.2, together with **T1**, Post’s tag systems and **T2**, the tag system I once wrote down almost by accident (See Sec. 7.3). This results in a total of 52 tag systems, so to say a tear-off calendar of tag systems, one for each week to guide us through the year ;-)

It is not our purpose here to build up a more philosophical theory of the notion of an experiment in mathematics, since we are convinced that it is far from unproblematic to isolate the notion of an experiment from other methods in mathematics. A very careful research would be needed here, and this is not the place to do this. Rather, we will assume that the things we have done with our computer to study tag systems can be considered as computer experiments, in the same way many mathematicians nowadays use this notion. Indeed, it has become rather common to talk about experiments in mathematics, the examples are legio. The new journal called *Experimental Mathematics,* founded in 1992, illustrates this point. We would like to start this chapter with a quote from the statement of the philosophy behind this journal, that beautifully summarizes our opinion about experiments in mathematics ([ELdlL92], p. 1):

> While we value the theorem-proof method of exposition, and while we do not depart from the established view that a result can only become part of mathematical knowledge once it is supported by a logical proof, we consider it anomalous that an important component of the process of mathematical creation is hidden from public discussion.[...] *Experimental Mathematics* was founded in the belief that theory and experiment feed on each other, and that the mathematical community stands to benefit from a more complete exposure to the experimental process. [...] The word "experimental" is conceived broadly: many mathematical experiments these days are carried out on computers, but others are still the result of pencil-and-paper work [...]

8.1 Purpose of the chapter

In this chapter 52 tag systems of which 50 were generated by algorithm 2 (See Sec. 7.4.2) will be studied. The reasons behind the experiments on these tag systems are manifold.

First of all, we want to understand better how good the algorithms described in Sec. 7.4 actually are to generate tag systems of which the behaviour is very hard, if not impossible, to predict. If it is possible to find experimental support for the fact that (some of) the tag systems generated by this algorithm are indeed very hard to predict, we have a clear indication, but of course not a proof, that this class of tag systems might contain unsolvable tag systems.

Secondly, we want to know whether it is possible to identify different classes of tag systems within the 52 studied here. Identifying different classes cannot only help to better understand tag systems, but might be a basic step to differentiate solvable from unsolvable classes. Furthermore, if we would be able to identify certain classes for the 52 tag systems to be considered here, the characteristics through which they are separated from each other could be used to further investigate the problem of determining “good” constraints for intractable tag systems.

Finally, the more general reason behind these experiments is to improve our understanding of tag systems. In trying to get a firmer grip on tag systems through these experiments, it is possible to build up a better intuition of these systems. Furthermore, building up such an intuition can help us to develop new approaches, methods or arguments for the more general theoretical results that we, ultimately, want to establish.

Before starting with a description of the experiments and their results, we will first consider some of the restrictions.

8.2 Some further restrictions

8.2.1 On the programming language used.

If somebody asks me what programming language I use right now, I always feel a bit ashamed to admit that I use Visual Basic (VB), and actually merely the Basic part. The reason for this choice is very simple: when I started with this research I couldn’t program, and I followed the advise of my former supervisor to learn to program in Visual Basic. At that time the plan was to make extensive use of computer visualizations, and he considered VB as the best language to do this. When this advise was given to me, I didn’t have a windows PC, but a Mac so I downloaded a Basic compiler which runs under MacOS 9. Then I went through a textbook on Basic and this was basically how I started to program. Later, when I knew a bit more about programming, it was too late to learn another language *and* use it for the experiments. In having read some books and papers by Chaitin I learned a bit of Lisp, and then spent some time on Scheme. Scheme is really completely different from Basic since it is a functional programming language. I liked it very much, and the idea of switching to Scheme became very tempting. However, liking one programming language more than another is not a good reason to make such a switch. Furthermore, since it was Basic and not Scheme I was used to, programming in Scheme was too time-consuming, and I never had the time to learn it in a decent way.[2] I thus stuck to VB, being aware of the fact that this was not really a well-considered decision. The way I have programmed the last two or three years can only be called pragmatic: learning the basics of Basic so I would be able to do would I wanted to do. If I would have known then what I know now, I would probably first have taken a closer look at several different programming languages, taking into consideration the speed with which one can program in the language,[3] the speed of the programs themselves and the specific purpose the language will be used for.

Despite the more practical reasons behind my choice for VB, it should be noted that the language has two features I like very much. First of all, contrary to C, it is possible to interrupt the code while it is running (it works with an interpreter). This is a very useful, if not necessary, feature in doing computer experiments. Secondly, and this concerns the Basic part of VB, it is a very accessible language for people who have never programmed. In a way you can almost directly write what you want to implement. But maybe this last feature is biased by the fact that Basic was the first language I learned.

8.2.2 Size of Sample space vs. computation time

Another important restriction on the experiments is the size of the samples. First of all, we merely study 52 tag systems, although there is an infinite class of tag systems for which $\mu = 2$, $v > 2$. Secondly, in all experiments limits have to be imposed somewhere. To give some examples, in experiment 1, we will only test 2000 initial conditions for each of the tag systems. In experiments 2 to 6, we merely used 10 (sometimes 20) initial conditions from the 2000 tested.

The fact that one has to impose a limit somewhere is a common feature of any computer experiment, since the objects studied are often infinite, while the

[2] I went through the first chapters of [AS96], a book I would highly recommend to anyone who wants to program in Scheme.

[3] E.g. while *Brainfuck, An Eight-Instruction Turing-Complete Programming Language* is from a certain point of view an attractive language, based as it is on register machines, programming in this language – as might be clear from its name – is a very time-consuming business. Brainfuck is available at: http://esoteric.sange.fi/brainfuck/

computer is finite. As long as one does not find a method for reducing an infinite class to a finite number of cases (e.g. through a theorem),[4] one is obliged to keep in mind, that one is only collecting heuristic material, collected within self-imposed, sometimes rather arbitrary limits.
Still, the limits chosen here are relatively low. Since my time, and that of my computer, was limited however, I could not allow myself to choose for higher limits. This might lead to a wrong interpretation of the results. For example, suppose that we would be able to find classes, it might be that there is some other kind of class of tag systems with $\mu = 2$, $v > 2$ which is much more interesting but not considered because a tag system from that class was not generated? In our interpretation of the results we have been as careful as possible, always taking into account the fact that a larger sample space might lead to other results. One should not forget though that as large a sample space might be, one can never be sure about the results. It is only when one has found a rigorous proof that certain results can be generalized. This is in the end the reason why we are talking about experiments and not proofs.

8.2.3 Focus on 2-symbolic tag systems

> The characteristics of [certain] results is that they often lie at a relatively low level in the overall canvas of intellectual functions, a level often dismissed with contempt by those who purport to study "higher, more central" problems of intelligence. Our reply to such criticism is that low-level problems probably do represent the easier kind, but that is precisely the reason for studying them first. When we have solved a few more, the questions that arise in studying the deeper ones will be clearer to us.
>
> David Marr, 1976.[5]

As is clear from the example of **T1**, the tag system Post described, there are 2-symbolic tag systems, with $v > 2$, for which it is by no means trivial to prove their solvability. The fact that the little research that has been done on **T1** has

[4] Only then it also becomes possible to consider the possibility of using the computer for actual proofs and not only for collecting, comparing, analyzing... data.

[5] [Mar76], p. 3

not lead to any result pointing into the direction that it might be solvable, can be regarded as an indication of the intractability existing on this level. As far as I know, **T1** is still not known to be solvable.

In what follows, the results to be discussed will be restricted to this class of tag systems, their seeming intractability of course being a precondition for this focus. There are several reasons for this restriction. First of all, there is of course **T1**, the tag system mentioned by Post. In starting to study tag systems, this was the one I started from, and until today, I still have more questions than answers as far as **T1** is concerned. Given the existing difficulties on this level, I consider it important to try to understand the smallest tag systems not known to be solvable first, before passing on to larger systems, presupposing that what I can learn from these systems, might help to understand the problems that arise in studying the more general class of tag systems better.

Of course this does not mean that there are no significant differences between 2-symbolic tag systems and μ-symbolic tag systems ($\mu > 2$). Indeed, as the alphabet of a tag system becomes larger, predicting the outcome of a tag process becomes harder and harder. If one e.g. has 10 symbols, and the words assigned to each of these symbols have different lengths, there will be a larger variety of ways in which the lengths of the strings produced shrink and grow. Furthermore, in having more symbols it becomes easier to encode certain functions over the integers, as was clear, e.g., from Minsky's proof of the general unsolvability of tag systems. Despite these differences, I have still not been able to detect a more fundamental (proven) difference, one which makes it possible to conclude that there can be no universality and thus unsolvability at the level of 2-symbolic tag systems, with $v > 2$, although reducing a μ-symbolic tag system, with $\mu > 2$ (even in the case where $\mu = 3$) to a two-symbolic tag system is far from trivial.[6] As long as there is no proof of the solvability of the class of 2-symbolic tag systems, they remain an interesting class to study.

A further reason for confining ourselves to 2-symbolic tag systems, is that they are the class of smallest possible tag systems not known to be solvable i.e. they lie at the lower boundary of the area between small tag systems known to be

[6] So far we have been unable to find a general procedure for this.

solvable, and classes of large tag systems known to be unsolvable. Studying such systems, systems on the edge of solvability, can help to bridge the gap between solvability and unsolvability.

To summarize, since the results to be discussed in the following sections are restricted to 2-symbolic tag systems which were generated by using the constraints described in the previous chapter, one should be very careful in making generalizations. Before making such generalizations, one should first of all study, to a more exact extent, in what ways 2-symbolic differ from μ-symbolic tag systems. For now, we will keep our eyes focussed on the so-called easiest, though possibly unsolvable, tag systems.

In total we have performed 6 different experiments. Because of the fact that the description of the set-up and the analysis of the results from the experiments is very lengthy as compared to the actual conclusions that can be drawn on the basis of each of the experiments, we will not deter the reader by describing every one of the experiments in all its details. Instead we have chosen to only include the details of experiment 1 and 2, since the (direct and indirect) results from these experiments are the more interesting and have laid the basis for the remaining 4 experiments. As for these 4 experiments, we will only include a summary of the conclusions drawn on the basis of these experiments. The interested reader is referred to Appendix C for a detailed description.

> Post found this (00, 1101) problem "intractable", and so did I even with the help of a computer. Of course, unless one has a theory, one cannot expect much help from a computer (unless *it* has a theory) except for clerical aid in studying examples; but if the reader tries to study the behaviour of 100100100100100100100 without such aid, he will be sorry.
>
> Marvin Minsky, 1967.[7]

> Although tracing individual tag systems, cannot answer the deepest of questions about the nature of systems, it is a useful way of gathering information about them. A computer, even one without theories, can be of much assistance in this information-gathering.
>
> Brian Hayes, 1986.[8]

8.3 Experiment 1: Distribution of general classes of behaviour in 2-symbolic tag systems.

The purpose of the first experiment is to get an idea of the distribution of the general classes of behaviour for the tag systems considered when started with arbitrary initial conditions. The experiment counts how many times one such tag system leads to periodicity, termination, unbounded growth or behaviour which cannot be classified in one of these three classes after 10000000 iterations. The initial conditions from this last class were then used in some of the other experiments.

8.3.1 Set-up of experiment 1

The program used for the experiment tests 999 random initial conditions of length 300.[9] One could object to this arbitrary choice of this length, since the

[7] [Min67], pp. 267–268

[8] [Hay86], p. 22

[9] The choice for 999 initial conditions and not 1000 might seem rather strange, and, actually it was my purpose to use 1000. However, it was only after the code had been running for days that I saw there was a rather stupid bug in the code, and I thus decided not to rerun it. The

number of letters scanned in the initial condition is determined by the shift number v. Then, **T1**, with $v = 3$, would scan 100 letters, while **T2** would merely scan 50. Upon further consideration however, one quickly sees that this is not a problem at all, since a larger v also means that the average length of the words added is larger.

In order to decide what kind of behaviour a tag system leads to, when given one of the initial conditions, the description of the procedure that tests constraint 6 was used (See 7.3.5). If an initial condition is found for which the tag system becomes periodic, a counter which keeps track of the number of initial conditions leading to periodicity, is added with one. The same is done for any other kind of behaviour. If an initial condition leads to a string longer than 15000 letters, it is classified as a possible case of unbounded growth, indicated as “growth?”.[10] The class of initial conditions which do not lead to termination, periodicity or Growth?, are classified as “Immortals?”. It is the existence of initial conditions for which it is far from clear what will happen to them that indicates the intractable behaviour of the tag system. Indeed, if we could find a procedure that decides for a given tag system that it will either lead to unbounded growth, become periodic or halt, it would be solvable and, as a consequence, tractable. The fact that a tag system is able to run for 10.000.000 iterations without leading to one of these classes of behaviour, is considered as an indication of its intractability.

If a classification has been made for a given condition, the program outputs the initial condition that led to the behaviour, for each of the cases in a different file. In case of periodicity, the last string produced and the length of the period are also written to a file.

The program also uses a counter keeping track of the number of iterations which have already been performed. This information is not only used to simply know when 10000000 iterations have been performed, but also to know, for

reason of the bug was that a new tag system was tested if the counter of the initial condition was not > 1000, but ≤ 1000, while the counter was always set to 1 (and not 0) for every new tag system tested.

[10] The problems described in 7.3.5 resurfaces here. The “?” is meant as a reminder of the problematic determination of “growth”.

each 10000 iterations, how many strings are still in the running as possible Immortals?. If an initial condition has led to periodicity, Growth? or termination, the number of iteration steps already performed is divided by 10000. The resulting integer part i of this number determines the interval $[i \cdot 10000, (i+1) \cdot 10000[$, where $0 < i \leq 1000$. For each of the intervals, a counter keeps track of the number of times an initial condition has led to one of the three classes of behaviour, for that interval. If periodicity is detected, one must be careful in adding one to the counter of a specific interval. Indeed if, for example, a period 200 is detected at step 70100, the actual point from which the system has become periodic, does not lie in the interval $[\frac{70000}{10000}, \frac{80000}{10000}[$. In the worst case, the system has already become periodic at step 69801. It is possible to detect the exact moment when the system has become periodic, by going back to the last reference string (i.e., in the example the string produced after 70000 iterations) and comparing successive pairs of strings separated by an interval having the length of the period. However, while exact, it would lead to time-consuming computer processes. Since it already takes days of calculation now to test all the tag systems, a more non-exact adjustment is made: if period p is detected at step n, the counter of the interval which will be increased is determined as follows: $n - (p + \frac{p}{2})$.

In keeping track of the number of initial conditions for each interval of iterations that has led to one of the three general classes of behaviour, one can then build up an idea about how fast initial conditions lead to one of the three classes of behaviour for each of the tag systems.

The experiment was performed twice, with two different sets of 999 initial conditions of length 300. This was done in order to check whether the numbers found during the first run for each of the classes are “representatively” indicative for the number of initial conditions that fall into each of these classes. I.e. it is a crude check on the statistical “representability”. If a second run of the experiment would result in large deviations from the original results, it is clear that the experiment should be run for a larger number of initial conditions in order to get a good estimate of the probabilities for each of the classes of behaviour.

8.3.2 Discussion of the results

In the following table one finds an overview of the number of times each kind of behaviour was found for the different tag systems, for the two runs of the experiment.

Table 8.1: Number of initial conditions that halt, become periodic, possibly lead to unbounded growth (Growth?), or cannot be classified in neither of these classes after 10000000 iterations (Immortals?).

Tag System	**Halts**		**Periodics**		**Immortal?**		**Growth?**	
T1	188	170	790	808	18	19	3	2
T2	0	0	891	897	108	102	0	0
T3	0	0	893	886	106	113	0	0
T4	0	0	926	918	73	81	0	0
T5	0	0	982	974	17	25	0	0
T6	953	954	9	10	37	30	0	5
T7	0	0	989	992	10	7	0	0
T8	0	0	968	961	17	27	14	11
T9	0	0	958	959	40	40	1	0
T10	0	0	973	952	26	43	0	4
T11	0	0	951	960	48	39	0	0
T12	962	955	11	7	26	31	0	6
T13	0	0	981	983	18	16	0	0
T14	0	0	950	952	49	47	0	0
T15	0	0	975	977	24	22	0	0
T16	0	0	979	990	15	8	5	1
T17	0	0	979	975	20	24	0	0
T18	0	0	972	968	25	28	2	3
T19	0	0	976	971	19	26	4	2
T20	0	0	833	878	166	121	0	0
T21	0	0	976	983	23	16	0	0

Continued on next page

Table 8.1 – continued from previous page

Tag System	Halts		Periodics		Non-periodics		Growth?	
T22	0	0	639	664	317	300	43	35
T23	0	0	958	973	41	25	0	1
T24	0	0	977	969	20	25	2	5
T25	0	0	987	993	12	6	0	0
T26	0	0	962	931	37	68	0	0
T27	0	0	983	987	16	12	0	0
T28	0	0	985	981	12	12	2	6
T29	0	0	952	943	47	56	0	0
T30	0	0	977	977	22	22	0	0
T31	0	0	965	971	34	28	0	0
T32	0	0	849	840	150	159	0	0
T33	0	0	964	970	35	29	0	0
T34	0	0	912	911	85	80	2	8
T35	0	0	954	942	30	39	15	18
T36	0	0	938	927	61	72	0	0
T37	0	0	817	819	182	180	0	0
T38	0	0	975	981	24	17	0	1
T39	0	0	983	988	10	7	6	4
T40	0	0	934	932	64	67	1	0
T41	828	807	152	172	11	11	8	9
T42	0	0	927	923	72	76	0	0
T43	0	0	978	981	19	10	2	8
T44	0	0	976	968	23	31	0	0
T45	0	0	983	978	14	18	2	3
T46	0	0	955	946	44	53	0	0
T47	546	521	427	458	13	14	13	6
T48	0	0	981	985	18	14	0	0
T49	0	0	946	946	52	52	1	1
T50	0	0	921	945	78	54	0	0
T51	0	0	926	917	72	76	1	6

Continued on next page

Table 8.1 – continued from previous page

Tag System	**Halts**		**Periodics**		**Non-periodics**		**Growth?**	
T52	0	0	977	968	22	31	0	0

The first thing to be noted in observing the results from the table, is that there are no large deviations between the first and the second run of the experiment. This serves as an indication of the fact that the numbers found give a good estimate of the distribution of the several classes of behaviour for each of the tag systems.

There are some results in the table that immediately catch the eye. First of all, the number of tag systems for which the number of halts is greater than 0 is very low. Indeed, only 5 of the 52 were run with an initial condition leading to termination, i.e. **T1**, **T6**, **T12**, **T41**, and **T47**. Furthermore, for those tag systems that lead to halts, the number of halts is rather high.

Given this low number of tag systems for which a certain number of initial conditions tested leads to a halt, and the fact that, if this is the case, the number of conditions that halt is relatively large, one wonders whether these other tag systems can actually ever lead to a halt. It can indeed be proven for each of the remaining 47 tag systems that they will never halt, except for a very specific class of initial conditions. So how will we prove this? Let us first consider **T2**. Clearly, if **T2** is started with initial condition 0, it will halt. It can be proven, however, that for any other initial condition it will never halt. This is the case because once $w_0(= 00101)$ or w_1 is produced, *and any of its letters is scanned* by the tag system, a halt can never occur. First of all, it should be noted that as long as the tag system produces a string that contains at least one time w_1 it can never lead to a halt, since its length is larger than $v = 6$. As a consequence, w_1 can never lead to the production of the empty string ϵ, the tag system either produces w_0, w_1 or $w_1 w_0$ from w_1. The only way to induce a halt, i.e., the production of the empty string ϵ, in this tag system is through w_0. However, the only way for w_0 to lead to a halt is that w_0 is entered with a shift such that none of its letters will be scanned, i.e. when the last letter scanned before the tag system moves to

w_0 is the letter that precedes w_0. This letter is either equal to 1 or to 0.[11] If it is equal to 0, this means that the word preceding w_0 is w_1 (= 1011010). Now, if the last letter in w_1 is scanned, its first letter will also have been scanned, and this is equal to 1. Thus, we can conclude that if w_0 is preceded by w_1, and none of its letters are scanned, the substring $w_1 w_0$ must lead to the production of $w_1 w_0$. If w_0 is preceded by w_0, and none of the letters in the second w_0 are scanned, the last letter scanned must have been equal to 1, and $w_0 w_0$ can thus neither lead to the production of the empty string ϵ. From this, it follows that **T2** can never halt, except when started with the initial condition equal to 0. We have thus proven the following theorem:

Theorem 8.3.1 *The first form of the problem of tag, i.e. the halting problem for tag systems, is solvable for **T2**.*

For all other 47 tag systems from the table in which no halt has been detected, except for **T13**, it can be proven that once these systems produce $w_1 w_0$ or $w_0 w_0$, they can never halt, by applying a similar kind of reasoning used for proving the theorem. As far as **T13** is concerned, it can be proven that once $w_1 w_0$ is produced, it can never halt. In other words, **T13** will only halt for initial conditions in which no letter 1 is scanned. We can thus prove the following theorem

Theorem 8.3.2 *The first form of the problem of tag, i.e. the halting problem for tag systems is solvable for all tag system from table 8.1 for which none of the initial conditions tested has led to a halt.*

We have checked the theorem for each of these tag systems but will spare the reader the details of the proof. The reader who is not convinced, can easily check the theorem by working out by hand each of the tag systems considered in the theorem.

Besides the fact that there are only a few tag systems that can lead to a halt, there is also a clear differentiation between the number of Immortals? Indeed, their number varies between 10 and 317. The number of Growths? is for all the tag system very low or even equal to 0, although it should maybe be noted that

[11] Remember that the shift number $\nu = 6$ for **T2**.

the tag system **T22** with the largest number of Immortals? has the largest number of Growths?. One could thus suspect that certain of the initial conditions classified as Immortals? are actually non-detected cases of growth. Some of the results to follow, especially those from experiment 4, will help to exclude this possibility. As far as the variety in the number of Immortals? is concerned it is hard to draw any conclusion. Although it is clear that the probability that one will find initial conditions that do not lead to one of the three general classes of behaviour after 10000000 iterations, can be relatively high or relatively low, depending on the tag system one is using, this does not allow to make any clear distinctions between the different tag systems.

The results from the experiment measuring how fast initial conditions lead to one of the three classes of behaviour for each of the tag systems, gives us some more information. We have made plots showing the number of iterations (using the intervals) against the number of initial conditions that have not (yet) led to one of the three general classes of behaviour for a given interval of iterations. What very much surprised me in these plots, is that there are clear differences between the several tag systems, if one considers the rate of decay of the number of initial conditions that have not yet resulted in behaviour of one the three classes after *n* iterations. We will not show all the plots here, because that would ask for a rather huge number of pages of plots interrupting the text, but we have printed all the plots in Appendix B. Here we will only show some of the plots that clearly differ from each other.

The plots from **T1**, **T2**, **T3** and **T13** result in a kind of hyperbolic shape. The plots suggest that the curves' lower leg converges to an axis parallel or identical to the X-axis, the larger the number of iterations. Of course, it is far from surprising that the larger the number of iterations, the fewer initial conditions are left that have not yet led to one of the three classes of behaviour. What is surprising is the shape of the plot: it suggests that the speed with which the number of left-overs decreases, is at first very high but then becomes very low. To put it differently, as fast as the number of left-overs decreases at first, as slow it decreases from a given point on.

In **T13**, the switch from fast to slow decrease is very abrupt, i.e. the transition between the upper and lower leg nearly coincide with a straight angle. For **T1**,

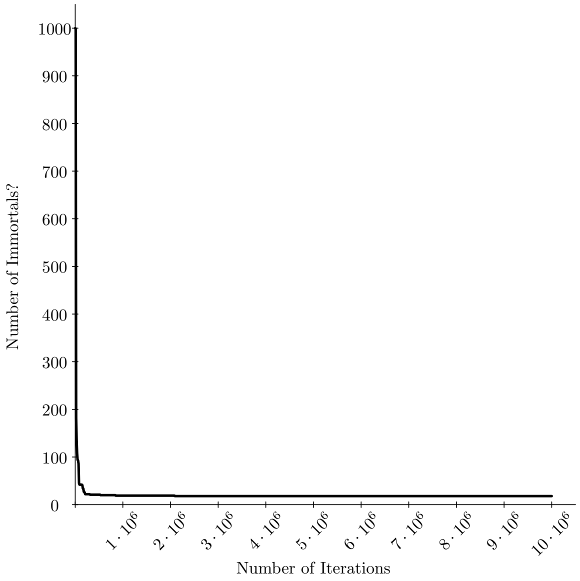


Figure 8.1: Plot of **T13**

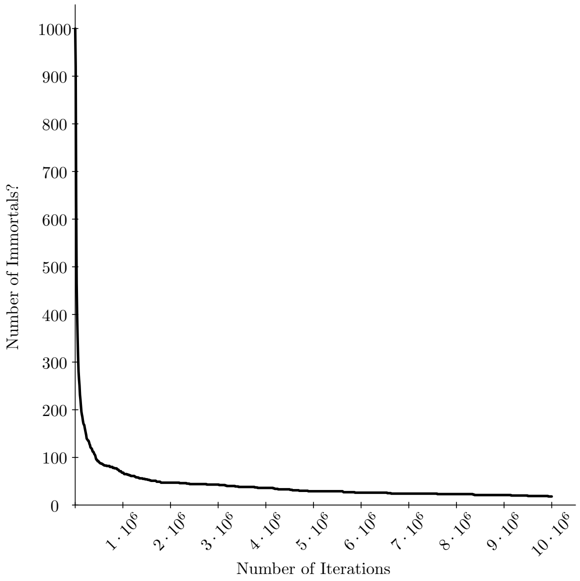


Figure 8.2: Plot of **T1**

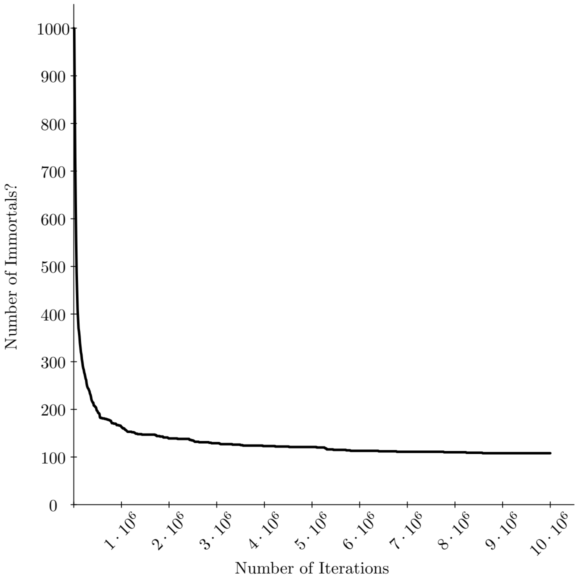


Figure 8.3: Plot of **T2**

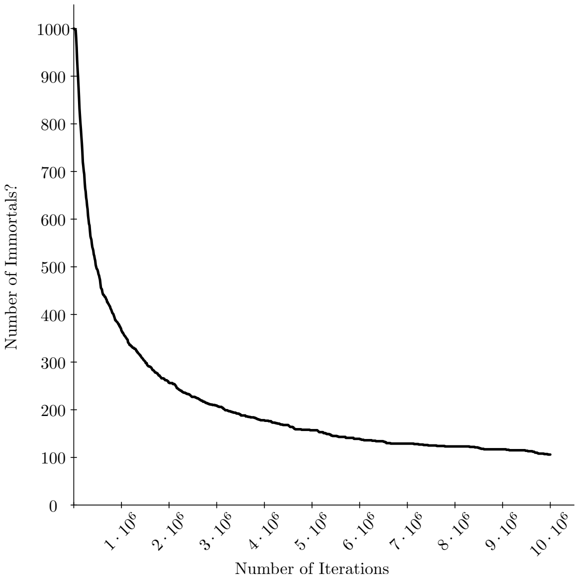


Figure 8.4: Plot of **T3**

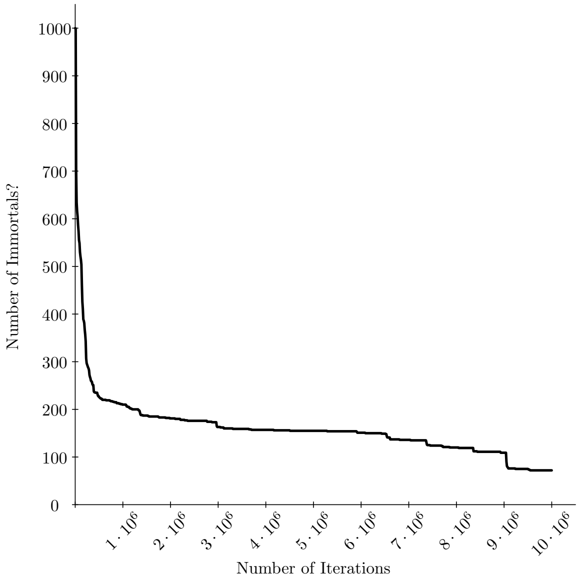


Figure 8.5: Plot of **T51**

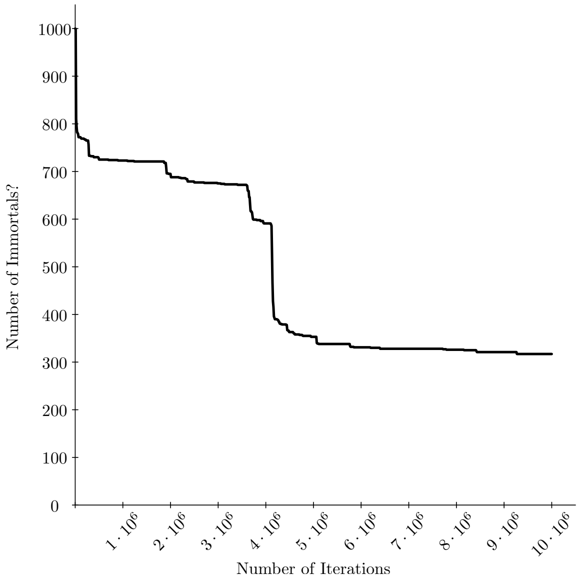


Figure 8.6: Plot of **T22**

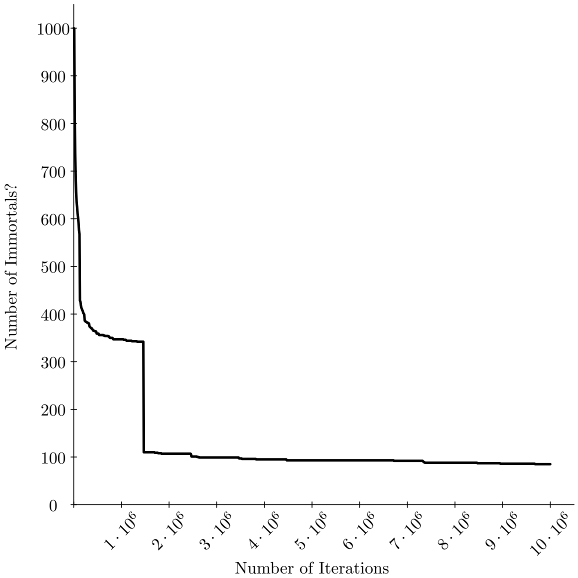


Figure 8.7: Plot of **T34**

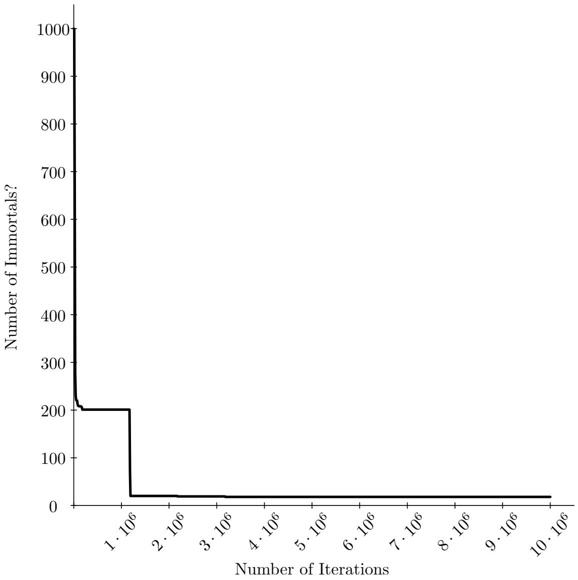


Figure 8.8: Plot of **T48**

this switch is still rather abrupt, but the transition from upper and lower leg is more continuous. This is the same for **T2**, but the lower leg does not approximate the X-axis, but a line parallel to it. The plot of **T3** is similar to that of the other three tag systems, but the change between fast and slow decrease is far less abrupt. These four plots are more or less continuous, where the small discrete transitions in the plots can be explained by the size of the intervals and the fact that the identification of the right interval for initial conditions that have become periodic is merely an approximation and must contain some errors.

The plots, however, of **T22**, **T34** and **T48** are far from continuous. Indeed, there are clear discrete transitions that can no longer be explained by the size of the intervals, or the possible errors induced by the determination of the intervals for initial conditions that become periodic. The plots of **T34** and **T48** show one large discrete transition. In case of **T34** the plot is at first very similar to that of **T2**, but then the number of left-overs suddenly drops, somewhere in an interval between $1 \cdot 10^6$ and $2 \cdot 10^6$. Of course, if we would make our intervals smaller, this transition might become less discrete, but still the fact remains that the number of left-overs initially decreases very fast. Then, the speed slows down in a way similar to the plots for **T2**, but after a few more intervals, the speed again becomes very fast, the plot suddenly dropping from about 350 to about 100 left-overs. The speed then again slows down but given the first discrete transition, one suspects that more might happen. Similar observations hold for the plot of **T48**, but the transitions seem to be even more discrete. For the case **T22** there is not one large discrete transition but many smaller ones. Again one cannot explain this by the size of the intervals or the possible errors caused by the determination of the intervals for initial conditions that have become periodic. The plot of **T51** is shown because it is difficult to decide whether the discrete irregularities are due to the set-up of the experiment or are inherent to the tag system.

So what can one conclude on the basis of these plots? Since we are dealing with experimental results, any conclusion can only be heuristic and has to be checked through further experimentation. Still, on the basis of the plots, one can make some inductions. The basic question to be asked here is whether we can generalize the plots. Let us first consider the plots for **T1**, **T2**, **T3** and **T13**, and all other plots, shown in appendix A that are similar to one of these plots. Given the fact that these plots do not contain any serious discrete transitions, let us assume that such transitions will not occur for these tag systems, however far one would extend the plot to the right, or whatever the number of initial conditions tested. This implies that the larger the number of iterations becomes, the number of left-overs will more and more converge to a constant value a. The question then is whether the plot merely converges to the line $y = a$, intersecting it at an infinite point, or, whether an intersection will occur at a finite point, i.e., the question of whether the number of left-overs ever becomes equal to a (or, if $a > 0$, smaller than a)? Indeed, since the number of initial conditions tested is finite, and, for now we have no decision procedure for the tag systems considered, there is no way to exclude one of these two possibilities, except if one would wait until all conditions tested have led to one of the three classes of behaviour. But of course, this is exactly what is at stake here.[12]

This problem becomes even worse if we take into account that we have merely tested 999 initial conditions of length 300. Since about $\lfloor 300/v \rfloor$ letters can be scanned given the shift number v, there are in fact about $2^{\lfloor 300/v \rfloor}$ possible initial conditions of length 300, for each of the tag systems considered. Supposing that we can more or less generalize the results from table 8.1, for those tag systems with plots similar to that of **T1**, **T2**, **T3** and **T13**, an estimate for the results, if all conditions of length 300 would be tested, is given by the multiplication of each of the numbers in the table by a factor of about $2^{\lfloor 300/v \rfloor} - 2^{10}$ (The subtraction of $2^{1}0$ results from the fact that 2^{10} approximates 999). This implies that the number of Immortals? will grow linearly with the size of the sample space,

[12] Indeed, suppose we would take the 18 left-overs from Post's tag system and see what has happened to them after 100.000.000 iterations. Maybe all of them will have halted or become periodic, maybe not. Suppose only 9 are left once we have performed 100.000.000 iterations, and we then take these 9 and see what happens to them after 1.000.000.000 iterations. Again, all of them might have led to periodicity or a halt, maybe not. As is clear this kind of reasoning shows what problem is involved here: the only way to solve it is to wait and see, but there seems to be no way to predict how long one should wait.

which grows exponentially with the length of the initial condition.[13]

If we would make a plot after having tested all possible conditions of length 300 for each of the tag systems, the lower leg of the hyperbolic shape would lie significantly higher (with the same limit to select Immortals? at 10.000.000 iterations). This reasoning implies that the more initial conditions one tests, the slower the number of left-overs converges to a constant, possibly 0, and the further the point of intersection, if finite, moves to the right. Furthermore, given the presumed exponential growth of the number of Immortals? with the length of the initial condition, one cannot but conclude – on assumption that the plots can be generalized – that this intersection point, if finite, moves exponentially fast to the right.

The basic question to be asked is whether any such plot converges to the a a line $y = a$ at a finite or at an infinite point. If we would be able to prove that for every class of initial conditions of arbitrary length l, this intersection point is finite for a given tag system, we would have proven its halting and reachability problem. For now however, there is no clear method to prove this for the tag systems considered here. The fact that, if we assume that we can generalize the plots, this intersection point moves exponentially fast to the right for increasing l serves as a clear indication of the difficulties that might be involved here, i.e., it illustrates the intractability (though possibly not inherent) of these tag systems.

We still have to consider the tag systems for which the plots are similar to those for **T22**, **T34** and **T48**, i.e., plots with clear discrete jumps. These cases illustrate that generalizing the plots from the other tag systems might be tricky. Indeed, for now nothing guarantees that such drops might not occur for the other tag systems if one would consider a larger sample space of initial conditions and

[13] We should point out here that one should of course take into account that there are always v different lengths of classes of initial conditions for which the tag system will scan the same relevant letters. For example, the relevant letters scanned in the class of initial conditions of lengths 298, 299 and 300 will all be the same for Post's tag system. This, however does not mean that given an initial condition of length 299 and length 300 for which the relevant letters are identical, that the tag system will lead to the same behaviour. On the contrary, because the sequence of relevant letters in the respective strings produced in the two cases from the initial condition will be different, given this length of the condition.

follow the lower leg of the plots further to the right. Still, sudden drops in these plots do not imply tractability. Indeed, as is e.g. clear for **T34**, after the large discrete jump in the plot, the lower leg seems to become more continuous. For now, we have not investigated these tag systems to an extent allowing to conclude that plots with or without discrete transitions can be explained by some structural property that makes it possible to differentiate between these two classes.

8.4 Experiment 2: Periodicity in the different tag systems

> Here again was the untiring calculator who blazed the way into the unknown. Gauss set up huge tables: of prime numbers, of quadratic residues and non-residues, and of the fractions $1/p$ for $p = 1$ to $p = 1000$ with their decimal expansions carried out to a complete period, and therefore sometimes to several hundred places! With this last table Gauss tried to determine the dependence of the period on the denominator p. What researcher of today would be likely to enter upon this strange path in search of a new theorem?
>
> Felix Klein, 1979.[14]

The second experiment to be discussed here uses data from Experiment 1, viz., the initial conditions that led to periodic behaviour. The purpose of this experiment is to answer empirically certain questions concerning the possible types of periods produced by tag systems.

In the discussion of Watanabe's [Wat63] (See 6.1.2) we described several aspects of the kind of periods produced in Post's tag system **T1**. Among other things, it was shown, through Shearer's proof [She96], that it is possible to generate all even numbers by a specifically configured combination of two different periodic strings, namely 2 and 4. Given such proof, one is tempted to ask whether one can generalize the method of the proof to the other tag systems investigated here, possibly involving other sequences of numbers e.g. the uneven

[14] [Kle79], quoted in [Gre82], p.5

numbers. In other words, is it the case for any tag system satisfying the constraints from Sec. 7, that, given a certain infinite sequence of numbers P, one can always construct a periodic string S for any $p \in S$, such that S has period p? The answer to this question is negative. It can for example be proven for **T2** that it is impossible to find at least two periodic strings of the type needed for Shearer's proof to work.

For Shearer's proof to work, we have to find at least two periodic strings A and B in **T2** which, when all their relevant letters have been processed, reproduce themselves after a different number of steps. Furthermore, A cannot simply be B repeated x times and vice versa, i.e. A and B must lead to different periods. A further restriction is the fact that the length of A and B must be divisible by v. This is necessary to guarantee that the shift A and B are entered with remains constant each time.[15] Suppose for example that $v = 3$, $l_A = 7, l_B = 10$, and that A and B self-reproduce themselves after all their relevant letters have been scanned. Let us further suppose that in starting from AB scanning all the letters in A, starting from the first letter in A, results in a new string BA, where B is entered with a shift 2, i.e. the additive complement of $7 \mod v$.[16] Now, scanning all the relevant letters of B, A will not be entered with a shift 0, but with a shift 1. Similarly, after all the relevant letters of A have been scanned, B will now be entered with a shift 0. In other words, if the lengths of the periodic strings used are not divisible by v, the shifts the A's and B's are entered with cannot remain constant. In this respect, for Shearer's proof to work, we need periodic strings with a length divisible by v. One could counter this restriction by saying that 10111011101000000 is also a periodic string in **T1**, although its length is not divisible by v. This is true. However, scanning all the letters of this string results in 110111011101000000, which has a length divisible by v. It is only, because 110111011101000000 is tagged to 10111011101000000 that 10111011101000000 results in 10111011101000000. In other words, if one wants to e.g. generate all the even numbers by concatenating a period 2 x times to a period 4, it might be the case that the initial length of the period 4 string is not divisible by v. However, one can only get the desired result if every periodic substring produced in the process is entered with the same shift. This is only possible if all these substrings have a length divisible by v. In the following, it is thus assumed that if one wants to apply Shearer's proof to another tag system, the two periodic strings A and B needed will have a length divisible

[15] A more exact definition of this type will be given on p. 390.

[16] See Sec. 6.3.1, 320.

by v, although it is possible that the periodic string (A or B) standing at the beginning of a concatenation of these two periodic strings is entered with a shift different from 0.

Now, it can indeed be proven for **T2** that the only possible periodic string which follows these restrictions is a period 2 string. In the following table, the strings resulting from every 2-combination of w_0 and w_1, in every possible shift (0 to 5) are shown. If the resulting string is *not* the same as the combination it is produced from, it is marked with $\checkmark$. If a string is not marked, there are two possibilities: the resulting string has a length which is or is not divisible by v. If it is not divisible by v, the additive complement of the remainder is mentioned.

	$w_1 w_0$	$w_0 w_1$	$w_0 w_0$	$w_1 w_1$
0	$w_1 w_0$	$w_0 w_0 \checkmark$	$\overline{\lvert w_0 w_0 \rvert \bmod v} = 2$	$w_1 w_0 w_1 \checkmark$
1	$w_1 w_1 \checkmark$	$w_0 w_1$	$w_0 w_1 \checkmark$	$w_0 w_1 w_0 \checkmark$
2	$w_1 w_0$	$w_1 w_1 \checkmark$	$w_1 w_0 \checkmark$	$w_1 w_0 \checkmark$
3	$w_0 w_0 \checkmark$	$w_0 w_0 \checkmark$	$w_1 w_1 \checkmark$	$\overline{\lvert w_1 w_1 \rvert \bmod v} = 1$
4	$w_1 w_1 \checkmark$	$w_1 w_1 \checkmark$	$w_1 \checkmark$	$w_0 w_1 \checkmark$
5	$w_0 w_0 \checkmark$	$w_1 w_0 \checkmark$	$w_0 \checkmark$	$w_1 w_0 \checkmark$

If a string is marked it follows that the string it is produced from is not self-reproducing when entered with the given shift Sx. From the table it is clear that from the 4x5 possibilities considered, 15 do not lead to self-reproduction. There are thus 5 possibilities left that might lead to the kind of periodic strings we are searching for: $(0, w_1 w_0), (2, w_1 w_0), (1, w_0 w_1), (0, w_0 w_0), (3, w_1 w_0)$, where (s, c) is the result from entering combination c with shift s. The first three of these couples are the three possible period 2 strings for **T2**, since they have a length divisible by v. Concatenating the first couple any number of times, always results in a string of period 2. The same goes for the other two.

But what about $(0, w_0 w_0)$ and $(3, w_1 w_1)$? Dividing the lengths of both combinations results in a remainder $r > 0$. This remainder r determines the shift the next word tagged to $w_0 w_0$ or $w_1 w_1$ will be entered with i.e., the first $\overline{r}$ letters of this word will be erased (it will be entered with a shift $\overline{r}$). Now the only possible way for $w_0 w_0$ or $w_1 w_1$ to become periodic so that Shearer's proof can be applied to **T2**, is that it should be possible to concatenate certain combinations of words to either $w_0 w_0$ or $w_1 w_1$ such that the resulting string is periodic. This possibility is excluded by looking at what happens to $w_0 w_0$ and $w_1 w_1$ if we look at

what happens in concatenating any of the 2-combinations of w_0 and w_1 to either $w_0 w_0$ and $w_1 w_1$. For the case $w_0 w_0$, whatever of these 2-combinations is concatenated to it, the 2-combination will be entered with a shift 2. The result of entering each of these possible combinations with this shift 2 can be read from row 3. Three out of four will not lead to reproduction. Concatenating $w_1 w_0$ to $w_0 w_0$, however, does lead to reproduction. Does this mean we have found the two periodic strings necessary for Shearer's proof to work? No. Suppose we want to generate the number 8 by concatenating 3 times $w_1 w_0$ to $w_0 w_0$. After all the relevant letters of this string have been scanned we will have the same string, but now, it will be entered not with the necessary shift 0, but with a shift 1. Indeed, because the length of $w_1 w_0$ is divisible by v, the next time $w_0 w_0$ is entered $w_0 w_0$ will no longer reproduce itself. Since there is no other 2-combination, besides $w_1 w_0$ that leads to self-reproduction when entered with a shift 2, we cannot use $w_0 w_0$. Indeed, whatever 2-combination is concatenated to $w_0 w_0$ it cannot result in the kind of periodic string we need here. The same reasoning can be applied for the case $w_1 w_1$ if entered with a shift 3, by looking at the results in row 1. The details of this reasoning are left to the reader. There is still one possibility left to get the periods we need here. Since $(0, w_1 w_0), (2, w_1 w_0), (1, w_0 w_1)$ are the kind of periodic strings we are searching for, we still have to look at the following possibilities: $(0, w_1 w_0 w_0 w_1), (2, w_1 w_0 w_0 w_1)$,
$(0, w_0 w_1 w_1 w_0)$. As might already be clear now, neither can lead to the right periodic strings, because of the shifts. If $(0, w_1 w_0 w_0 w_1)$ is the case $w_0 w_1$ will not be entered with a shift 1, but with a shift 0, thus not leading to the desired result. The same goes for $(2, w_1 w_0 w_0 w_1)$ and $(0, w_0 w_1 w_1 w_0)$. We still have to consider the case for which only one word is tagged to any of the 2-combinations instead of another 2-combination, or a 2-combination is tagged at one of the words. Again it can be shown that no such 3-combination can lead to the kind of self-reproduction we are searching for. This follows from the fact that the two words w_0 and w_1 have a length that is not divisible by v. The details of the proof are left to the reader, the method to be used being similar to the one used for proving the result for 2-combinations. Since the only possible periodic strings that can be produced by **T2** of the kind needed for Shearers's proof to work are period 2 strings described, we thus conclude that we cannot apply the method

of the proof to **T2**.

Besides the fact that it is possible for certain tag systems to construct, for any number $x \in P$ – where P is a given infinite computable sequence of numbers – a periodic string with period x, another important feature concerning the periods in **T1**, already pointed out, is the structure of a period. Indeed, as was shown, the majority of the periods produced by **T1** have a very transparent structure, the majority of them being compositions of four basic periods, i.e., 2, 4, 6 and 10. Only two exceptions to the rule were found until now, namely a string with period $p = 44$ and one with $p = 66$. These periods not only lack any transparent structure, but the length of their structure is much shorter than the length of the period itself, the lengths of the structures varying between 13-15 ($p = 40$) rsp. 23-26 ($p = 66$).
Since we seem to be confronted here with two different types of periodic strings in **T1** one wonders whether there exist any other types of periodic strings. At least, that is the question we asked ourself. Furthermore, since Shearer's proof cannot be applied to **T2**, one also wonders whether such periodic types can be used to differentiate between classes of tag systems. The data generated by the experiment to be described here were, amongst others, used to further investigate these two problems.

8.4.1 Set-up of experiment 2

As was said, this experiment uses data generated by the previous experiment. For each of the tag systems from Table 8.1 we used one of the files generated in the previous experiment. This file contains 1) the initial conditions that lead to periodic behaviour, 2) the last periodic string produced, as well as 3) the length of the period.
We already know that Post's tag system allows for a great variety of different periods ranging over all the even numbers. This does not mean at all that if this tag system is run with several initial conditions each even number has an equal chance to be produced. On the contrary, period 6 has a clear dominance over the other periods, followed by period 10. This observation led me to the ques-

tion whether one can also find such dominance in other tag systems. In this respect, the experiment used the input files to store the different periods found for each tag system, as well as the number of times each period is produced. Then, for each period, its probability is calculated by dividing the number of times the given period appears by the total number of periodic string found.
A related question to be asked is how large the "variety" of periods produced is for the given sample. Indeed, given the dominance of period 6, one suspects that the variety of different periods produced by **T1** when starting with a certain sample of initial conditions, will be rather low. In order to measure the variety of different periods for a given tag system, the program also measured the ratio between the total number of different periods found and the total number of periods occurring (times 100 to get a percentage).[17]
As was said, another (theoretically more interesting) question I wanted to investigate is whether there exist other types of periodic structures than the two types found in Post's tag system, i.e. the "regular" periodic strings, like the period 6 strings, and the "irregular", like the period 40 string found. Maybe such new types – if they exist at all – can lead to another possibility for applying Shearer's proof, using other types.
In order to find at least an empirical answer to such questions, the periodic strings from each of the input files were taken as initial conditions for the tag system under study. Then if the period is x, the tag system is run for x iterations. If the sequence of resulting strings thus produced has not been found before, it is stored in a file. In this way it is possible not only to find the structures for each of the different periods, but also, if a period is found more than once, to see whether one period can have several structures. This was done, because in Post's tag system one single period can have several different structures. E.g. period 12 might be composed out of period 6 + 3 times period 2 or period 4 + 4 times period 2. The several structures found in the different tag systems were then further analyzed by hand.
But before passing on to the actual results it is important to emphasize again the problematical character of this experimental approach. Given e.g. a tag

[17] It should be mentioned here that the data used are those from the first run of experiment 1, and the size of the maximal sample is thus restricted to 989 periods (for **T7**).

system for which the analysis of the structures results in the conclusion that all periods are compositions of one or more basic periods of the "regular" type. If this is the case, one cannot conclude that all its periods will be of this form. Indeed as will become clear immediately, no period 40 nor 66 of the "irregular" type was found for Post's tag system in running experiment 1, although we know that they exist. The only way to exclude such possibilities is through theoretical reasoning, as was e.g. done for **T2** in the intermezzo. Of course one could make the sample file larger and larger, taking lots of computer time to strengthen certain hypotheses or conjectures. However, for now the sample file is too small to conclude for hypotheses or conjectures such as tag system x cannot generate periods of type y.

8.4.2 Discussion of the results

In the following table an overview is given of the different periods found for each of the tag systems. The last column gives an overview of all the different periods (put in bold) and their respective probabilities, resulting from the division of the number of times a certain period appeared in the sample by the total number of periods found. The second column (%D.P.) gives the measure of the variety of the periods, i.e. the ratio between the total number of different periods and the total number of periods occurring (times 100 to get a percentage). The third column gives the total number of periods (Tot.) found. Although these last results were already given in table 8.1, they are repeated here to make the comparison a bit more easy.

Table 8.3: Overview of the results from Experiment 2.

T.S.	%D.P.	Tot.	Periods and # of each period rel. to Tot. periods
T1	1,772	790	**6** (84.2), **10** (9.37), **28** (1.39), **36** (1.27), **34** (0.89), **22** (0.76), **46** (0.38), **16** (0.38), **40** (0.38), **20** (0.38), **32** (0.25), **54** (0.13), **14** (0.13), **70** (0.13)
T2	0,448	891	**168** (65.1), **2** (23.8), **5386** (8.53), **6704** (1.68)

Continued on next page

Table 8.3 – continued from previous page

T.S.	%D.P.	Tot.	Periods and # of each period rel. to Tot. periods
T3	1,679	893	**202** (39.5), **124** (29.1), **8** (12.2), **752** (4.7), **32** (4.37), **40** (3.14), **192** (3.14), **48** (2.13), **4** (0.34), **178** (0.34), **64** (0.22), **56** (0.22), **758** (0.22), **316** (0.22), **1686** (0.11)
T4	2,483	926	**48** (48.2), **24** (14.3), **32** (7.34), **8** (6.57), **44** (5.18), **56** (5.18), **52** (2.27), **412** (1.84), **528** (1.84), **64** (1.73), **60** (1.51), **68** (1.19), **2454** (0.65), **1634** (0.54), **1224** (0.54), **286** (0.43), **72** (0.43), **84** (0.32), **80** (0.22), **76** (0.22), **88** (0.11), **12498** (0.11), **40** (0.11)
T5	2,851	982	**4** (32.8), **16** (11.9), **34** (10.1), **18** (9.67), **20** (8.35), **26** (5.91), **22** (4.48), **14** (2.85), **30** (2.75), **38** (2.14), **24** (1.93), **28** (1.12), **32** (0.71), **864** (0.71), **10** (0.61), **36** (0.61), **40** (0.51), **50** (0.41), **46** (0.31), **42** (0.31), **52** (0.2), **60** (0.1), **56** (0.1), **74** (0.1), **84** (0.1), **44** (0.1), **48** (0.1), **8** (0.1)
T6	22,22	9	**6** (88.9), **138** (11.1)
T7	0,606	989	**72** (43.9), **2090** (36.1), **600** (16.2), **2** (1.92), **4440** (1.82), **362** (0.1)
T8	3,099	968	**6** (27.1), **18** (19.3), **48** (13.5), **54** (5.58), **24** (3.82), **30** (3.41), **90** (2.76), **60** (2.69), **12** (2.58), **66** (2.38), **78** (2.17), **36** (2.1), **45** (2.1), **84** (1.76), **42** (1.34), **72** (1.14), **33** (1.14), **96** (0.72), **39** (0.62), **276** (0.62), **27** (0.41), **21** (0.41), **9** (0.31), **120** (0.31), **126** (0.1), **57** (0.1), **13446** (0.1), **3** (0.1), **108** (0.1), **102** (0.1)
T9	1,356	958	**14** (55.1), **692** (18.7), **28** (5.74), **24** (4.91), **20** (4.28), **12** (3.44), **8** (2.4), **16** (1.88), **36** (1.77), **730** (0.84), **2134** (0.52), **40** (0.31), **32** (0.21)
T10	0,513	973	**268** (61.8), **20** (25.3), **46** (10.4), **572** (2.47), **376** (0.1)

Continued on next page

Table 8.3 – continued from previous page

T.S.	**%D.P.**	**Tot.**	**Periods and # of each period rel. to Tot. periods**
T11	2,628	951	**4** (36.8), **10** (33.4), **18** (3.79), **12654** (3.58), **22** (3.15), **6** (2.73), **14** (2.63), **40** (1.79), **16** (1.58), **20** (1.37), **32** (1.26), **12** (1.26), **24** (0.95), **34** (0.84), **38** (0.74), **30** (0.74), **42** (0.63), **26** (0.63), **222** (0.53), **28** (0.53), **50** (0.42), **48** (0.21), **44** (0.21), **58** (0.11), **46** (0.11)
T12	9,090	11	**6** (100)
T13	0,305	981	**2** (97.5), **3784** (1.83), **78110** (0.71)
T14	2,421	950	**48** (52.8), **2** (21.3), **84** (8.21), **850** (3.58), **100** (3.16), **68** (2.95), **40** (2.11), **24** (1.16), **132** (1.16), **640** (0.53), **164** (0.42), **180** (0.42), **116** (0.42), **148** (0.42), **16** (0.32), **72** (0.21), **298** (0.21), **80** (0.11), **88** (0.11), **52** (0.11), **212** (0.11), **292** (0.11), **56** (0.11)
T15	4,102	975	**48** (20.2), **588** (13.6), **16** (12.2), **60** (10.9), **14** (10.1), **44** (6.56), **56** (4.92), **72** (4.82), **36** (2.56), **32** (2.51), **24** (1.54), **84** (0.92), **64** (0.92), **96** (0.82), **1302** (0.82), **68** (0.72), **88** (0.72), **40** (0.51), **116** (0.41), **92** (0.41), **108** (0.41), **104** (0.41), **52** (0.31), **120** (0.21), **112** (0.21), **188** (0.21), **100** (0.21), **424** (0.21), **128** (0.1), **124** (0.1), **144** (0.1), **184** (0.1), **76** (0.1), **28** (0.1), **136** (0.1), **160** (0.1), **164** (0.1), **132** (0.1), **80** (0.1), **172** (0.1)
T16	0,817	978	**10** (81.3), **52** (7.36), **80** (5.83), **62** (3.78), **224** (0.82), **986** (0.51), **66** (0.31), **424** (0.1)
T17	1,021	979	**196** (94.3), **2102** (2.42), **2** (2.15), **72** (0.61), **3706** (0.31), **68** (0.2), **1140** (0.1), **80** (0.1), **454** (0.1), **1778** (0.1)
T18	0,514	972	**18177** (52.3), **3** (45.4), **282** (1.95), **15** (0.21), **325953** (0.21)
T19	2,254	976	**48** (44.8), **54** (14.7), **42** (12.8), **84** (5.33), **36** (5.12), **72** (2.66), **96** (2.15), **90** (1.95), **60** (1.84), **894** (1.74), **6** (1.74), **66** (1.64), **78** (0.92), **108** (0.72), **120** (0.61), **102** (0.41), **114** (0.31), **63** (0.2), **156** (0.1), **132** (0.1), **144** (0.1), **12** (0.1)

Continued on next page

Table 8.3 – continued from previous page

T.S.	%D.P.	Tot.	Periods and # of each period rel. to Tot. periods
T20	0,840	833	**252** (58.6), **206** (24.2), **848** (12.2), **226** (3.6), **728** (0.48), **18480** (0.48), **48606** (0.36)
T21	1,536	976	**38** (505), **17782** (19.7), **46** (10.4), **680** (6.15), **1624** (5.23), **54** (4.1), **42** (1.95), **58** (1.24), **62** (0.61), **50** (0.41), **1982** (0.31), **78654** (0.2), **66** (0.1), **78** (0.1), **96** (0.1)
T22	2,347	639	**28** (70.6), **3284** (9.23), **188** (5.79), **8556** (5.16), **22** (2.66), **40** (2.5), **72** (1.1), **84** (0.78), **56** (0.63), **48** (0.47), **36** (0.31), **60** (0.31), **44** (0.16), **68** (0.16), **124** (0.16)
T23	2,296	958	**1485** (21.2), **12009** (12.5), **48** (11.7), **60** (11.3), **30** (8.14), **3** (8.14), **66** (4.49), **72** (4.28), **54** (4.18), **90** (3.13), **42** (2.61), **24** (2.51), **78** (1.88), **36** (0.94), **84** (0.84), **96** (0.73), **108** (0.52), **102** (0.42), **126** (0.21), **120** (0.1), **132** (0.1), **114** (0.1)
T24	0,818	977	**80** (47.1), **68** (33.2), **38** (11.8), **4** (4.2), **8** (1.94), **14** (1.33), **48** (0.31), **72** (0.2)
T25	1,013	987	**3308** (34.2), **10452** (30.5), **118** (29.1), **358** (3.34), **14290** (0.51), **16** (0.51), **410** (0.41), **20290** (0.2), **120** (0.2), **664** (0.1)
T26	0,727	962	**504** (71.8), **1954** (23.5), **4474** (3.14), **244** (0.94), **32** (0.42), **2186** (0.21), **1468** (0.1)
T27	0,305	983	**2** (99.4), **878** (0.51), **576** (0.1)
T28	0,913	985	**10** (80.3), **80** (6.9), **52** (6.5), **62** (3.96), **224** (0.91), **197264** (0.71), **66** (0.3), **986** (0.3), **4116** (0.1)
T29	0,420	952	**2** (80.6), **68** (18.8), **798** (0.42), **366** (0.21)
T30	0,921	977	**212** (37.3), **2308** (28.4), **616** (27.6), **9370** (3.48), **480** (1.94), **33218** (0.92), **18** (0.41), **164** (0.2), **4216** (0.1)
T31	0,518	965	**6** (99.4), **1124** (0.21), **70** (0.21), **30** (0.1), **778** (0.1)
T32	1,884	849	**108** (65.5), **140** (7.3), **72** (5.18), **96** (4.83), **40** (3.65), **442** (3.42), **1252** (2.83), **124** (2.24), **92** (1.65), **48** (1.6), **282** (1.6), **49318** (0.71), **156** (0.24), **110** (0.12), **64** (0.12), **148** (0.12)

Continued on next page

Table 8.3 – continued from previous page

T.S.	%D.P.	Tot.	Periods and # of each period rel. to Tot. periods
T33	0,622	964	**262** (38.6), **72** (29.7), **2312** (14.2), **274** (11.5), **182** (3.84), **16** (2.18)
T34	0,657	912	**3** (52.7), **462321** (26.5), **22302** (17.3), **522** (3.18), **636** (0.11), **465** (0.11)
T35	1,362	954	**7** (53.6), **42** (17.5), **28** (10.7), **56** (6.18), **63** (3.46), **126** (2.73), **70** (1.99), **84** (1.47), **2002** (0.73), **784** (0.73), **2709** (0.42), **11760** (0.31), **112** (0.21)
T36	3,731	938	**32** (20.1), **28** (19.2), **40** (10.7), **3748** (9.28), **76** (7.46), **48** (6.86), **88** (3.91), **36** (3.73), **6** (3.3), **56** (1.92), **622** (1.76), **52** (1.6), **72** (1.6), **60** (1.49), **92** (1.39), **68** (1.17), **172** (1.17), **64** (0.85), **104** (0.85), **108** (0.64), **80** (0.53), **140** (0.43), **96** (0.43), **120** (0.32), **124** (0.32), **156** (0.21), **6656** (0.21), **224** (0.11), **192** (0.11), **84** (0.11), **112** (0.11), **152** (0.11), **236** (0.11), **128** (0.11), **44** (0.11)
T37	4,161	817	**92** (16.9), **2990** (16.5), **84** (14.2), **6** (9.18), **50** (7.1), **42** (5.1), **100** (4.28), **62** (4.28), **108** (3.55), **66** (2.94), **58** (2.2), **70** (1.96), **34** (1.96), **48** (1.35), **94** (1.22), **54** (0.98), **78** (0.73), **74** (0.73), **90** (0.61), **86** (0.49), **4406** (0.49), **46** (0.37), **102** (0.37), **226** (0.24), **106** (0.24), **134** (0.12), **1954** (0.12), **15560** (0.12), **156** (0.12), **110** (0.12), **98** (0.12), **132** (0.12), **1080** (0.12), **164** (0.12)
T38	0,717	975	**3** (70.1), **12471** (19.3), **915** (4.72), **2160** (2.87), **2208** (1.13), **71253** (0.92), **150** (0.1)
T39	0,610	983	**845** (81.1), **405** (8.14), **2495** (5.19), **90** (3.66), **40** (0.71), **65** (0.31)
T40	0,535	934	**18177** (55.1), **3** (41.2), **282** (2.25), **15** (0.43), **325953** (0.11)
T41	3,947	152	**26** (57.9), **8** (34.9), **198** (3.95), **2** (1.32), **32** (1.32), **24** (0.66)

Continued on next page

Table 8.3 – continued from previous page

T.S.	%D.P.	Tot.	Periods and # of each period rel. to Tot. periods
T42	2,481	927	**48** (46.1), **24** (12.5), **32** (7.12), **8** (6.26), **44** (5.93), **56** (5.72), **412** (3.34), **528** (2.91), **64** (2.27), **60** (1.4), **68** (1.4), **52** (1.19), **72** (0.76), **2454** (0.76), **1224** (0.54), **286** (0.43), **76** (0.32), **84** (0.22), **80** (0.22), **96** (0.22), **1634** (0.22), **36** (0.11), **40** (0.11)
T43	0,511	978	**32** (74.7), **72** (12.4), **188** (6.44), **4** (4.81), **28548** (1.64)
T44	1,434	976	**1808** (38.7), **48** (28.3), **60** (12.6), **72** (9.32), **322** (5.23), **84** (2.66), **36** (0.82), **6** (0.72), **96** (0.51), **488** (0.51), **132** (0.31), **108** (0.1), **408** (0.1), **916** (0.1)
T45	0,406	983	**6** (90.9), **142714** (7.83), **16** (1.27), **152** (0.2)
T46	7,748	955	**74** (6.18), **70** (5.24), **66** (4.83), **62** (4.82), **34** (4.61), **50** (4.5), **38** (4.29), **58** (3.87), **78** (3.66), **82** (3.25), **94** (3.14), **54** (2.94), **86** (2.83), **98** (2.72), **4** (2.51), **72** (2.2), **42** (2.2), **60** (1.88), **88** (1.68), **64** (1.68), **90** (1.68), **118** (1.57), **52** (1.57), **102** (1.47), **110** (1.36), **5382** (1.36), **236** (1.36), **106** (1.36), **68** (1.36), **76** (1.36), **46** (1.26), **122** (1.15), **114** (1.15), **160** (0.94), **96** (0.94), **84** (0.94), **80** (0.94), **40** (0.84), **56** (0.84), **48** (0.73), **112** (0.63), **36** (0.52), **104** (0.52), **130** (0.52), **134** (0.52), **128** (0.42), **138** (0.42), **180** (0.42), **126** (0.42), **1194** (0.42), **152** (0.42), **100** (0.31), **30** (0.31), **108** (0.31), **166** (0.21), **124** (0.21), **146** (0.21), **32** (0.21), **170** (0.21), **116** (0.21), **178** (0.21), **142** (0.21), **120** (0.21), **136** (0.21), **92** (0.21), **144** (0.1), **154** (0.1), **186** (0.1), **770** (0.1), **132** (0.1), **174** (0.1), **218** (0.1), **148** (0.1), **156** (0.1)
T47	3,981	427	**866** (32.8), **18** (18.5), **6** (13.1), **12** (7.49), **24** (7.49), **1836** (5.15), **30** (4.22), **42** (4.22), **36** (2.58), **48** (1.41), **60** (0.94), **54** (0.7), **8** (0.47), **66** (0.23), **72** (0.23), **22** (0.23), **90** (0.23)
T48	0,713	981	**2320** (77.6), **177802** (13.4), **183532** (5.3), **224** (2.34), **336** (1.29), **872** (0.31), **178** (0.1)
T49	1,057	946	**42** (29.3), **168** (26.5), **18** (26.1), **4200** (16.1), **114** (0.74), **228** (0.63), **24** (0.21), **66** (0.21), **210** (0.11), **144** (0.11)

Continued on next page

Table 8.3 – continued from previous page

T.S.	%D.P.	Tot.	Periods and # of each period rel. to Tot. periods
T50	0,651	921	**436** (49.5), **52** (29.4), **24** (11.1), **2044** (8.25), **48** (1.95), **1778** (0.65)
T51	0,863	926	**3802** (74.6), **2** (21.7), **60928** (1.94), **1350** (0.65), **2894** (0.43), **44314** (0.32), **248** (0.22), **10580** (0.11)
T52	0,818	977	**62** (43.3), **2** (38.9), **138** (15.9), **23792** (0.82), **270** (0.51), **62954** (0.31), **15008** (0.2), **17134** (0.1)

As is clear from this table there is a rich variety present in the different tag systems tested as far as periodicity is concerned. But before discussing this any further, it is important to exclude **T6** and **T12** from our discussion, given the low number of periods found in experiment 1 for these two tag systems. Looking at the results for the other tag systems, it is clear that there are significant differences in the variety of different periods relative to the total number of periods. The maximum variety of periods is found in **T46** with no less than 74 different periods, while **T13** and **T27** have the lowest scores. Between these two extremes, the variety of periods for each of the tag systems does not allow for a clear classification, since the different values vary between these two extremes in a relatively continuous way.

As was said before, there is a clear dominance of period 6 in **T1**, and the results from table 8.3 only affirm this, 84.2% of the periods found being a period 6. This dominance of one period can be is also seen in several other tag systems. In **T13**, **T17**, **T27**, **T31** and **T45** for example, there are periods with probabilities higher than 90%. There are also tag systems where the difference between the most dominant and the one following it is relatively small, so one could speak about more than one dominant period. This is for example the case for **T11**. There are also tag systems where the probability of the most dominant period lies significantly low. This is for example the case for **T46**, the tag system with the largest variety in the number of different periods found. This does not mean that there is some correlation between variety and probability of the dominant period. For example, although the variety of different periods in **T1**, equal to 1.772, lies significantly higher than in **T2**, equal to 0.448, the probability that period 6 will occur for **T1** lies significantly higher than that for the dominant period 168 in **T2**. This becomes even more apparent in comparing the results for **T1** with a tag system for which the relative variety is more or less the same, such as **T3**, where the probability of the dominant period 202, equal to 39.5%, is less than halve of the probability of period 6 in **T1**.

It should also be noted that there are tag systems that produce periods that are not even. For example, **T35** has produced strings with period 7. In fact all the different periods found for **T35** are divisible by 7. Another example is **T38**, where all the periods are divisible by 3. The fact that we have already observed that for **T1**, many of the periods are in fact additive compositions of several different even numbers, while there are clearly also other compositions possible for other tag systems, illustrates, on a very intuitive level, the number-theoretical character of tag systems. Indeed, we are convinced that interesting things might be done on this level if the periodic types in tag systems would be investigated in more detail. For instance, it would to our mind be interesting if one would be able to construct composed tag systems, from other tag systems with different periods, such that one could begin to compute with the periods themselves. But, as is the case for many of the results described in this dissertation, more research is needed here.

Clearly, the results from table 8.3 do not bring us very far theoretically. Although we have made some observations on the basis of the table, they can only serve to build up an intuition of the behaviour of periods in tag systems. They do not directly lead to appealing results. Another source of information however are the several periodic structures found for each of the tag systems. Here, the results – it must be said – were rather surprising. We already discussed two different types of periods for **T1**: the so-called "regular" and "irregular" ones. After a rather nerve-racking analysis of the several structures found for the remaining 51 tag systems, combining classic pencil-and-paper work with computer work I found two more types and will now give a detailed description of these types. I should warn the reader though that a serious formalization is still needed here in order to make the definitions more immediate. Furthermore, the definitions

should be considered as preliminary definitions. The definitions are meant as a description, a means for recognizing the different types. This implies that the definitions are most probably in part redundant, i.e., some of the characteristics may be derived from a smaller set of characteristics. Some more research time would be needed, to find more elegant and explanatory definitions.
Still, we would like to make some notational conventions. First of all, remember that a *periodic structure* is the sequence of relevant letters in a given periodic string S_x. From now on, the periodic structure of a given string S_x will be denoted as $\mathfrak{S}_x$ and its length as $|\mathfrak{S}_x|$. It should be noted that normally, no index will be used, so that $\mathfrak{S}_x$ will be shortened to $\mathfrak{S}$. The length of a period will be indicated as p, while given a periodic string S with period length p, S plus the next $p-1$ strings produced from S, i.e., a sequence of p strings with period p, will be denoted as $[P]$. If a given string is denoted as S_x then S_{x+i} denotes the string that is produced from S_x after i iterations. We would also like to introduce two operations $\overset{\circ}{\rightarrow}$ and $\overset{\circ}{\rightarrow}^{+}$. Given a tag system $\mathbf{T}$, with shift number v and a string $S = a_1 a_2 ... a_n$. Then, a new string:

$$S' = a_{n-(n-1 \bmod v)} b_1 ... b_i w_{a_1} w_{a_{v+1}} w_{a_{2v+1}} ... w_{a_{n-v-(n-1 \bmod v)}}$$

with $i < v-1$ is produced from S after all its relevant letters have been scanned except for the last. The result of applying $\overset{\circ}{\rightarrow}^{+}$ to S is the string:

$$w_{a_1} w_{a_{v+1}} w_{a_{2v+1}} ... w_{a_{n-(n-1 \bmod v)}}$$

i.e. the string resulting from S after all its relevant letters have been scanned, but without taking into account the shift induced by $l_S \bmod v$. A string S' resulting from a string S by applying $\overset{\circ}{\rightarrow}^{+}$ to S will be indicated as S^{+}. The result of applying $\overset{\circ}{\rightarrow}$ to S is the string S^{+} minus its first $\overline{l_S \bmod v}$ leading letters, i.e. the string resulting from S after all its relevant letters have been scanned, taking into account the shift induced by the length of S.[18]

[18] To explain this with an example, let us apply $\overset{\circ}{\rightarrow}^{+}$ rsp. $\overset{\circ}{\rightarrow}$ to **1**01**1**10**1**11**0**10**0**00**0** for **T1**, resulting in 1**1**01**1**10**1**11**0**10**0**00**0**0 rsp. **1**01**1**10**1**11**0**10**0**00**0**0.

Periodic strings of type 1 Periods of type 1 are those that behave in the same way as a period 6 string in Post's tag system. A string with period 6 in **T1** is for example S_1 = **1**01**1**10**1**11**0**10**0**00**0**0. Starting **T1** with S as initial condition, we get the following sequence of productions:

101**1**10**1**11**0**10**0**00**0**0
110**1**11**0**10**0**00**0**01**1**01
111**0**10**0**00**0**01**1**01**1**10**1**
010**0**00**0**01**1**01**1**10**1**11**0**1
000**0**01**1**01**1**10**1**11**0**10**0**
001**1**01**1**10**1**11**0**10**0**00
101**1**10**1**11**0**10**0**00**0**0

After 6 iterations, S reproduces itself. There are several features that characterize this kind of periodic strings. First of all, for at least one string $S \in [P]$, $|\mathfrak{S}| \equiv 0 \bmod p$. In the example, the first, second and fifth string have this property. These strings are such that they will always produce a string S^+ through $\overset{\circ}{\rightarrow}^{+}$ of length $l_{S^+} \equiv 0 \bmod v$. Furthermore, after erasing the first $\overline{l_S \bmod v}$ letters from this S^+, S results again. I.e. in applying $\overset{\circ}{\rightarrow}$ to S, with $|\mathfrak{S}| \equiv 0 \bmod p$, it will reproduce itself. The fact that S^+ is of length divisible by v, while $|\mathfrak{S}| \equiv 0 \bmod p$, is basic for this type. Indeed, this is why the string resulting from concatenating the string S^+ n times erasing the first $\overline{l_S \bmod v}$, will always result in that same string after its first p letters have been scanned.
Let us now define a periodic string of type 1.

Definition 8.4.1 *A sequence of periodic strings* $[P]$ *is said to be of type 1, iff. for each of these strings,* $|\mathfrak{S}| \geq p$ *and there is at least one string* S *for which* $|\mathfrak{S}| \equiv 0 \bmod p$*,* $l_{S^+} \equiv 0 \bmod v$ *and* $S \overset{\circ}{\rightarrow} S$*.*

Given this definition of type 1, it is possible to apply Shearer's proof to a given tag system if at least two strings A and B of this type but with different p are found. For the proof to be applicable, one more condition has to be fulfilled. Suppose A precedes B, then the shift induced by scanning all the relevant letters in A must be such that it is exactly that shift that allows B to self-reproduce, and vice versa. Now suppose A is of period p_A and B of period p_B. Then for any n and m, it is always possible to construct a string with period $nP_A + mP_B$.

Periodic strings of type 2 To explain the second type of periodic strings, we will use an example produced by **T3**, i.e. the string

$$S_1 = 111101000010000100001000111111111111010000100001000$$

Let me remember that the production rules for **T3** are: $1 \rightarrow 01000, 0 \rightarrow 111, v = 4$. Starting **T3** with S we get the following productions:

$\underbrace{\mathbf{1}111\mathbf{0}100\mathbf{0}010\mathbf{0}00}1\mathbf{0}000\mathbf{1}000\mathbf{1}111\mathbf{1}111\underbrace{\mathbf{1}111\mathbf{0}100\mathbf{0}010\mathbf{0}001\mathbf{0}00}$

0100**0**010**0**001**0**000**1**000**1**111**1**111**1**111**0**100**0**010**0**001**0**000**1**000

0010**0**001**0**000**1**000**1**111**1**111**1**111**0**100**0**010**0**001**0**000**1**000**1**11

0001**0**000**1**000**1**111**1**111**1**111**0**100**0**010**0**001**0**000**1**000**1**111**1**1

0000**1**000**1**111**1**111**1**111**0**100**0**010**0**001**0**000**1**000**1**111**1**111**1**

1000**1**111**1**111**1**111**0**100**0**010**0**001**0**000**1**000**1**111**1**111**1**111

1111**1**111**1**111**0**100**0**010**0**001**0**000**1**000**1**111**1**111**1**111**0**1000

1111**1**111**0**100**0**010**0**001**0**000**1**000**1**111**1**111**1**111**0**100**0**010**0**0

1111**0**100**0**010**0**001**0**000**1**000**1**111**1**111**1**111**0**100**0**010**0**001**0**00

As is clear from the productions S_1 reproduces itself after exactly 8 iterations. So in what way does this type of periodic string differ from those of type 1? The basic difference here is the fact that in a given $[P]$ of type 1 at least one of the strings $S \in [P]$ is such that $|\mathfrak{S}| \equiv 0 \bmod p$ and S reproduces itself after one application of the operation $\overset{\circ}{\rightarrow}$. This is not the case for periods of type 2. First of all, as is clear from the example, none of the periodic structures $\mathfrak{S}$ (put in bold) is such that $|\mathfrak{S}| = p$, while for any S it is always the case that $|\mathfrak{S}| > p$. Furthermore, there is no $S \in [P]$ such that $S \overset{\circ}{\rightarrow} S$. Instead, for any $S_x, S_y \in [P]$, $S_x \overset{\circ}{\rightarrow} S_y$ with $S_x \neq S_y$. Indeed, in case of the example, starting from S_1, applying $\overset{\circ}{\rightarrow}$ will first result in the 6th string from the sequence, then in the 3th, the 8th and the 5th. Finally, applying $\overset{\circ}{\rightarrow}$ one more time to the fifth string, results in S_1. We thus need 5 applications of the operation $\overset{\circ}{\rightarrow}$ instead of one for S_1 to be reproduced.

In general, for any periodic strings $S_x \in [P]$ of this type, the number of times n we need to apply $\overset{\circ}{\to}$ before S_x is reproduced is such that $1 < n \leq p$. It is also possible that S_x can never be reproduced through $\overset{\circ}{\to}$, but that another string $S_y \neq S_x$, $S_y \in [P]$ results from S_x after application of $\overset{\circ}{\to}$ for a certain number of times m, $0 < m \leq p$, such that S_y can be produced by applying $\overset{\circ}{\to}$ for a certain number of times n. It is also important to note that if S_x can be reproduced in applying $\overset{\circ}{\to}$ for a certain number of times n, at least one of the strings S_y produced will be such that $l_{S_y^+} \neq 0 \bmod v$.
Taking a closer look at the string S_1 from the example, it is clear that S_1 is periodic because the last $l_{S_1} - 24$ letters from S_1 are the same as the first $l_{S_1} - 24$. Indeed, after the first $8 \cdot 4$ letters from S_1 have been processed by **T3** the last $l_{S_1} - 24$ letters from S_1 plus the new letters tagged to S_1 are again S_1. This is not only valid for S_1, but for any string produced in the example.
Periodic strings of type 2 are defined as follows:

Definition 8.4.2 *A sequence of periodic strings* $[P]$ *is said to be of type 2, iff. for each* $S \in [P]$, $|\mathfrak{S}| > p$, $|\mathfrak{S}| \neq 0 \bmod p$. *For each string* $S \in [P]$ *applying* $\overset{\circ}{\to}$ *a certain number of times* n, $1 < n \leq p$ *starting from* S *will either result in* S*, or produce another string* $S' \in [P]$ *for which this is the case. If a given string* $S_x \in [P]$ *can be reproduced in this way, at least one of the intermediary strings* $S_y \in [P]$ *produced, will be such that* $l_{S_y^+} \neq 0 \bmod v$.

Given a string S_x of type 2 that can be reproduced after x applications of the operation $\overset{\circ}{\to}$. Since at least one of the strings S_y produced during this process will be such that $l_{S_y^+} \neq 0 \bmod v$, this type of periodic strings cannot be cannot be used to find a variant of Shearer's proof even if we would combine it with strings of type 1.

Periodic strings of type 3 To discuss periodic strings of type 3 we will give two examples. Let us first consider the following string:

$$S_1 = 00011101000111$$

produced by **T3**. Starting **T3** with $S_{T3,1}$ we get the following sequence of productions:

0001**1**101**0**001**1**1

1101**0**001**1**111**1**

0001**1**111**1**010**0**0

1111**1**010**0**011**1**

1010**0**011**1**010**0**0

0011**1**010**0**001**0**00

1010**0**001**0**001**1**1

0001**0**001**1**101**0**00

0001**1**101**0**001**1**1

A second example is produced by **T14**. When started with:

$$S_{T14,1} = 101001010101010101101001101001010110100110100101\\ 0101011010010101010110100101010101101001101001010$$

T14, with production rules 0 → 1010, 1 → 110100, $v = 5$ will produce the following sequence of strings:[19]

10100**1**0101**0**1010**1**0110**1**0011**0**1001**0**1011**0**1001**1**0100**1**0101**0**1011**0**1001**0**1010**1**0110**1**0010**1**0101**0**1101**0**0110**1**0010**1**0

10101**0**1010**1**0110**1**0011**0**1001**0**1011**0**1001**1**0100**1**0101**0**1011**0**1001**0**1010**1**0110**1**0010**1**0101**0**1101**0**0110**1**0010**1**0110**1**00

01010**1**0110**1**0011**0**1001**0**1011**0**1001**1**0100**1**0101**0**1011**0**1001**0**1010**1**0110**1**0010**1**0101**0**1101**0**0110**1**0010**1**0110**1**0011**0**100

10110**1**0011**0**1001**0**1011**0**1001**1**0100**1**0101**0**1011**0**1001**0**1010**1**0110**1**0010**1**0101**0**1101**0**0110**1**0010**1**0110**1**0011**0**1001**0**10

10011**0**1001**0**1011**0**1001**1**0100**1**0101**0**1011**0**1001**0**1010**1**0110**1**0010**1**0101**0**1101**0**0110**1**0010**1**0110**1**0011**0**1001**0**1011**0**100

01001**0**1011**0**1001**1**0100**1**0101**0**1011**0**1001**0**1010**1**0110**1**0010**1**0101**0**1101**0**0110**1**0010**1**0110**1**0011**0**1001**0**1011**0**1001**1**0100

01011**0**1001**1**0100**1**0101**0**1011**0**1001**0**1010**1**0110**1**0010**1**0101**0**1101**0**0110**1**0010**1**0110**1**0011**0**1001**0**1011**0**1001**1**0100**1**010

01001**1**0100**1**0101**0**1011**0**1001**0**1010**1**0110**1**0010**1**0101**0**1101**0**0110**1**0010**1**0110**1**0011**0**1001**0**1011**0**1001**1**0100**1**0101**0**10

10100**1**0101**0**1011**0**1001**0**1010**1**0110**1**0010**1**0101**0**1101**0**0110**1**0010**1**0110**1**0011**0**1001**0**1011**0**1001**1**0100**1**0101**0**1010**1**0

10101**0**1011**0**1001**0**1010**1**0110**1**0010**1**0101**0**1101**0**0110**1**0010**1**0110**1**0011**0**1001**0**1011**0**1001**1**0100**1**0101**0**1010**1**0110**1**00

01011**0**1001**0**1010**1**0110**1**0010**1**0101**0**1101**0**0110**1**0010**1**0110**1**0011**0**1001**0**1011**0**1001**1**0100**1**0101**0**1010**1**0110**1**0011**0**100

01001**0**1010**1**0110**1**0010**1**0101**0**1101**0**0110**1**0010**1**0110**1**0011**0**1001**0**1011**0**1001**1**0100**1**0101**0**1010**1**0110**1**0011**0**1001**0**10

01010**1**0110**1**0010**1**0101**0**1101**0**0110**1**0010**1**0110**1**0011**0**1001**0**1011**0**1001**1**0100**1**0101**0**1010**1**0110**1**0011**0**1001**0**1010**1**0

10110**1**0010**1**0101**0**1101**0**0110**1**0010**1**0110**1**0011**0**1001**0**1011**0**1001**1**0100**1**0101**0**1010**1**0110**1**0011**0**1001**0**1010**1**0101**0**

[19] I apologize for the small font used here, but else it would not be possible to fit the strings on the page.

10010**1**0101**0**1101**0**0110**1**0010**1**0110**1**0011**0**1001**0**1011**0**1001**1**0100**1**0101**0**1010**1**0110**1**0011**0**1001**0**1010**1**0101**0**1101**0**0

10101**0**1101**0**0110**1**0010**1**0110**1**0011**0**1001**0**1011**0**1001**1**0100**1**0101**0**1010**1**0110**1**0011**0**1001**0**1010**1**0101**0**1101**0**0110**1**00

01101**0**0110**1**0010**1**0110**1**0011**0**1001**0**1011**0**1001**1**0100**1**0101**0**1010**1**0110**1**0011**0**1001**0**1010**1**0101**0**1101**0**0110**1**0011**0**100

00110**1**0010**1**0110**1**0011**0**1001**0**1011**0**1001**1**0100**1**0101**0**1010**1**0110**1**0011**0**1001**0**1010**1**0101**0**1101**0**0110**1**0011**0**1001**0**10

10010**1**0110**1**0011**0**1001**0**1011**0**1001**1**0100**1**0101**0**1010**1**0110**1**0011**0**1001**0**1010**1**0101**0**1101**0**0110**1**0011**0**1001**0**1010**1**0

10110**1**0011**0**1001**0**1011**0**1001**1**0100**1**0101**0**1010**1**0110**1**0011**0**1001**0**1010**1**0101**0**1101**0**0110**1**0011**0**1001**0**1010**1**0110**1**00

10011**0**1001**0**1011**0**1001**1**0100**1**0101**0**1010**1**0110**1**0011**0**1001**0**1010**1**0101**0**1101**0**0110**1**0011**0**1001**0**1010**1**0110**1**0011**0**100

01001**0**1011**0**1001**1**0100**1**0101**0**1010**1**0110**1**0011**0**1001**0**1010**1**0101**0**1101**0**0110**1**0011**0**1001**0**1010**1**0110**1**0011**0**1001**1**0100

01011**0**1001**1**0100**1**0101**0**1010**1**0110**1**0011**0**1001**0**1010**1**0101**0**1101**0**0110**1**0011**0**1001**0**1010**1**0110**1**0011**0**1001**1**0100**1**010

01001**1**0100**1**0101**0**1010**1**0110**1**0011**0**1001**0**1010**1**0101**0**1101**0**0110**1**0011**0**1001**0**1010**1**0110**1**0011**0**1001**1**0100**1**0101**0**10

10100**1**0101**0**1010**1**0110**1**0011**0**1001**0**1010**1**0101**0**1101**0**0110**1**0011**0**1001**0**1010**1**0110**1**0011**0**1001**1**0100**1**0101**0**1010**1**0

10101**0**1010**1**0110**1**0011**0**1001**0**1010**1**0101**0**1101**0**0110**1**0011**0**1001**0**1010**1**0110**1**0011**0**1001**1**0100**1**0101**0**1010**1**0110**1**00

01010**1**0110**1**0011**0**1001**0**1010**1**0101**0**1101**0**0110**1**0011**0**1001**0**1010**1**0110**1**0011**0**1001**1**0100**1**0101**0**1010**1**0110**1**0011**0**100

10110**1**0011**0**1001**0**1010**1**0101**0**1101**0**0110**1**0011**0**1001**0**1010**1**0110**1**0011**0**1001**1**0100**1**0101**0**1010**1**0110**1**0011**0**1001**0**10

10011**0**1001**0**1010**1**0101**0**1101**0**0110**1**0011**0**1001**0**1010**1**0110**1**0011**0**1001**1**0100**1**0101**0**1010**1**0110**1**0011**0**1001**0**1011**0**100

01001**0**1010**1**0101**0**1101**0**0110**1**0011**0**1001**0**1010**1**0110**1**0011**0**1001**1**0100**1**0101**0**1010**1**0110**1**0011**0**1001**0**1011**0**1001**1**0100

01010**1**0101**0**1101**0**0110**1**0011**0**1001**0**1010**1**0110**1**0011**0**1001**1**0100**1**0101**0**1010**1**0110**1**0011**0**1001**0**1011**0**1001**1**0100**1**010

10101**0**1101**0**0110**1**0011**0**1001**0**1010**1**0110**1**0011**0**1001**1**0100**1**0101**0**1010**1**0110**1**0011**0**1001**0**1011**0**1001**1**0100**1**0101**0**10

01101**0**0110**1**0011**0**1001**0**1010**1**0110**1**0011**0**1001**1**0100**1**0101**0**1010**1**0110**1**0011**0**1001**0**1011**0**1001**1**0100**1**0101**0**1011**0**100

00110**1**0011**0**1001**0**1010**1**0110**1**0011**0**1001**1**0100**1**0101**0**1010**1**0110**1**0011**0**1001**0**1011**0**1001**1**0100**1**0101**0**1011**0**1001**0**10

10011**0**1001**0**1010**1**0110**1**0011**0**1001**1**0100**1**0101**0**1010**1**0110**1**0011**0**1001**0**1011**0**1001**1**0100**1**0101**0**1011**0**1001**0**1010**1**0

01001**0**1010**1**0110**1**0011**0**1001**1**0100**1**0101**0**1010**1**0110**1**0011**0**1001**0**1011**0**1001**1**0100**1**0101**0**1011**0**1001**0**1010**1**0110**1**00

10110**1**0011**0**1001**1**0100**1**0101**0**1010**1**0110**1**0011**0**1001**0**1011**0**1001**1**0100**1**0101**0**1011**0**1001**0**1010**1**0110**1**0010**1**0101**0**

10011**0**1001**1**0100**1**0101**0**1010**1**0110**1**0011**0**1001**0**1011**0**1001**1**0100**1**0101**0**1011**0**1001**0**1010**1**0110**1**0010**1**0101**0**1101**0**0

01001**1**0100**1**0101**0**1010**1**0110**1**0011**0**1001**0**1011**0**1001**1**0100**1**0101**0**1011**0**1001**0**1010**1**0110**1**0010**1**0101**0**1101**0**0110**1**00

10100**1**0101**0**1010**1**0110**1**0011**0**1001**0**1011**0**1001**1**0100**1**0101**0**1011**0**1001**0**1010**1**0110**1**0010**1**0101**0**1101**0**0110**1**0010**1**0

Let us now look at the features shared by the periodic strings produced from $S_{T3,1}$ (reproduces itself after 8 steps) as well as from $S_{T14,1}$ (reproduces itself after 40 steps). First of all, it is basic to both sequences of periodic strings, that for any string $S_x \in [P]$, with periodic structure $\mathfrak{S}_x$ that $|\mathfrak{S}_x| < p$. Furthermore, there are at least two strings $S \in [P]$ such that $p \equiv 0 \bmod |\mathfrak{S}|$.[20] Since for any S_x,

[20] In fact, for almost all the sequences of periodic strings $[P]$ of this type we have investigated,

$|\mathfrak{S}_\mathfrak{x}| < p$, applying $\overset{\circ}{\rightarrow}$ to S_x will not lead to self-reproduction. For those strings $S_x \in [P]$ for which $p \equiv 0 \bmod |\mathfrak{S}_x|$ applying $\overset{\circ}{\rightarrow}$ to S_x will result in another string $S_y \in [P]$, with $S_x \neq S_y$. Clearly, applying $\overset{\circ}{\rightarrow}$ to this string $S_x \in [P]$ in fact leads to the string $S_{x+|\mathfrak{S}_x|}$. Furthermore, for each such string S_x, $l_{S_x^+} \equiv 0 \bmod v$. If S_x is a string such that $p \equiv 0 \bmod |\mathfrak{S}_x|$, the string $S_{x+|\mathfrak{S}_x|}$ will share the same properties as S_x, with $p \equiv 0 \bmod |\mathfrak{S}_{x+|\mathfrak{S}_x|}|$, $l^+_{S_{x+|\mathfrak{S}_x|}} \equiv 0 \bmod v$. For the two examples given, it is the case for any string $S_x \in [P]$ that $p \equiv 0 \bmod |\mathfrak{S}_\mathfrak{x}|$ and $l_{S_x^+} \equiv 0 \bmod v$. Note also that for any string in the examples considered, it will be reproduced after exactly two applications of $\overset{\circ}{\rightarrow}$. Of course, there are also examples of strings S_x for other tag systems that need more than two applications of $\overset{\circ}{\rightarrow}$ before S_x is produced again, but we did not provide examples, because this would lead to even longer productions. Note also that the number of times n $\overset{\circ}{\rightarrow}$ has to be applied before reproduction, is equal to $p/|\mathfrak{S}|$.
We have provided two different examples here, because there is one feature of the first example not shared by the second, this feature being that S_1 and S_5 are mirror images of each other, i.e. the structures 0101 and 1010. We wanted to mention this here since the feature is shared by almost all the periodic strings of this type we have studied. It is however by no means a necessary condition for a sequence of periodic strings to be of type 3.

Let us now turn to a more formal definition of sequences of periodic strings of type 3.

Definition 8.4.3 *A sequence of periodic strings* $[P]$ *is said to be of type 3, iff. for each* $S \in [P]$, $|\mathfrak{S}| < p$ *and there is at least one string* S_x*, for which* $p \equiv 0 \bmod |\mathfrak{S}_x|$, $l_{S_x^+} \equiv 0 \bmod v$. *Furthermore, for any string* S_y *that can be produced from* S_x *through repeated application of* $\overset{\circ}{\rightarrow}$, $y = x + i \cdot |\mathfrak{S}_x|$ *the same properties hold. Clearly, if* $i = p/|\mathfrak{S}_x|$*, then* $S_x = S_y$*, i.e.,* $p/|\mathfrak{S}_x|$ *applications of* $\overset{\circ}{\rightarrow}$ *starting with* S_x *produces* S_x*.*

Contrary to periodic strings of type 2, these strings do allow for the kind of construction of periodic strings necessary to find a variant of Shearer's proof for a given tag system. Given a tag system **T** for which at least two periodic strings S_x

this is the case for every $S \in [P]$ (See for instance the examples).

and S_y of type 3 are found with different periods p_{S_x} and p_{S_y} for which the two extra properties mentioned in the definition hold.[21] Then it is always possible, given m and n, to construct a periodic string with period $nP_A + mP_B$. As was the case for periodic strings of type 1 an extra condition has to be fulfilled, viz., that the shifts induced by each of the periodic strings used have to be properly synchronized.

Periodic strings of type 4 An example of a periodic string of type 4, is the string:

$$S_1 = 010000000000110111011101001101110111010000$$

produced by **T1**, and leads to the following productions:

010**0**00**0**00**0**00**1**10**1**11**0**11**1**01**0**01**1**01**1**10**1**11**0**10**0**00
000**0**00**0**00**1**10**1**11**0**11**1**01**0**01**1**01**1**10**1**11**0**10**0**00**0**0
000**0**00**1**10**1**11**0**11**1**01**0**01**1**01**1**10**1**11**0**10**0**00**0**00**0**
000**1**10**1**11**0**11**1**01**0**01**1**01**1**10**1**11**0**10**0**00**0**00**0**00
110**1**11**0**11**1**01**0**01**1**01**1**10**1**11**0**10**0**00**0**00**0**00**0**0
111**0**11**1**01**0**01**1**01**1**10**1**11**0**10**0**00**0**00**0**00**0**01**1**01
011**1**01**0**01**1**01**1**10**1**11**0**10**0**00**0**00**0**00**0**01**1**01**1**10**1**
101**0**01**1**01**1**10**1**11**0**10**0**00**0**00**0**00**0**01**1**01**1**10**1**00
001**1**01**1**10**1**11**0**10**0**00**0**00**0**00**0**01**1**01**1**10**1**00**1**10**1**
101**1**10**1**11**0**10**0**00**0**00**0**00**0**01**1**01**1**10**1**00**1**10**1**00
110**1**11**0**10**0**00**0**00**0**00**0**01**1**01**1**10**1**00**1**10**1**00**1**10**1**
111**0**10**0**00**0**00**0**00**0**01**1**01**1**10**1**00**1**10**1**00**1**10**1**11**0**1
010**0**00**0**00**0**00**0**01**1**01**1**10**1**00**1**10**1**00**1**10**1**11**0**11**1**01
000**0**00**0**00**0**01**1**01**1**10**1**00**1**10**1**00**1**10**1**11**0**11**1**01**0**0
000**0**00**0**01**1**01**1**10**1**00**1**10**1**00**1**10**1**11**0**11**1**01**0**00**0**
000**0**01**1**01**1**10**1**00**1**10**1**00**1**10**1**11**0**11**1**01**0**00**0**00
001**1**01**1**10**1**00**1**10**1**00**1**10**1**11**0**11**1**01**0**00**0**00**0**0
101**1**10**1**00**1**10**1**00**1**10**1**11**0**11**1**01**0**00**0**00**0**00**0**
110**1**00**1**10**1**00**1**10**1**11**0**11**1**01**0**00**0**00**0**00**0**11**0**1

[21] I.e. $p \equiv 0 \bmod |\mathfrak{S}_x|$, $l_{S_x^+} \equiv 0 \bmod v$ and $p \equiv 0 \bmod |\mathfrak{S}_y|$, $l_{S_y^+} \equiv 0 \bmod v$

100**1**10**1**00**1**10**1**11**0**11**1**01**0**00**0**00**0**00**0**11**0**11**1**01
110**1**00**1**10**1**11**0**11**1**01**0**00**0**00**0**00**0**11**0**11**1**01**1**10**1**
100**1**10**1**11**0**11**1**01**0**00**0**00**0**00**0**11**0**11**1**01**1**10**1**11**0**1
110**1**11**0**11**1**01**0**00**0**00**0**00**0**11**0**11**1**01**1**10**1**11**0**11**1**01
111**0**11**1**01**0**00**0**00**0**00**0**11**0**11**1**01**1**10**1**11**0**11**1**01**1**10**1**
011**1**01**0**00**0**00**0**00**0**11**0**11**1**01**1**10**1**11**0**11**1**01**1**10**1**11**0**1
101**0**00**0**00**0**00**0**11**0**11**1**01**1**10**1**11**0**11**1**01**1**10**1**11**0**10**0**
000**0**00**0**00**0**11**0**11**1**01**1**10**1**11**0**11**1**01**1**10**1**11**0**10**0**11**0**1
000**0**00**0**11**0**11**1**01**1**10**1**11**0**11**1**01**1**10**1**11**0**10**0**11**0**10**0**
000**0**11**0**11**1**01**1**10**1**11**0**11**1**01**1**10**1**11**0**10**0**11**0**10**0**00
011**0**11**1**01**1**10**1**11**0**11**1**01**1**10**1**11**0**10**0**11**0**10**0**00**0**0
011**1**01**1**10**1**11**0**11**1**01**1**10**1**11**0**10**0**11**0**10**0**00**0**00**0**
101**1**10**1**11**0**11**1**01**1**10**1**11**0**10**0**11**0**10**0**00**0**00**0**00
110**1**11**0**11**1**01**1**10**1**11**0**10**0**11**0**10**0**00**0**00**0**00**1**10**1**
111**0**11**1**01**1**10**1**11**0**10**0**11**0**10**0**00**0**00**0**00**1**10**1**11**0**1
011**1**01**1**10**1**11**0**10**0**11**0**10**0**00**0**00**0**00**1**10**1**11**0**11**1**01
101**1**10**1**11**0**10**0**11**0**10**0**00**0**00**0**00**1**10**1**11**0**11**1**01**0**0
110**1**11**0**10**0**11**0**10**0**00**0**00**0**00**1**10**1**11**0**11**1**01**0**01**1**01
111**0**10**0**11**0**10**0**00**0**00**0**00**1**10**1**11**0**11**1**01**0**01**1**01**1**10**1**
010**0**11**0**10**0**00**0**00**0**00**1**10**1**11**0**11**1**01**0**01**1**01**1**10**1**11**0**1
011**0**10**0**00**0**00**0**00**1**10**1**11**0**11**1**01**0**01**1**01**1**10**1**11**0**10**0**
010**0**00**0**00**0**00**1**10**1**11**0**11**1**01**0**01**1**01**1**10**1**11**0**10**0**00

As is clear, S_1 will reproduce itself after 40 iterations. The best way to explain periodic string of this type is by comparing them with periodic strings of type 3. As is the case for type 3, for each of the strings $S \in [P]$ of this type the periodic structure $\mathfrak{S}$ is such that $|\mathfrak{S}| < p$. The basic difference between these two types is that no string $S_x \in [P]$ of type 4 will be such that S_x will lead to S_y, $y = x + i \cdot |\mathfrak{S}_{\mathfrak{r}}|$, $S_x = S_y$, after exactly $p/|\mathfrak{S}_{\mathfrak{r}}|$ applications of $\overset{\circ}{\rightarrow}$ starting from S_x. For example given S_1 from the example, applying $\overset{\circ}{\rightarrow}$ to S will first result in the 14th then in the 28th, the 2th, the 16th, 29hth, the 3th and then after one more application of $\overset{\circ}{\rightarrow}$ to the 3th string, the 16th is produced again. It should also be noted that in applying $\overset{\circ}{\rightarrow}^{+}$ to each of these strings, none of them has a length $l \neq 0$ mod

v. As far as the example is concerned, it is clear that once the 16th string is produced, the system gets into a cycle by repeated application of $\overset{\circ}{\rightarrow}$ starting with the 16th string. Taking into account the length n of the periodic structure of the 16th string, it is clear that the number of times $\overset{\circ}{\rightarrow}$ has to be applied is equal to $\lfloor p/n \rfloor$. It will never be the case however for this type of periodic strings that if one of the strings S has a periodic structure $\mathfrak{S}$ for which $p \equiv 0 \bmod |\mathfrak{S}|$, S can be reproduced from S after exactly $p/|\mathfrak{S}|$ applications of $\overset{\circ}{\rightarrow}$, starting from S. Given a string S_x that can be reproduced through $\overset{\circ}{\rightarrow}$, at least one string S_y of the strings produced in this sequence going from S_x to S_x, will be such that $l_{S_y^+} \neq 0 \bmod v$. In general, for periodic strings of this type, a given string $S \in [P]$ of type 4, will either be reproduced after x applications of the operation $\overset{\circ}{\rightarrow}$ with $\lfloor p/|\mathfrak{S}| \rfloor \leq x \leq p$, $x \neq p/|\mathfrak{S}|$ or else, S will lead to a string with this property. It should be stated explicitly here that although in this example $x = \lfloor p/n \rfloor$ we have found examples for which $x > \lfloor p/n \rfloor$. Note also that this type has a clear similarity with strings of type 2.
We can now define a sequence of periodic strings of type 4 as follows:

Definition 8.4.4 *A sequence of periodic strings* $[P]$ *is said to be of type 4, iff. for each* $S \in [P]$, $|\mathfrak{S}| < p$. *For each string* S *of such sequence of strings, applying* $\overset{\circ}{\rightarrow}$ *a certain number of times* x, *starting from* S *will either result in* S, *or produce another string that can be reproduced after repeated application of* $\overset{\circ}{\rightarrow}$, *where it is always the case that* $\lfloor p/|\mathfrak{S}| \rfloor \leq x \leq p$, $x \neq p/|\mathfrak{S}|$. *Furthermore, for those strings* S_x *that can be reproduced through* $\overset{\circ}{\rightarrow}$, *at least one of the strings* S_y *produced through* $\overset{\circ}{\rightarrow}$ *will always be such that* $l_{S_y^+} \neq 0 \bmod v$.

Note that, given the last property mentioned in the definition, periodic strings of type 4 cannot be used to find a variant of Shearer's proof for a given tag system.

Although we did not expect this before we started with this limited investigation on periods in tag systems, we have found two more types of periodic string in tag systems. As is clear types 1 and 2 can be differentiated from types 3 and 4 through the relation between the period p and the lengths of the periodic structures involved. Indeed, for strings of type 1 and 2 it is always the case that for any string $S \in [P]$ of this type, $|\mathfrak{S}| \geq p$, while for strings of type 3 and 4, $|\mathfrak{S}| < p$. The following table summarizes some of the basic characteristics of the four main types.

	Type 1	Type 2	Type 3	Type 4
$\forall S \in [P] : \min(p, \|\mathfrak{S}\|)$	p	p	$\|\mathfrak{S}\|$	$\|\mathfrak{S}\|$
$\exists S \in [P] : n \overset{?}{=} \frac{p}{\min(p,\|\mathfrak{S}\|)}$	✓	╲	✓	╲

In the table $\min(p, |\mathfrak{S}|)$ returns the minimum of one of the two arguments. Note, that for type 1, it is also possible for some cases that $p = |\mathfrak{S}|$. The value n denotes the minimum number of times $\overset{\circ}{\rightarrow}$ has to be applied to a given string S for S to be reproduced. A ✓ indicates that there is an S fulfilling the condition, a ╲ indicates that there is none.

As for the applicability of Shearer's proof, if at least two periodic strings A and B are found for a given tag system, with different periods from either of the more "regular" types 1 or 3, it is always possible to apply it. In this way, it is very straightforward to construct an infinite number of different periods for tag systems that allow this kind of strings. This is clearly in contrast with periodic strings of type 2 and 4, which are more "irregular", and for which Shearer's proof cannot be applied in any direct way, given the fact that for any of the strings of this type used, at least one string S^+ will be produced such that $l_{S^+} \neq 0 \bmod v$. Of course, maybe a more complicated construction could be found, using more sophisticated "synchronization" techniques, for combining these "irregular" types in order to construct an infinite number of different periods for a given tag system.

The discussion of the several types was based on a combination of pencil-and-paper work and the use of the computer. This study resulted in a table, giving an overview of the different types found for each of the 52 tag systems. Although we cannot with complete certainty exclude the possibility that errors occurred in the preparation of this table, it gives an approximate idea about how the different types are distributed over the several tag systems. Table 8.5 gives this overview. A column headed with Tp*x* indicates the type. The symbol ✓ means that the specific type was found for a tag system, a ∕ means that it was not found. The table also included two extra columns headed Sh1, Sh2 and Sh3. These give an overview of the applicability of Shearer's proof for each of the tag systems. If more than two periodic strings of type 1 rsp. 2 were found with different periods for a given tag system, the column headed Sh1 rsp. Sh2 is marked with ✓, else it is marked with ∕. If both type 1 and 3 occur, the column headed Sh3 is marked with ✓, indicating that a combination of the two types can be used.

Table 8.5: Overview of the different types of periods.

T.S.	**Tp1**	**Tp2**	**Tp3**	**Tp4**	**Sh1**	**Sh2**	**Sh3**
T1	✓	∕	∕	∕	✓	∕	∕
T2	✓	∕	∕	✓	∕	∕	∕
T3	∕	✓	✓	✓	∕	✓	∕
T4	∕	✓	✓	✓	∕	✓	∕
T5	✓	✓	∕	✓	✓	∕	∕
T6	✓	∕	∕	✓	∕	∕	∕
T7	✓	∕	∕	✓	∕	∕	∕
T8	✓	✓	✓	✓	✓	∕	✓
T9	∕	∕	✓	✓	∕	✓	∕
T10	∕	∕	∕	✓	∕	∕	∕
T11	✓	✓	∕	✓	✓	∕	∕
T12	∕	∕	∕	✓	∕	∕	∕
T13	✓	✓	∕	✓	∕	∕	∕
T14	✓	∕	✓	✓	∕	∕	✓
T15	∕	∕	✓	✓	∕	✓	∕
T16	∕	✓	∕	✓	∕	∕	∕
T17	✓	✓	✓	✓	∕	∕	✓
T18	∕	✓	∕	✓	∕	∕	∕

Continued on next page

Table 8.5 – continued from previous page

T.S.	**Tp1**	**Tp2**	**Tp3**	**Tp4**	**Sh1**	**Sh2**	**Sh3**
T19	/	✓	✓	✓	/	/	/
T20	/	/	/	✓	/	/	/
T21	/	/	✓	✓	/	✓	/
T22	/	/	✓	✓	/	✓	/
T23	/	✓	✓	✓	/	✓	/
T24	/	✓	✓	✓	/	/	/
T25	/	✓	/	✓	/	/	/
T26	/	/	/	✓	/	/	/
T27	✓	✓	/	✓	/	/	/
T28	/	✓	/	✓	/	/	/
T29	✓	/	/	✓	/	/	/
T30	/	/	/	✓	/	/	/
T31	/	✓	/	✓	/	/	/
T32	/	/	✓	✓	/	/	/
T33	/	✓	/	✓	/	/	/
T34	/	✓	/	✓	/	/	/
T35	/	✓	✓	✓	/	/	/
T36	/	✓	✓	✓	/	/	/
T37	/	✓	✓	✓	/	✓	/
T38	/	✓	/	✓	/	/	/
T39	/	/	/	✓	/	/	/
T40	/	✓	/	✓	/	/	/
T41	✓	✓	✓	✓	/	/	✓
T42	/	✓	✓	✓	/	✓	/
T43	✓	✓	/	✓	/	/	/
T44	/	✓	✓	✓	/	/	/
T45	/	/	✓	✓	/	/	/
T46	/	✓	✓	✓	/	✓	/
T47	✓	/	/	✓	✓	/	/
T48	/	/	✓	✓	/	/	/

Continued on next page

Table 8.5 – continued from previous page

T.S.	Tp1	Tp2	Tp3	Tp4	Sh1	Sh2	Sh3
T49	/	✓	/	✓	/	/	/
T50	/	/	✓	✓	/	/	/
T51	✓	✓	/	✓	/	/	/
T52	✓	/	/	✓	/	/	/

As is clear from the table, there are several tag systems for which it is impossible to find a variant of Shearer's proof. Of course, this result is based on a sample of the different periodic strings found for a given tag system from experiment 1, and one should be careful in drawing any definite conclusion here. However, as we showed for **T2** there are clearly tag systems for which only one kind of periodic string of type 1 can be produced, and it seems probable that a similar proof might be found for some of the other tag systems. A similar proof might be found for the impossibility of periodic strings of type 3. As far as our experience goes with these tag systems, we are convinced that there do exist tag systems for which a variant of Shearer's proof cannot be found.

It is also interesting to point out that periodic strings of type 4 occur for all tag systems. As for **T1**, we already now that the period 40 found during another preliminary experiment is of type 4. The same goes for the period 66. Despite the existence of strings of type 4, we did not put a ✓ in the column for **T1** since no period of this type was found for the sample from experiment 1, thus indicating the problematic character of drawing conclusions on the basis of heuristic evidence.

Another observation that can be made on the basis of table 8.5 is the fact that only 3 tag systems are able to produce periodic strings of all four types. Furthermore, in most of the cases, tag systems able to produce strings of type 1 will not produce strings of type 3 and vice versa.

To conclude this section on periods in tag systems it is important to point out that much more research is needed here, research that must involve a combination of both pencil-and-paper work, help from the computer, the use of experimental data and the development of more general methods to prove e.g. the non-existence of certain periodic types for certain tag systems. In any case, we think that, although we have not fully explored periods in tag systems, further research on such periods might play an important role in any further study of tag systems. Not only can such research contribute to a better understanding of these systems but it might allow to connect tag systems to other branches of mathematics where periods play an important role.

It should also be pointed out here that types 1 (and 3) will play an important role in the next chapter, when we will consider the possibility of constructing a universal tag system with two symbols. Although for now, we have failed to construct one, these periods seem to offer an interesting possibility in this context. In this sense, the fact that there are tag systems for which periods of type 1 and 3 seem impossible, could be an important feature for identifying subclasses in the class of tag systems defined through the constraints. Of course, to really classify a given tag system according to these criteria, one would need an automated method to prove that it cannot produce periods of type 1 and/or 3.

The fact that there is such a rich variety in the behaviour of tag systems with $\mu = 2$, $v > 2$ furthermore illustrates how complicated this class actually is. As far as our experience goes with the solvable class of tag systems with $\mu = 2$, $v = 2$, it is important to point out here that this kind of complexities do not occur in this class.

8.5 Experiments 3–6: Summary of the results.

As was said, we will not give a detailed description of experiments 3–6 since these might deter the reader, given the disproportion between the length of description and analysis of the results and the actual conclusions that can be drawn from these experiments. Instead, we will merely summarize the main conclusions from each of the experiments. For more details, the reader is referred to Appendix C.

8.5.1 Experiment 3

The purpose of this experiment was to measure Lyapunov exponents for each of the 52 tag systems for 10 of the initial conditions classified as Immortals? in experiment 1. A positive exponent is considered as one of the signs of chaos and measures sensitive dependence on initial conditions. It can thus serve as an indication of the intractability of the system tested. The Lyapunov exponent was measured with respect to changes in the length and changes of one bit in the initial condition. Although there are some problems involved with measuring the Lyapunov exponent in the way we did this for tag systems – it is impossible to make the error in the initial condition arbitrary small or big – the results indicate a positive Lyapunov exponent for each of the tag systems. Although one system could be said to be more "sensitive" than the other, it is clear from the results that all tag systems show sensitive dependence on initial conditions. To our mind, this has been the least informative experiment we have done, since it is almost trivial to understand that the tag systems we have considered show sensitive dependence.

8.5.2 Experiment 4

In this experiment we measured the statistical distribution of the 0's and 1's in the strings produced by the tag systems when started with initial conditions classified as Immortals? in experiment 1. The question underlying this experiment was whether the proportion of the number of 0's and 1's imposed in the words from the production rules for each of the tag systems through constraint 3, would be statistically represented in the productions of the tag systems. If the answer to this question would be positive, one could conclude on a heuristic basis that for each of the initial conditions the tag system would ultimately lead to a halt or periodicity.[22]

Our main conclusion from this experiment was that, although the distributions of the 0's and 1's seemed to converge to this equilibrium, the mean for the 1's produced was always just a little bit higher than the expected value, while the

[22] Remember the remarks by Minsky and Hayes in this context. See Sec. 6.1.2.

mean for the number of 0's was always a bit smaller than the expected value. In other words, the distribution of the 0's and 1's produced were such that the tag systems seemed to have just enough space to grow a little bit, without implying unbounded growth.

8.5.3 Experiment 5

The goal of this experiment was to further refine the results from experiment 4, and checked the randomness of the distribution of the 0's and 1's through Marsaglia's battery of tests for randomness called DIEHARD. It was clear from the results, that the majority of the tag systems considered pass for at least some of the tests, although there were two that did not pas for any test (**T1** and **T34**). To summarize the results, although these tag systems seem capable of at least some randomness, it is clear that when to so-called "harder" tests are concerned almost all the tag systems fail.[23] In the end, we could not but conclude that a more detailed research would be needed here.

8.5.4 Experiment 6

In this last experiment, we measured the information entropy [Sha48] for each of the tag systems. Since this measure is used to have an idea of the amount of unpredictability of a given discrete source, a high entropy served as indication of the intractability of each of the tag systems involved. For most of the tag systems considered this entropy was relatively high, some even almost attaining the maximum value 1.0. Important to note here is that a side-result of this measurement was the fact that some of the tag systems were capable to produce any combination of 1 and 0 for arbitrary length n, with $0 < n \leq 10$. For most of the tag systems considered however, the total number of combinations found for each n, seemed to decay with increasing n, but this could be due to the size of the sample.

[23] There was only one that passed the difficult birthday spacing test, i.e., **T41**. This tag system is the only one that passed for 9 of the 15 DIEHARD tests.

8.6 Conclusion

If one is honest about the actual results from this experiment, it must be admitted that, besides the results from experiment 2, no real fundamental theoretical conclusions can be drawn from these experiments. Does this mean that the experimental approach on tag systems has failed? Not to our mind.

First of all, although all the conclusions have a clear heuristic basis, and one should thus be very cautious in drawing conclusions on this basis, it is clear that all the experiments indicate the difficult character of this class of tag systems. If we look at the conclusions from experiments 3 – 6, it is clear that these tag systems have certain (heuristic) properties that are often used in the literature as indicators of "complexity", where this notion should here be understood in a vague intuitive sense, and not be confused with complexity as defined in certain branches of computer science like computational complexity theory or algorithmic information theory. Still, given the results from experiments 3 and 4, these tag systems seem to lack just that little bit more needed to become completely random. Since statistical randomness would provide a strong heuristic basis to conclude that all these tag systems should ultimately halt or become periodic,[24] this "just not enough" in fact makes the production of the difficult Immortals? possible. To put this in an intuitive language, the tag systems considered here seem very unpredictable, but are not unpredictable enough to become predictable. Of course, these conclusions are presumptive and need more research.

The results from experiment 1, on the other hand, are a further indication of the difficulties involved in studying these tag systems. The plots from the experiment indicate that the hypothetical intersection points with a line $y = a$ seem to move exponentially fast to the right, with the length of the initial conditions. Although we can hardly conclude anything more on the basis of the results from this experiment, this seems to be a clear indication of the intractability (though not necessary inherent) of this class. To explain this a bit more, even if the numerous conditions tested for **T1** always lead to periodicity or a halt, we have not found any way to prove that this will happen for any initial condition of ar-

[24] Again, remember the remarks by Minsky and Hayes in this context. See Sec. 6.1.2.

bitrary length. The problems involved for proving this, seem to lie in the fact that it seems possible for every given value m, to always find an initial condition that will not have produced a periodic string, nor have led to a halt in less than m iterations.

To our mind, the results from our heuristic and really nerve-racking research on periods in tag systems are the most theoretically appealing. However, as was said, more research is needed here. The results merely offer a preliminary basis for a more theoretically oriented research on periods in tag systems.

Maybe we had expected too much of the experimental approach when we started with it. The time it takes to set-up an experiment, and to study the results is often in complete disproportion with the results one ultimately gets. Still, for us, these experiments have been invaluable to build up our intuition of tag systems. It was in this way that we were able to convince ourselves from a very basic difference between the class of tag systems with $v = \mu = 2$, the class we will prove solvable in the next chapter, and the tag systems studied here. Although we are not able to give a definite structural explanation for this difference, we guarantee the reader that tag systems with $v = \mu = 2$ are really a piece of cake as compared to the tag systems considered here. Tag systems from this solvable class would in fact be completely useless for these experiments, since they almost immediately lead to one of the three general classes of behaviour, once the directly traceable effects of the initial conditions has disappeared.

In part I, Sec. 4.2 we discussed Lehmer's comments on the possibility computers have offered us to disclose the universe of discourse of certain mathematical objects. It has also been our experience that the computer is a very important tool in studying tag systems, and this is not only valid with respect to the presumptive but also with respect to the more theoretical but "humanly impractical" results on tag systems that might be attained.

Chapter 9

Universality and Unsolvability in tag systems: Some questions concerning the usefulness of small universal systems.

> In any case, the writer feels that in view of the efforts expended on the calculation of Sigma(3), for instance, it is rather unrealistic to accept the *mere existence* of a Turing machine that computes Σ(3) as evidence that Σ(3) can be "effectively calculated". This realistic attitude is based, in part, on the writer's experiences in actual computer work (involving large and logically intricate programs concerned with the optimal design of automatic systems). In fact, he feels that, in his work with such programs, he profited greatly from the efforts expended (along with others) to subdue somehow the exasperatingly elusive BB-3 problem, even though this problem seems to be merely a nice exercise in a course for beginners. A very simple and direct answer to the questions raised here may very well be that the writer misinterpreted the definition of 'computability' as stated, for example, by Kleene (...) or Davis (...). In a way, this would be a very grateful outcome. Indeed, the BB-n problem would appear than as an instance of non-computability *in its perhaps most primitive form* [m.i.] and hence as a potential source of new insights regarding the extent to which computers can relieve the human mind of monotonous tasks, setting it free to exercise its powers on the highest level.

Tibor Rádo, 1963.[1]

One of the main motivations behind this research has been to understand better the connection between the general unsolvability of a certain decision problem of a whole class of systems (and the theory underlying it) and the actual discourse of particular (classes of) systems from this class. In this chapter we will consider this connection by starting from a discussion on the significance of *(small) universal systems*. On the one hand, (small) universal systems are *particular instances* of systems with an unsolvable decision problem. On the other hand, research on (small) universal systems can help to gain more insight into the *limits of solvability and unsolvability* and thus make it possible to differentiate between particular subclasses with or without an unsolvable decision problem. Given these two aspects, small universal systems, might help to understand better the link between the general unsolvability of a class of systems and the solvability or unsolvability of particular (subclasses) of such systems.

In a first section (Sec. 9.1), we will give a brief overview of the history of (small) universal systems, and look at some of the arguments from the literature, as to why research on (small) universal systems is considered interesting. We will then argue that, although small universal systems are very important in the context of research on the limits of solvability and unsolvability, they are not useful for a study that starts from the behaviour and properties of particular (classes of) systems (Sec. 9.2). To be more specific, it will be shown that if we want to focus on the actual execution, the known (small) universal systems become theoretical constructions that have no special advantage over systems not known to be universal or solvable.

In a next section (Sec. 9.3) we will argue, through examples from the literature, that if one starts from a study of particular (small) systems – lying at the edge of solvability but not known to be solvable – to understand the limits of solvability and unsolvability, it is exactly an analysis of properties related to the behaviour of particular (classes of) systems that is an important ingredient to make further progress in this area.

In the last section (Sec. 9.4) this whole discussion will be connected to our re-

[1] [Rád63], pp. 80–81

search on tag systems. We will describe some theoretical results on tag systems, that can help to determine their limits of solvability and unsolvability. In the end, it will be argued that the limits of unsolvability in tag systems lie considerably lower relative to those for Turing machines.

9.1 Why are (small) universal machines interesting? A historical account

> The size of the smallest universal machines remains unknown, however, whether or not it is of any serious mathematical interest. With our minds remaining open to the possibility of practical, scientific, or simply philosophical interest we shall examine the area of simple (...) machines.
>
> Allen H. Brady, 1988.[2]

In 1936 Alan Turing constructed the first universal Turing machine, and its significance, however large and theoretical the encoding was, should never be underestimated.[3] First of all, the universal Turing machine is part of Turing's proof of the unsolvability of the halting problem. Basic to this proof is that it is constructive in a quite literal sense: its implication is that you can actually construct a theoretical machine with an unsolvable halting problem. Universal machines are indeed specific instances of formalisms having an unsolvable decision problem.

Secondly, the universal Turing machine can be used to prove other particular classes of formal systems universal and thus unsolvable. Indeed, by reducing a given class of Turing machines containing a universal Turing machine to the class one is investigating, one can prove that that class contains a universal system and is thus unsolvable.

The universal Turing machine is not only important on the level of these more theoretical results, but also because of the techniques Turing developed to construct one. First of all, one has to find a general method for representing any

[2] [Bra88], p. 260

[3] It should be noted that there are several "bugs" in Turing's description of his universal machine. These were first pointed out by Post in his [Pos47].

system from a given class of systems in one machine. I.e. one must be able to find the right translations between the description of and input for a given system and the universal machine that has to represent it. Secondly, basic to the universal Turing machine is the fact that operator and operand, description and input, are manipulated on one and the same level – they become interchangeable. Both techniques can be very useful in reductions from formalism A to B. It are furthermore these techniques that have been important for the influence of the universal Turing machine on the development of some of the first computers (See Sec. 4.1).

9.1.1 Size and definition of universal machines.

Soon after the publication of Turing's paper, the Second World War was a fact. In Sec. 4.1 we already described Turing's contributions to the victory over Nazi Germany. His seminal paper though was hardly known to anyone when the war started. In 1956, a collection of papers, *Automata studies* [MS56a] was published, including many important contributions to the domain of computers and computability. Whatever few people may have read Turing's paper when it was first published, it is clear that about ten years after World War II its influence could no longer be underestimated. One third of the volume is devoted to Turing machines, while many of the other papers are clearly influenced by Turing's work. Two of the papers of this volume are important here, i.e. Martin Davis's [Dav56] and Claude Shannon's [Sha56]. Davis's paper has been influential in the context of universal machines, because it provides an explicit definition of the notion of universality, that will show significant in 9.3.3, Shannon's paper is significant here because of its proposal of a measure of the size for Turing machines.

Davis's definition of universal systems.

In previous chapters the notion of universality has not been defined explicitly. It was assumed that a universal machine is simply a machine that can compute anything computable by any Turing machine i.e. a feature of universal machines often interpreted as the ability to *simulate* any other machine. While this description of a universal machine is not really wrong, it does not take into account properties of the encoding functions for the input of the universal machine, and merely focuses on what the machine itself should be able to do. It is precisely this aspect that is included in Davis's definition. Taking into consideration properties of input encoding motivated Davis's note ([Dav56], p. 167):

> The universality of a Turing machine is manifested by its ability, via a suitable encoding, to perform any computation which would be performed by any given Turing machine. However, the condition must surely be added that the encoding itself be, in some suitable sense, simple. For there would be no such point in claiming universality for a Turing machine for which the encoding would require, in essence another universal machine to carry it out. This raises the problem of explicitly defining universal Turing machine [...]

Davis has, to our mind, correctly pointed out the significance of a *simple encoding*. Indeed, if one would e.g. need a universal machine to encode description and input of a given machine T as the input of the universal machine T_U, in order for T_U to be able to compute what T computes, the universal character of T_U depends too heavily on the universality of another machine.
Before being able to give the definition proposed by Davis, we need some preliminary definitions, the first being that of a r.e. complete set R.

Definition 9.1.1.1 *A set C is r.e. complete if it is recursively enumerable and if for every recursively enumerable set R, there is a recursive function $\rho(x)$ such that:*

$$R = (x|\rho(x) \in C)$$

Definition 9.1.1.2 *With each Turing machine T we associate a set D_T of instantaneous descriptions (ID) defined as follows. An ID α belongs to D_T iff. there exists a sequence of ID's $\alpha = \alpha_1, \alpha_2, ..., \alpha_n = \beta$, such that $\alpha_i \rightarrow \alpha_{i+1}, i = 1, 2, ..., n-1$, with β terminal. I.e. D_T is the set of all ID's α, for which T, when started with α, eventually halts.*

Definition 9.1.1.3 *δ_T is defined as the set of Gödel numbers of all elements of D_T.*

Given these definitions we are now ready to define the notion of a universal machine U_T

Definition 9.1.1.4 *A machine T_U is universal of δ_{T_U} is complete.*[4]

Davis proved that, for any machine T_U fulfilling this definition, the encoding can be entirely accomplished by a non-universal machine. It was exactly this result that formed the main motivation behind Davis's definition. Davis proved this by showing that the encoding can always be done by recursive functions, and showed that any recursive function can be computed by a non-universal Turing machine. He in fact proved a stronger result, namely that every recursive function is *strongly computable*. Basically, a function is considered to be *strongly computable* if and only if it is computable by a Turing machine that has no immortal ID's, i.e. a Turing machine that always halts. This result will show very important in the discussion of the proof of the "universality" of the so-called rule 110 cellular automaton (See Sec. 9.3.3). Since Davis has shown that the encoding of a universal machine satisfying his definition can be done by recursive functions, while every recursive function is strongly computable it follows that the encoding can be done by a non-universal machine.[5] From now on, every time we use the notion of universality, we are using Davis's definition of universality, except when explicitly mentioned otherwise.

Measuring the size of Turing machines.

Shannon's contribution [Sha56] not only contains a proof of the fact that any Turing machine can be reduced to a Turing machine with only two internal states *or* one with two symbols, but proposes a method to measure the size of

[4] In Davis's [Dav57], a further restriction was added to this definition in order to restrict the number of steps performed by T_U.

[5] The notion of strong computability was some years later used by Shepherdson [She65] who constructed a Turing machine U that always halts iff. the function f it computes is recursive (computable) i.e. it strongly computes the recursive functions. Similar constructions and the notion of strong computability underlying it, are nowadays also used in the context of research on CA in connection to discussions on the classification of CA as proposed by Wolfram (See [BS00, Sut03, Sut05].)

universal Turing machines. The fact that any Turing machine can be reduced to two-symbolic machines can be used to simplify certain unsolvability proofs, since one merely has to look at the 2-symbol machines. Minsky's second proof of the universality of tag systems for example uses this feature in order to improve his previous proof: instead of a minimal shift number $v = 6$, he was able to reduce it to 2. More important in this context though is Shannon's measure for small universal machines: he suggested to use the product of the number of states and the number of symbols as the measure of the size of Turing machines: ([Sha56], p. 165):

> The results we have obtained, together with other intuitive considerations, suggest that it is possible to exchange symbols for states and vice versa (within certain limits) without much change in the product. In going to two states, the product in the model given was increased by a factor of about 8. In going to two symbols, the product was increased by a factor of about 6, not more than 8. [...] At any rate, the number of logical elements such as relays required for physical realization will be a small constant (about 2 for relays) times the base two logarithm of the state-symbol product, and the factor of 6 or 8 therefore implies only a few more relays in such a realization. An interesting unsolved problem is to find the minimum possible state-symbol product for a universal Turing machine.

Shannon's measure is thus rooted in the fact that adding more symbols makes it possible to reduce the number of states and vice versa, thus leading to the idea of the interchangeability of number of states and number of symbols. From now on, a class of Turing machines of size $a \times b$ will be indicated as $TM(a, b)$, where a is the number of states and b the number of symbols, similarly, UTM(a, b) is the class of universal Turing machines with a symbols and b states.

The quote is ended with, as far as I know, one of the first formulations of the problem of finding the smallest universal Turing machine – a problem still unsolved nowadays – however without clearly stating why this kind of research could be interesting. Shortly after the publication of Shannon's paper, several computer scientists entered the "competition" of finding smallest universal machines.

9.1.2 Small universal machines: an overview.

The competition was started around 1958, with Ikeno's proof of a universal machine in the class TM(10,6) [Ike58]. Two years later Watanabe proved the existence of the class UTM(8,6) [Wat60], Minsky improved the result to UTM(7,6). Watanabe in his turn improved Minsky's result with a result of UTM(8,5) and even UTM(6,5) [Wat61], later Minsky proved the existence of the class UTM(6, 6) [Min62b]. Finally Minsky was able to construct a universal machine of size 7x4 and was thus able to prove that the class UTM(7,4) is not empty [Min62, Min62b]. For some years nobody seemed to be really looking any longer for smaller machines, but for the last 20 years or so several researchers have searched and constructed several other small machines. The smallest known classes containing a universal Turing machines are: TM(18,2) (Neary 2006, mentioned in [NW06c]), TM(9,3) (Neary 2006, mentioned in [NW06c]), TM(7,4) (Minsky 1961, [Min61]), TM(5,5) (Rogozhin 1982, [Rog82]), TM(4,6) (Rogozhin 1982, [Rog82]), TM(3,9) (Kudlek and Rogozhin 2002, [KR02]) and TM(2, 18) (Rogozhin 1996, [Rog96]).

Besides proving for specific classes of Turing machines that they contain a universal machine, it is of course equally interesting to find classes that can be shown to be solvable and thus cannot contain universality. Indeed, by looking at both sides of the problem one can try to bridge the gap between known universal classes and thus classes that are unsolvable, and known solvable classes. As far as solvability is concerned, it should be noted that Minsky mentions that he and Bobrow had been able to prove that the class of machines TM(2,2) is decidable, through a reduction to thirty-odd cases (See [Min67], p. 281), a shorter proof was published by Pavlotskaya [Pav73]. She also proved that the class of machines TM(3,2) is solvable [Pav78]. She furthermore proved that the class TM(2, 3) is solvable (unpublished).

Besides small universal Turing machines, and the associated study of the frontiers of solvability and unsolvability in Turing machines, there are of course many results for other computational systems in this context, but these will not be discussed here. An overview for some of these results can be found in [Mar00].

Now that we have an idea about the smallest universal machines, we are finally ready to dig into the question as to why such (small) universal systems are considered interesting. Minsky gave two reasons. First of all, as he writes in [Min62b], p. 230:

> This section is concerned with explaining the encoding for the machine, and how it is interpreted. As such, the discussion can be regarded as concerned with questions of digital computer programming. As a matter of "programming appreciation" it seems interesting that such a small structure can be so complicated.

Indeed, especially in those early years of computing, it must have been quite surprising that such small structures can be so complicated (sic)! In the same paper, Minsky furthermore remarks ([Min62b], p. 237):

> A very small machine might turn out to be useful in construction of a very simple unsolvable decision problem, e.g., one in elementary number theory.

Minsky clearly believed at that time that research on (small) universal machines might show useful in finding simple examples of unsolvable decision problems. While Minsky was, for some time, convinced of the significance of doing research on (small) universal systems, he stated only some years later ([Min67], p. 277):

> The reader is welcome to enter the competition (...) although the reader should understand clearly that the question is an intensely tricky puzzle and has essentially *no* serious mathematical interest.

As is pointed out by Brady, around 1964 John McCarthy had a similar reversal of opinion (John McCarthy in a private conversation with Allen Brady ca. 1964, cited in [Bra88], p. 648):

> We thought if we were to find the smallest universal universal machine then we could learn a great deal about computability – of course that wouldn't be so

Despite the negative comments by two pioneers, research on smaller universal systems is still a serious research domain. Nowadays there are different people, with different research interests, who work in this domain. One of the main motivations behind this research is that small machines can help us to understand better and know the frontiers between solvability and unsolvability.[6] Another reason is pointed out by Stephen Wolfram, who is most famous for his research on cellular automata ([Wol02], p. 5):

> [...] there are systems whose rules are simple enough to describe in just one sentence that are nevertheless universal. And this immediately suggests that the phenomenon of universality is vastly more common and important – in both abstract systems and nature – than has ever been imagined before.

Since Wolfram is convinced that everything in our world is governed by algorithms, finding very low limits for universality, implies for Wolfram that almost everything in nature is of the same kind of complexity. Later on in his book [Wol02] he even talks about a new kind of Copernican revolution, man's complexity being set equal to the complexity of almost anything else in nature. I will not discuss Wolfram's opinions here, because this would automatically shift to a discussion on the computational character of our world, a discussion which falls beyond the scope of this text.

In the end, I think the most important reason for doing more research on (small) universal systems is to determine the boundaries between solvability and unsolvability. In order to further study this problem, it is very important to have a look at some of the reasons why (small) universal systems are *not* interesting at all.

9.2 Why are (small) universal systems not interesting?

As is clear from the previous section, there are two basic reasons why universal systems are very important. First of all, they are examples of particular instances with an unsolvable decision problem, and are thus important tools to

[6] This is one of the objectives of an international research group, including Kudlek, Margenstern, Pavlotskaya and Rogozhin running under the heading *Small Universal Turing machines.*

prove certain decision problems unsolvable. Secondly, if one constructs a universal system, one can determine upper limits for unsolvability.
There are basically two ways to construct a universal system:

1. One can construct one specific universal system in class *B*, by finding a way to represent any system of class *A*, containing a universal system, into a system of class *B*. This system must be encoded in a way that it will compute what any system from *A* computes, following the restriction of Davis's definition. In this case, one does not use a general scheme for compiling any system from *A* into some system of *B*, but rather a general scheme to encode both description as well as input of any system in *A*, into the input of one and the same system from *B*, constructed such that it is able to interpret both instructions and input of any system from class *A* and is thus universal. Typical here is that the description of *B* itself is a kind of interpreter of the encoding of description and input of a system from *A*.

2. One can provide a general compiling scheme which can then be used to translate any system and its input from class *A* to some system of class *B*. One can then apply the scheme to a universal system from class *A* in order to construct a universal system of class *B*. A typical feature of this method is that the description (instructions) of a system in *A* will be encoded in the description (instructions) of a system in *B*, while the input will be encoded as the input. This process is more closely connected to compiling, than to interpreting: once the encoding has been fixed, the system runs, without having to interpret the description of the system it simulates.

As was said, in using one of these methods, it is possible to demarcate the limits of solvability and unsolvability for a given class of formal systems. It should be noted that it is the first method that is normally used in constructing small universal Turing machines.
For now however, there is still a rather large gap between the classes of Turing machines known to be solvable and those that contain a universal Turing

machine and are thus unsolvable. It are the in-between classes that thus offer the greatest challenge. For now, there is no clear method for constructing a universal Turing machine for these in-between classes like e.g. TM(2, 10). Neither is there a clear indication of the solvability of many of these classes. Given this problem, the question indeed arises whether the kind of encodings to be discussed here – starting from the encoding itself rather than from an existing (class of) machines – can help us to find the correct boundaries, bridging the now existing gap between known solvable classes and known unsolvable classes?

In Ch. 7 we saw how one can define certain constraints for tag systems, that help to generate tag systems that seem very intractable, there being no obvious way to decide their behaviour. Many of the experimental results found in the previous chapters indicate that these tag systems are indeed very hard to get a grip on. These results however can only be used as heuristic support for the possibility of unsolvability of a given class of systems. Also in case of Turing machines, it is believed that the known limits of unsolvability can be lowered. But neither in the case of Turing machines has one been able to prove universality for relatively small classes.[7]

This problem becomes even more intricate in the light of Post's problem (first stated by Post in [Pos44]) solved independently by Friedberg [Fri57] and Muchnik [Muc56], the problem of whether there can be two recursively enumerable sets that are non-recursive, such that the first is recursive relative to the other, but not vice versa, i.e., an oracle for solving the decision of the first would not result in a solution for the second, while the oracle for the second would give solutions for both problems. To put this in the context of universal machines, the problem asks for the existence of a recursively enumerable set that is not recursive and not Turing complete, i.e., universal. The method developed by Muchnik and Friedberg, i.e. the priority method, is important in the context of research on degrees of unsolvability. However, as is pointed out by Sutner, who has studied Post's problem in the context of cellular automata ([Sut05], p. 50):

> Alas, the sets constructed via priority arguments appear somewhat ad hoc and

[7] More will be said about this in Sec. 9.3.

> artificial. It is therefore tempting to search for "natural" examples of intermediate degrees, examples that would presumably arise as a side-effect of a less complicated construction. By natural we here mean that the generating device should admit a very simple description as opposed to, say, invariance under automorphisms of the semilattice of c.e. degrees. Of course, there are well-known results to the effect that every c.e. degree appears in a certain mathematical context. For example, all c.e. sets are Diophantine and can thus be defined by an integer polynomial. Similarly, every c.e. set is Turing equivalent to a finitely axiomatizable theory and word problems in finitely presented groups may have arbitrary c.e. degree. But the point here is to obtain a specific example of an intermediate degree using a reasonably simple mechanism to do so.

Sutner has asked whether it is possible to find "natural" examples of an intermediate degree of unsolvability. To explain a bit better what he means with "natural", it is interesting to mention that he e.g. thinks it possible that a certain cellular automaton, known as rule 30,[8] could be such a natural example of an intermediate degree [Sut03]. The behaviour of this formally very simple automaton, is very complicated and is used as the default random number generator in *Mathematica*. Its behaviour has been identified as class 3 cellular automaton behaviour by Wolfram [Wol02], which is characterized by random, non-periodic patterns. The other classes identified by Wolfram are classes 1, 2 and 4. Class 1 behaviour evolves to a fixed homogeneous state, class 2 behaviour converges to separated periodic structures while class 4 behaviour, also identified as "complex" behaviour by Wolfram, is characterized by complex patterns of localized structures that interact with each other against a periodic background. Rule 110, which we will discuss in Sec. 9.3.3, is considered to be in class 4. Cellular automata in this class, are considered universal. As was said, according to Sutner, rule 30 might be a "natural" example of an intermediate degree ([Sut03], p. 367):

[8] See [Wol02]. Some sample pictures of the behaviour of this automaton as well as the description of the rues are available through the Wikipedia entry on rule 30. One can of course also very easy implement the automaton on one's computer to study the behaviour of this automaton.

> [...] the orbits of rule 30, or, at the very least, of similar Class 3 automata, could be of intermediate degree: undecidable, yet less complicated than the Halting problem.

In the light of the possibility of finding "natural" very simple examples of intermediate degree, we are faced with a double problem in the context of studying limits of solvability and unsolvability. We are confronted with a number of classes of Turing machines for which it is not known whether they are solvable or unsolvable, and we do not know whether these in-between classes, if unsolvable, are universal or not. Especially those classes for which it is known to be very hard to find methods to predict them, while it is far from clear in what way they could be shown to be universal, given the size of their description, offer to our mind the greatest challenge here.

We will consider two approaches here that can be useful to tackle this problem. One can use the more classic technique of proving or arguing for certain properties of a given class, by *constructing* systems in that class that encode specific functions, like the construction of a universal machine. Or, one can start from a detailed analysis of the behaviour of several specific instances from a given class.[9]

Both methods – encoding through construction or analysis of behaviour – clearly have their merits. As we will for example see in Sections 9.3 and 9.4 constructing certain systems, encoding specific functions over the integers, can provide useful information about the difficulties that might be involved in proving a given class solvable. We are convinced, however, that if we want to bridge the gap between solvable and unsolvable classes, it is equally useful not to solely focus on the specific interpretation of a given class of systems in terms of what functions can be computed through the description of systems from the class, but rather on the actual discourse of the systems in a given class, (initially) abstracting away from semantics. In analyzing the behaviour one might for example find encoding methods rooted in the behaviour of a given system rather

[9] Of course, these are not the only approaches possible here. In Ch. 7 we already argued for the significance of constraints for differentiating between tag systems that are solvable, and tag systems for which this is not known. The more precise notion of a decidability criterium has already been proven its merits in the literature. See [Mar00] for an overview.

than in the description of the rules for the system. It should be noted however, that both approaches cannot be strictly separated from each other.

In Sec. 9.3 we will discuss some examples from the literature where the analysis of the behaviour of systems belonging to a given class is basic to certain results in the context of solvability and unsolvability. This makes the connection between the general unsolvability of a class of systems and concrete instances of systems from the class more explicit. In this section we will show that universal systems, able to compute anything computable by Turing machines, are far from interesting in the context of studying the *behaviour* of specific (classes of) systems, in order to gain new results. We will argue that it is because specific encodings are involved in the construction of a universal system (thus focussing on the computability concept, on the "semantics" of computation), an analysis of the behaviour of these systems offers no extra's as compared to an analysis of the behaviour of particular (classes of) systems for which it is not known whether they are solvable or not.

We will discuss two examples of universal systems, the first being Minsky's 4-symbol, 7-state machine, an example of the first reduction method. The second, a universal tag system, is an example of the second reduction method.

9.2.1 Minsky's 4-symbol, 7-state machine.

All the small universal Turing machines known so far, result from an application of the reduction method (method 1). One of these machines is Minsky's 4-symbol, 7-state machine, shown to be universal because it can represent any tag system with a shift number $v = 2$. For this machine, both production rules as well as input for the tag system to be simulated are encoded on the tape. The machine is then able to interpret the rules of the tag system encoded on its tape, repeatedly applying them to the encoding of the strings produced in the tag system, starting with the initial condition of the tag system. As was already explained in the intermezzo from Sec. 6.2.3, the tape of the machine is divided into three regions:

...	0	0	**Rules Region**	**Erased Region**	**Sequence Region**	0	0	...

The rules are encoded in the rules region, and the strings produced (starting from the initial condition), are encoded in the sequence region. Before explaining the encoding, it is important to add that Minsky assumes that the tag systems simulated have a special halting letter, that leads the Turing machine to a halt. In order for Minsky's encoding to work, it is important that this halting symbol is the first letter a_1 from the alphabet. It should also be noted that the machine can only represent tag systems with $v = 2$. However, since it had already been proven by Minsky that it is possible to construct a universal tag system with $v = 2$, this is no problem (See Sec. 6.1).
Now, given a tag system, with μ symbols $a_1, a_2, ..., a_\mu$, and a shift number $v = 2$, the production rules all of the following form:

$$a_i \to a_{i,1}a_{i,2}...a_{i,r(i)}$$

The lengths of the words associated with each of the letters a_i of the alphabet are equal to $r(i)$. The production rules encoded on the tape of the Turing machine are then encoded as strings of 0's and 1's as follows. First we calculate for each a_i a number $\overline{a}_i$:

$$\overline{a}_i = 1 + \sum_{j=1}^{i-1}[r(i)+1]$$

For each letter a_i, the encoded word R_i corresponding to it is encoded as:

$$R_i = 110^{\overline{a}_{i,r(i)}}1...10^{\overline{a}_{i,2}}10^{\overline{a}_{i,1}}1$$

These encodings are placed in the rules region going from left to right, starting with R_1, ending with R_μ. The strings produced by the tag system are encoded by a combination of the letters X and B, the B's being used to mark the end of an encoding of one of the symbols a_i. Given an initial sequence $a_{i,1}a_{i,2}...a_{i,n}$, each of the symbols, except for the last is encoded as: $X^{\overline{a}_{i,j}-1}B$. The last symbol in the sequence is encoded in the same way, but without the letter B at the tail. After production rules and initial condition have been encoded in this way on the tape, the machine is started in state q_1, the reading head being on the square containing the symbol X to the extreme left of the sequence region. To give an example, let us use this encoding scheme to the following tag system, based on

T1 and introduced as an example by Minsky. The word w_1 corresponding to the halting symbol a_1 is set to $a_1 a_1 a_1$, $w_2 = 00, w_3 = 1101$. Using the encoding scheme we get: $\overline{a}_1 = 1, \overline{a}_2 = 5, \overline{a}_3 = 8, R_0 = 11010101, R_1 = 11000001000001, R_3 =$ $11000000001000001000000001000000001$. If the tag system is started with the initial condition $I = 1001$, the machine's initial tape will be:

...000	$110^8 10^5 10^8 10^8 1$	$110^5 10^5 1$	11010101	$X^7 B X^4 B X^4 B X^7$	000...

So how will our machine interpret these rules? If the encoding of a symbol from a tag string is encountered by the machine in the sequence region, the Turing machine must be able to locate the corresponding word in the rules region. This is done through $\overline{a}_i$, since the value of $\overline{a}_i$ gives us the number of 1's the machine has to traverse to reach the appropriate word, using the double 11 as a mark of the end of a word. The left 1 from this pair is not taken into account in this word-locating process, so that the process will always count $r(i) + 1$ 1's in order to perform this allocation. The symbols contained in the words are always represented by one extra 0, as compared to the symbol's representation in the sequence region, because the copying process, used to perform the tag operation, skips the first 0 of each symbol copied.

When the machine is started it moves back and forth, counting X's in the sequence region and counting 1's in the rules region. If the first B is encountered, the process will have located the representation of the word corresponding with the representation of the symbol scanned. It then goes again back and forth, copying each of the symbols of the word, separated by a 1, at the end of the sequence represented in the sequence region. Each 0 in the representation of the symbols of the words (except for one) yields an X in the sequence region, and each 1 in the rules region yields a B in the sequence region. When the double 1 is encountered, the copying stops, and the machine's tape is restored. During this process, the encoding of the first two letters in the sequence region will have been erased. The encoding of the second letter can also be erased, because, the letter B which separated it from the first had been set to X during the rule-allocation process. Then, when the process is finishing up, the first X's at the extreme left of the rules region are erased, until the first B is encountered (which will also be erased). It is in this way that the erased region must become larger and larger when simulating tag systems that do not halt.

Readers who want a better understanding of the operations of this universal Turing machine, and are not yet familiar with it, are advised to go through the movements of the universal machine themselves. They will soon understand how it does the job. But, before being able to do this, one of course needs the transition table of the machine. The triples associated with each symbol-state pair are: symbol written, direction of motion, new state. If a couple is used instead of a triple, it means that the next state is the same as that which the machine was in. The machine is started in state 1, scanning the leftmost X in the sequence region:

Table 9.1: Table of Minsky's 4x7 universal Turing machine

	q_1	q_2	q_3	q_4	q_5	q_6	q_7
X	0L	0Lq_1	XL	XL	XR	XR	0R
0	0L	XR	HALT	XRq_5	XLq_3	BLq_3	XRq_6
1	1Lq_2	BR	BL	1Lq_7	BR	BR	1R
B	1L	XLq_3	1Lq_4	1L	1R	1R	0Rq_2

The machine will halt iff. it encounters the representation of the halting symbol, i.e. BB. It will then go from state 7, to state 2, to state 3 and result in a halt. Of course, one does not necessarily need a halting symbol in the tag system one wants to simulate. One just has to be careful that the encoding of the words, is always preceded by a sequence of the form $11(01)^x$, adapting our $\overline{a}_i$ according to the value of x.

As was said, since this 4 by 7 machine is able to represent any tag system with $v = 2$, it is universal. Given its size, it is known as one of the smallest universal Turing machines ever constructed. Besides its smallness and its consequent significance for the more theoretically minded programmer, one must dare ask whether there is any other reason as to why this machine would be more interesting than certain other machines? As far as Minsky's earlier arguments are concerned, it is far from clear how such machine could help to find simpler examples of unsolvable decision problems, Minsky himself having seriously revised his opinion here. Its basic significance is that it helps to lower the limits of unsolvability for Turing machines and it is thus an important theoretical result in the context of this research. But is it not possible to learn something more from this specific instance of a Turing machine with an unsolvable decision problem in relation to limits of solvability and unsolvability?

A good starting point to tackle this question could be a study of the behaviour of this machine. So, in what way can we learn something from the execution of a universal Turing machine on a computer? The answer to this question is: nothing that cannot be learned already from a study of the behaviour of other machines. There are two main reasons for this answer, the first being countered in the meantime by a number of recently published papers by Neary and Woods.

One of the most important disadvantages of working with these small universal Turing machines, at least until one year ago, was the fact that they exponentially slow down in simulating other Turing machines. This is the case because the known small universal machines are based on their ability to represent tag systems with $v = 2$, while the simulation of Turing machines in this class of tag systems is itself inefficient. If one then wants to look at the actual execution of a machine with an unsolvable decision problem, looking at millions of iterations this inefficiency is a very real obstacle. When I first developed my arguments, I thought this was a very strong argument against the usefulness of (small) machines. It was only when I started writing this text that I found a paper by Neary and Woods in which small polynomial-time universal machines are constructed, machines for which the time needed to perform a computation is not greater than a polynomial function of the size of the input. In [NW06d] they constructed universal machines that directly simulate Turing machines in the classes UTM(11,3), UTM(7, 5), UTM(6, 6), UTM(5,7) and UTM(4, 8). These machines are all a bit larger than the small universal machines that simulate tag systems. In the meantime, they have also been able to prove that tag systems, with $v = 2$, efficiently simulate cyclic tag systems [NW06a], after having proven that cyclic tag systems efficiently simulate Turing machines [NW06b]. Taking these two results together, they showed that for any Turing machine T, one can construct a tag system that computes the computations of T in polynomial time $O(t^4(\log t)^2)$. From this result it immediately follows that the small machines by Rogozhin, Minsky, Baiocchi et. al, are polynomial time simulators $O(t^8(\log t)^4)$ of Turing machines.[10]

Due to the results by Neary and Woods one can no longer hold on to the idea that an exponential slow down in the execution of universal machines forms an obstacle for studying their actual execution. There is still a more basic disadvantage to be discussed here, summarized by the following question: what kind of new information can we get from studying the behaviour of universal Turing machines, as compared to a study of the behaviour of the systems they can represent? Indeed, since they are specific instances of unpredictable machines, it might seem interesting to study the machine's behaviour.

In the previous chapter we studied tag systems without taking into account what functions can be encoded in these systems. Because of this, we did not need to structure our initial conditions in any specific way, and could use arbitrary conditions. So what would happen to our universal machine when feeded random initial conditions?[11] In order to check this I programmed Minsky's machine and tested it for numerous initial conditions, having a random length between 200 and 400, the starting position of the head also being determined at random. The machine was always started in state 1. For every of these conditions, the machine always entered a halt, a simple loop or a Christmas tree.

Simple Loops and Christmas trees. Machlin and Stout [MS90] developed several methods to detect infinite loops in Turing Machines. These methods were

[10] As is pointed out by Neary and Woods [NW06c], the advantage of this result is the fact that one can now really think about applications. In their [NW06c], Neary and Woods sketch several examples. One of the examples is the proof of the universality of splicing tissue P systems, which are studied in the context of computational biology. It was shown by Rogozhin and Verlan [RV96] that one can have small such universal systems, again via tag simulation, these machines now being able to simulate Turing machines efficiently.

[11] Note that this question should not be confused with the idea of making sure that the universal machine strongly computes a recursive function f, i.e. that it halts on the encoding of every ID from the Turing machine computing f, since the universal machine itself is not recursive!

developed in order to easily exclude a large number of Turing machines as possible Busy Beaver winners.[12]

For a word W and a state r, W_r means that the machine is in state r examining the rightmost symbol of W. The symbolic expression 0^* is used to indicate infinite occurrences of 0. A Turing machine T is said to enter a simple loop if the behaviour of T is one of two following forms.

1. Some tape configuration is repeated infinitely often. That is, there is a state s and words X and Y such that at some time step the tape configuration is $0^*X_sY0^*$ and the same tape configuration is reached at some later time step.
2. T periodically moves to the right (or left) in that there is some state s, words X and Y and a nonempty word V, such that at one time the tape configuration is $0^*X_sY0^*$, while at some later time the tape configuration is $0^*X_sYV0^*$, and between these times, T never moved left of the right edge of Y.

For the initial conditions tested, Minsky's machine always entered a simple loop when from a given time on, its state remained constant, the machine moving ad infinitum to the right or left. To give an example, the machine entered a simple loop of the second form when, from a given time on, the machine never leaves state 2, never again moving to the left, changing each 0 encountered to X.

The behaviour of a machine T is said to be a Christmas tree if the behaviour of the machine fulfills two conditions. For the full details of the definition of a Christmas tree we refer to [Kop81]. Important for us is that, given nonempty words U, V, and X, if the tape configuration at some time is $0^*UV_s0^*$ it must be equal to $0^*UXV_s0^*$ at some later time, where the machine sweeps back and forth growing a periodic middle part. For example if the tape configuration at time i is:

$$\underbrace{BX100111BXXBB11B00}_{U}\underbrace{\underbrace{XXXXXXX...X}_{n}BXX001BXXXXBX11}_{V_s},$$

and at some later time j it is equal to

$$\underbrace{BX100111BXXBB11B00}_{U}XXXXXXX...X\underbrace{\underbrace{XXXXXXXX....X}_{n}BXX001BXXXXBX11}_{V_s},$$

the behaviour is identified as a Christmas tree, of course having checked that this kind of behaviour remains constant. We did not have the time to write a code that

[12] For an explanation of the Busy Beaver Game, see 9.3.1.

> is able to correctly detect the several kinds of what Machlin and Stout call infinite loops, since that would probably have taken several weeks, the actual design and programming having taken up most of Machlin's PhD [Kop81]. Instead we developed a code that easily detected a halt or simple loops, interrupting it when neither a simple loop nor a halt was detected after 1.000.000 iterations. If this was the case, we were able to conclude for christmas trees based on a more interactive analysis between me and my computer, checking whether the machine keeps on producing strings of the form $0^*UX_iV_s0^*$, where V and U are constant words and X_i is a repetitive string. After several christmas trees had been detected, we also programmed part of the general structure of the tree most frequently detected to automate the detection. After this was done, we frequently interrupted the code to again perform our interactive analysis. After this test, we could not but conclude that the machine always halted, entered a simple loop or a christmas tree for the thousands of initial conditions tested, although our test is not completely water-proof.

When Minsky's machine was tested for thousands of random initial conditions, all the conditions led to a halt, a simple loop or a Christmas tree after at most 1.000.000 steps. Given the results from experiment 1 for our 2-symbolic tag systems, one is of course led to the question why this universal machine far more quickly leads to predictable behaviour as compared to the very simple and not proven to be universal class of tag systems. An intuitive answer to this question is that the inner workings of the machine are in a way too "specialized": it was developed to be able to interpret a certain class of specially encoded initial conditions. The instructions of the machine are developed to be able to compute something very specific i.e. that what can be computed by tag systems with $v = 2$. As a consequence, input that does not have the structure needed for the machine to perform the "simulation", will in most of the cases lead to predictable behaviour in a relatively small number of steps.

I will not go into a detailed analysis explaining why one should expect the machine to lead to such behaviour for the majority of conditions tested, but it is important to give at least some indications to understand what is meant with

"the machine is too specialized". Let us first look, at what happens if we only slightly disturb the encoding. For example, let us change the first 1 to the left of the leftmost X to X or 0, using the example of an encoding of a tag system as described above. This small change will completely disturb the simulation, erasing at certain points more than v symbols. The reason for this is very simple. In counting the number of 1's to the left, to find the right production rule to be applied, one 1 will be missing and thus no B will be printed at the end of the sequence in the sequence region. This means that for any encoding of a word tagged at the end of the encoding of a string produced in the tag system, two symbols will have become 1. To explain this, suppose for example that the last symbol in our sequence is the encoding of 0, i.e. XXXX, the B being omitted since it is the last symbol of our sequence. If the following word to be tagged is the encoding of 1101, we would get: $\mathrm{XXXX}\underbrace{\mathrm{XXXXXXXXBXXXXXXXXBXXXXBXXXXXXXX}}_{1101}$.
If it is the case, that XXXXXXXXXXXXB is the encoding of a relevant letter, the counting process of X's to the left and 1's to the right, used to identify the right production rule will fail: there are less 1's than X's, so the machine will end up somewhere to the left of the rules region and consequently get into a simple loop, as described in the intermezzo. A similar reasoning can be made in setting one of the X's in the encoding of the first symbol equal to 1 or B.

Of course in using random input, the chance that an initial condition will be generated that resembles the kind of conditions needed for the simulation, is very small and it is thus important to ask why the machine will most probably lead to some kind of predictable behaviour, after a small number of steps when given such random conditions. Looking in more detail at the instruction table of Minsky's universal machine, one quickly understands that it is indeed in a way too "specialized", the content of the tape having too fulfill, for each state, certain constraints for the machine not to get into a loop or halt.

For example, if the machine is in state 1, and it is at the left of the leftmost symbol, it will get into a loop. As we already saw, this will happen if the number of 1's to the left of the first X scanned in state 1 is less than the number of X's between this first X and the first B. Starting with a random initial condition which already has this form, will lead to this kind of cycle. If the machine is in state

2, the only way to leave this state is that an X or a B is read. Now, if an X is scanned, and there are no B's left to the right of X, the machine must also lead to a loop, since in the end, all X's will be set to 0, thus making it impossible to go from state 2 to state 1, nor to go to state 3 (since there are no B's). The machine will get into a simple loop, moving to the right for eternity (theoretically speaking). This kind of loops are generated by the fact that, while the machine's instructions are adapted to some kind of balance between number of 0's or 1's to the left, there is also some kind of balance between the number of moves to left and right. This will often not be the case for random conditions, causing the machine to move ad infinitum either to the left or to the right.

Furthermore, the machine always halts when it scans a 0 in state 3. State 3 can be entered from states 2, 5 and 6. If state 2 was entered from state 1 the transition to state 3 cannot lead to a halt. This is the case because, state 3 is entered from 2 if the last symbol scanned was a B. State 2 is entered from state 1 because a 1 has been scanned. Every symbol between the symbol 2 squares to the left of this 1, and the B causing the transition to state 3 will be equal to X or B, so a halt can never occur, since in its movement to the left during state 3, at least one symbol will be equal to B, i.e. the 1 having caused the transition from state 1 to state 2. However, if state 2 was entered from state 7, it is possible for the machine to halt: if the machine scans a B in state 7, setting it to 0 and the symbol to the left of this B is also equal to B, the process will terminate. Indeed, the machine was encoded in order to guarantee a halt if we would have two consecutive B's in the sequence region, in order to simulate a halt in tag systems, a complication which is to our mind far from necessary since this introduction of a halting symbol for tag systems is completely artificial and, in my opinion, too much adapted to Turing computability.[13] A halt might furthermore occur if the machine is in state 5 or 6, and the symbol to the left of the last 0 scanned is equal to 0.

As is clear from this reasoning, at each state transition the machine must fulfill

[13] Of course, there are reasons to introduce such a symbol in the framework of tag systems, e.g. in order to guarantee that a computation halts, but I think that it is better to use the original formulation by Post of a halt in tag systems. This might complicate any encoding process, but it is closer to, or in a way even more respectful towards, tag systems.

very special conditions in order not to get into an infinite loop or lead to a halt. While it is guaranteed that this will not happen as long as one is working with the right encodings, it is very hard to find random conditions that fulfill these conditions at every iteration step for a long number of iterations. Comparing this situation with the results from experiment 1, the situation as sketched here for universal machines is rather disappointing, at least, from the more experimental point of view.

If we want to study the behaviour of particular (classes of) machines we are led to the conclusion that Minsky's universal machine will most probably offer us no more than the information we can already get from running the systems it simulates. While other small machines, such as those by Rogozhin, were not tested in this same way, due to a lack of time, it is assumed here that they can be analyzed in a similar way, given the specialization of the machine to simulate tag systems.

A further indication of the fact that these machines are, from a certain point of view not very interesting, is the fact that these machines offer no challenge as far as the Busy Beaver game is concerned. The (generalized) Busy Beaver game is to determine for a given class of Turing machines, defined through the number of states and symbols, what is the largest number of 1's $\Sigma(m,n)$ printed by one of these machines when started with a blank tape. Minsky's machine, as well as all the other small universal Turing machines I know off, always immediately enter a simple loop when started with a blank tape, and can thus not be winners of the Busy Beaver game in their respective classes. In general, the Busy Beaver game is unsolvable: there is no effective procedure to calculate the value of Σ for any arbitrary class of machines. Although the fact that the small universal machines are no challenge with respect to determining the value of Σ in their respective classes is, of course, completely unproblematic from a theoretical point of view it is, to our mind, from a more intuitive point of view.

To summarize, we hope it is clear that the instructions of the universal machine as described by Minsky (and a similar reasoning can be applied to the other machines) are in a way too "specialized" to make it an interesting machine in the context of an analysis of their behaviour. We will not really get any more information from these machines on this level, as compared to the direct execution of the tag systems or Turing machines it is able to simulate. It is thus more efficient, avoiding encoding processes and the time needed to simulate one computation step of a tag system or a Turing machine, to look at the behaviour of the systems directly instead of through the framework of their simulator.

9.2.2 Generating a universal tag system.

In Sec. 6.1.1, it was shown how Minsky proved that any Turing machine can be reduced to a tag system with a shift number $v = 2$. He provided a general compiling scheme for translating any Turing machine to a tag system. This is thus an application of the second method mentioned in 9.2 for constructing universal systems. Fascinated as I was (and still am) with tag systems, I of course wanted to use Minsky's scheme in order to construct a universal tag system. Using e.g. Minsky's 4-symbol, 7-state machine, I would then be able to have a tag system able to represent any other tag system with $v = 2$, since it would simulate a simulator of such tag systems. However, I soon learned that this universal tag system is far less interesting than e.g. **T1** on the level of the execution of either of these two systems. To begin with, it was very clear that this universal tag system would be rather gigantic as compared to e.g. the simplicity of **T1**. Secondly, not knowing about the result by Woods and Neary at that time, the universal tag system would suffer from an exponential slow-down, the length of its input being determined by the binary number represented on the tape of the Turing machine. Still, I wanted to program a universal tag system. Given the fact that one needs hundreds of production rules in order to simulate the operations of the universal Turing machine constructed by Minsky, I wrote a program that generated the universal tag system, sparing myself lots of work.
Since the encoding from Turing machines to tag systems is based on 2-symbolic Turing machines, the tag systems having a shift number $v = 2$, I first had to translate the 4 by 7 machine into a binary machine. Minsky himself provided such a translation, but there are many errors contained in the encoding. Finally I ended up with a machine in the class UTM(27, 2) – I will not give its instruction table here – and in the end I decided to use Rogozhin's UTM(24, 2) machine.
The following method was used to determine the letters of the alphabet for

the universal tag system. The letters from Minsky's encoding without index, $A, \alpha, B, \beta, C, c, D_1, D_0, d_1, d_0, t_1, t_0, x$ were given a numerical value going from 42 to 58. For simulating each state-symbol instruction the tag system needs 15 production rules and thus different symbols, (except when the tag system encounters a halting state). Each of the letters needed to perform the simulation of one of these instructions, performed by the machine in a given state q_i scanning the symbol s_i, is determined by concatenating to the number associated with each of the symbols $A, \alpha, B, \beta, C, c, D_1, D_0, d_1, d_0, t_1, t_0, x$, the index of the state the machine is in and the symbol scanned. For example 42020 is to be interpreted as $A_{2,0}$ – the symbol A used for simulating the instructions when the Turing machine is in state 2, scanning a 0. Before giving the production rules for our universal tag system, it should be noted that the words associated with the letters needed to simulate a halt are all set to the empty string ϵ. Since the state leading to this halting state is 11, the symbol being scanned equal to 1, all symbols ending in 111 produce ϵ. In this way it is guaranteed that the tag system will halt, when the Turing machine halts. Let us now turn to the instruction table of a universal tag system. At first I was not sure whether to include the table given its length, but in the end, I thought it important that the reader can really see how big this universal tag system actually is.

Table 9.2: Instruction table universal tag system.

42010 → 4601058010	42011 → 46011580114701158011
42020 → 46020580204702058020	42021 → 48021
42030 → 48030	42031 → 48031
42040 → 48040	42041 → 48041
42050 → 46050580504705058050	42051 → 48051
42060 → 48060	42061 → 48061
42070 → 48070	42071 → 48071
42080 → 48080	42081 → 46081580814708158081
42090 → 4609058090	42091 → 48091
42100 → 48100	42101 → 4610158101

Continued on next page

Table 9.2 – continued from previous page

42110 → 48110	42111 → ϵ
42120 → 4612058120	42121 → 48121
42130 → 4613058130	42131 → 46131581314713158131
42140 → 48140	42141 → 48141
42150 → 4615058150	42151 → 46151581514715158151
42160 → 4616058160	42161 → 46161581614716158161
42170 → 4617058170	42171 → 46171581714717158171
42180 → 4618058180	42181 → 46181581814718158181
42190 → 48190	42191 → 46191581914719158191
42200 → 46200582004720058200	42201 → 4620158201
42210 → 4621058210	42211 → 46211582114721158211
42220 → 48220	42221 → 46221582214722158221
42230 → 46230582304723058230	42231 → 4623158231
42240 → 4624058240	42241 → 48241
43010 → 47010580104701058010	43011 → 47011580114701158011
43020 → 47020580204702058020	43021 → 49021
43030 → 49030	43031 → 49031
43040 → 49040	43041 → 49041
43050 → 47050580504705058050	43051 → 49051
43060 → 49060	43061 → 49061
43070 → 49070	43071 → 49071
43080 → 49080	43081 → 47081580814708158081
43090 → 47090580904709058090	43091 → 49091
43100 → 49100	43101 → 47101581014710158101
43110 → 49110	43111 → ϵ
43120 → 47120581204712058120	43121 → 49121
43130 → 47130581304713058130	43131 → 47131581314713158131
43140 → 49140	43141 → 49141
43150 → 47150581504715058150	43151 → 47151581514715158151
43160 → 47160581604716058160	43161 → 47161581614716158161
43170 → 47170581704717058170	43171 → 47171581714717158171

Continued on next page

Table 9.2 – continued from previous page	
43180 → 47180581804718058180	43181 → 47181581814718158181
43190 → 49190	43191 → 47191581914719158191
43200 → 47200582004720058200	43201 → 47201582014720158201
43210 → 47210582104721058210	43211 → 47211582114721158211
43220 → 49220	43221 → 47221582214722158221
43230 → 47230582304723058230	43231 → 47231582314723158231
43240 → 47240582404724058240	43241 → 49241
44010 → 48010	44011 → 48011
44020 → 48020	44021 → 46021580214702158021
44030 → 4603058030	44031 → 4603158031
44040 → 46040580404704058040	44041 → 4604158041
44050 → 48050	44051 → 4605158051
44060 → 4606058060	44061 → 46061580614706158061
44070 → 4607058070	44071 → 4607158071
44080 → 4608058080	44081 → 48081
44090 → 48090	44091 → 46091580914709158091
44100 → 46100581004710058100	44101 → 48101
44110 → 4611058110	44111 → ϵ
44120 → 48120	44121 → 46121581214712158121
44130 → 48130	44131 → 48131
44140 → 4614058140	44141 → 46141581414714158141
44150 → 48150	44151 → 48151
44160 → 48160	44161 → 48161
44170 → 48170	44171 → 48171
44180 → 48180	44181 → 48181
44190 → 46190581904719058190	44191 → 48191
44200 → 48200	44201 → 48201
44210 → 48210	44211 → 48211
44220 → 46220582204722058220	44221 → 48221
44230 → 48230	44231 → 48231
44240 → 48240	44241 → 4624158241

Continued on next page

Table 9.2 – continued from previous page

45010 → 49010	45011 → 49011
45020 → 49020	45021 → 47021580214702158021
45030 → 47030580304703058030	45031 → 47031580314703158031
45040 → 47040580404704058040	45041 → 47041580414704158041
45050 → 49050	45051 → 47051580514705158051
45060 → 47060580604706058060	45061 → 47061580614706158061
45070 → 47070580704707058070	45071 → 47071580714707158071
45080 → 47080580804708058080	45081 → 49081
45090 → 49090	45091 → 47091580914709158091
45100 → 47100581004710058100	45101 → 49101
45110 → 47110581104711058110	45111 → ϵ
45120 → 49120	45121 → 47121581214712158121
45130 → 49130	45131 → 49131
45140 → 47140581404714058140	45141 → 47141581414714158141
45150 → 49150	45151 → 49151
45160 → 49160	45161 → 49161
45170 → 49170	45171 → 49171
45180 → 49180	45181 → 49181
45190 → 47190581904719058190	45191 → 49191
45200 → 49200	45201 → 49201
45210 → 49210	45211 → 49211
45220 → 47220582204722058220	45221 → 49221
45230 → 49230	45231 → 49231
45240 → 49240	45241 → 47241582414724158241
46010 → 5101050010	46011 → 5101150011
46020 → 5102050020	46021 → 5502154021
46030 → 5503054030	46031 → 5503154031
46040 → 5504054040	46041 → 5504154041
46050 → 5105050050	46051 → 5505154051
46060 → 5506054060	46061 → 5506154061
46070 → 5507054070	46071 → 5507154071

Continued on next page

Table 9.2 – continued from previous page	
46080 → 5508054080	46081 → 5108150081
46090 → 5109050090	46091 → 5509154091
46100 → 5510054100	46101 → 5110150101
46110 → 5511054110	46111 → ϵ
46120 → 5112050120	46121 → 5512154121
46130 → 5113050130	46131 → 5113150131
46140 → 5514054140	46141 → 5514154141
46150 → 5115050150	46151 → 5115150151
46160 → 5116050160	46161 → 5116150161
46170 → 5117050170	46171 → 5117150171
46180 → 5118050180	46181 → 5118150181
46190 → 5519054190	46191 → 5119150191
46200 → 5120050200	46201 → 5120150201
46210 → 5121050210	46211 → 5121150211
46220 → 5522054220	46221 → 5122150221
46230 → 5123050230	46231 → 5123150231
46240 → 5124050240	46241 → 5524154241
47010 → 5301052010	47011 → 5301152011
47020 → 5302052020	47021 → 5702156021
47030 → 5703056030	47031 → 5703156031
47040 → 5704056040	47041 → 5704156041
47050 → 5305052050	47051 → 5705156051
47060 → 5706056060	47061 → 5706156061
47070 → 5707056070	47071 → 5707156071
47080 → 5708056080	47081 → 5308152081
47090 → 5309052090	47091 → 5709156091
47100 → 5710056100	47101 → 5310152101
47110 → 5711056110	47111 → ϵ
47120 → 5312052120	47121 → 5712156121
47130 → 5313052130	47131 → 5313152131
47140 → 5714056140	47141 → 5714156141

Continued on next page

Table 9.2 – continued from previous page	
47150 → 5315052150	47151 → 5315152151
47160 → 5316052160	47161 → 5316152161
47170 → 5317052170	47171 → 5317152171
47180 → 5318052180	47181 → 5318152181
47190 → 5719056190	47191 → 5319152191
47200 → 5320052200	47201 → 5320152201
47210 → 5321052210	47211 → 5321152211
47220 → 5722056220	47221 → 5322152221
47230 → 5323052230	47231 → 5323152231
47240 → 5324052240	47241 → 5724156241
48010 → 5501054010	48011 → 5501154011
48020 → 5502054020	48021 → 5102150021
48030 → 5103050030	48031 → 5103150031
48040 → 5104050040	48041 → 5104150041
48050 → 5505054050	48051 → 5105150051
48060 → 5106050060	48061 → 5106150061
48070 → 5107050070	48071 → 5107150071
48080 → 5108050080	48081 → 5508154081
48090 → 5509054090	48091 → 5109150091
48100 → 5110050100	48101 → 5510154101
48110 → 5111050110	48111 → ϵ
48120 → 5512054120	48121 → 5112150121
48130 → 5513054130	48131 → 5513154131
48140 → 5114050140	48141 → 5114150141
48150 → 5515054150	48151 → 5515154151
48160 → 5516054160	48161 → 5516154161
48170 → 5517054170	48171 → 5517154171
48180 → 5518054180	48181 → 5518154181
48190 → 5119050190	48191 → 5519154191
48200 → 5520054200	48201 → 5520154201
48210 → 5521054210	48211 → 5521154211
Continued on next page	

Table 9.2 – continued from previous page

48220 → 5122050220	48221 → 5522154221
48230 → 5523054230	48231 → 5523154231
48240 → 5524054240	48241 → 5124150241
49010 → 5701056010	49011 → 5701156011
49020 → 5702056020	49021 → 5302152021
49030 → 5303052030	49031 → 5303152031
49040 → 5304052040	49041 → 5304152041
49050 → 5705056050	49051 → 5305152051
49060 → 5306052060	49061 → 5306152061
49070 → 5307052070	49071 → 5307152071
49080 → 5308052080	49081 → 5708156081
49090 → 5709056090	49091 → 5309152091
49100 → 5310052100	49101 → 5710156101
49110 → 5311052110	49111 → ϵ
49120 → 5712056120	49121 → 5312152121
49130 → 5713056130	49131 → 5713156131
49140 → 5314052140	49141 → 5314152141
49150 → 5715056150	49151 → 5715156151
49160 → 5716056160	49161 → 5716156161
49170 → 5717056170	49171 → 5717156171
49180 → 5718056180	49181 → 5718156181
49190 → 5319052190	49191 → 5719156191
49200 → 5720056200	49201 → 5720156201
49210 → 5721056210	49211 → 5721156211
49220 → 5322052220	49221 → 5722156221
49230 → 5723056230	49231 → 5723156231
49240 → 5724056240	49241 → 5324152241
50010 → 580104205058050	50011 → 580114202058020
50020 → 580204201058010	50021 → 580214203058030
50030 → 580304204058040	50031 → 580314202058020
50040 → 580404212058120	50041 → 580414209058090

Continued on next page

Table 9.2 – continued from previous page	
50050 → 580504201058010	50051 → 580514206058060
50060 → 580604207058070	50061 → 580614207058070
50070 → 580704208058080	50071 → 580714206058060
50080 → 580804207058070	50081 → 580814202058020
50090 → 580904219058190	50091 → 580914204058040
50100 → 581004204058040	50101 → 581014213058130
50110 → 581104204058040	50111 → ϵ
50120 → 581204219058190	50121 → 581214214058140
50130 → 581304210058100	50131 → 581314224058240
50140 → 581404215058150	50141 → 581414211058110
50150 → 581504216058160	50151 → 581514217058170
50160 → 581604215058150	50161 → 581614210058100
50170 → 581704216058160	50171 → 581714221058210
50180 → 581804219058190	50181 → 581814220058200
50190 → 581904203058030	50191 → 581914218058180
50200 → 582004218058180	50201 → 582014218058180
50210 → 582104222058220	50211 → 582114223058230
50220 → 582204210058100	50221 → 582214221058210
50230 → 582304221058210	50231 → 582314221058210
50240 → 582404213058130	50241 → 582414203058030
51010 → 4205158051	51011 → 4202158021
51020 → 4201158011	51021 → 4203158031
51030 → 4204158041	51031 → 4202158021
51040 → 4212158121	51041 → 4209158091
51050 → 4201158011	51051 → 4206158061
51060 → 4207158071	51061 → 4207158071
51070 → 4208158081	51071 → 4206158061
51080 → 4207158071	51081 → 4202158021
51090 → 4219158191	51091 → 4204158041
51100 → 4204158041	51101 → 4213158131
51110 → 4204158041	51111 → ϵ

Continued on next page

Table 9.2 – continued from previous page

51120 → 4219158191	51121 → 4214158141
51130 → 4210158101	51131 → 4224158241
51140 → 4215158151	51141 → 4211158111
51150 → 4216158161	51151 → 4217158171
51160 → 4215158151	51161 → 4210158101
51170 → 4216158161	51171 → 4221158211
51180 → 4219158191	51181 → 4220158201
51190 → 4203158031	51191 → 4218158181
51200 → 4218158181	51201 → 4218158181
51210 → 4222158221	51211 → 4223158231
51220 → 4210158101	51221 → 4221158211
51230 → 4221158211	51231 → 4221158211
51240 → 4213158131	51241 → 4203158031
52010 → 4305058050	52011 → 4302058020
52020 → 4301058010	52021 → 4303058030
52030 → 4304058040	52031 → 4302058020
52040 → 4312058120	52041 → 4309058090
52050 → 4301058010	52051 → 4306058060
52060 → 4307058070	52061 → 4307058070
52070 → 4308058080	52071 → 4306058060
52080 → 4307058070	52081 → 4302058020
52090 → 4319058190	52091 → 4304058040
52100 → 4304058040	52101 → 4313058130
52110 → 4304058040	52111 → ϵ
52120 → 4319058190	52121 → 4314058140
52130 → 4310058100	52131 → 4324058240
52140 → 4315058150	52141 → 4311058110
52150 → 4316058160	52151 → 4317058170
52160 → 4315058150	52161 → 4310058100
52170 → 4316058160	52171 → 4321058210
52180 → 4319058190	52181 → 4320058200

Continued on next page

Table 9.2 – continued from previous page	
52190 → 4303058030	52191 → 4318058180
52200 → 4318058180	52201 → 4318058180
52210 → 4322058220	52211 → 4323058230
52220 → 4310058100	52221 → 4321058210
52230 → 4321058210	52231 → 4321058210
52240 → 4313058130	52241 → 4303058030
53010 → 4305158051	53011 → 4302158021
53020 → 4301158011	53021 → 4303158031
53030 → 4304158041	53031 → 4302158021
53040 → 4312158121	53041 → 4309158091
53050 → 4301158011	53051 → 4306158061
53060 → 4307158071	53061 → 4307158071
53070 → 4308158081	53071 → 4306158061
53080 → 4307158071	53081 → 4302158021
53090 → 4319158191	53091 → 4304158041
53100 → 4304158041	53101 → 4313158131
53110 → 4304158041	53111 → ϵ
53120 → 4319158191	53121 → 4314158141
53130 → 4310158101	53131 → 4324158241
53140 → 4315158151	53141 → 4311158111
53150 → 4316158161	53151 → 4317158171
53160 → 4315158151	53161 → 4310158101
53170 → 4316158161	53171 → 4321158211
53180 → 4319158191	53181 → 4320158201
53190 → 4303158031	53191 → 4318158181
53200 → 4318158181	53201 → 4318158181
53210 → 4322158221	53211 → 4323158231
53220 → 4310158101	53221 → 4321158211
53230 → 4321158211	53231 → 4321158211
53240 → 4313158131	53241 → 4303158031
54010 → 4405058050	54011 → 4402058020
Continued on next page	

Table 9.2 – continued from previous page

54020 → 4401058010	54021 → 4403058030
54030 → 4404058040	54031 → 4402058020
54040 → 4412058120	54041 → 4409058090
54050 → 4401058010	54051 → 4406058060
54060 → 4407058070	54061 → 4407058070
54070 → 4408058080	54071 → 4406058060
54080 → 4407058070	54081 → 4402058020
54090 → 4419058190	54091 → 4404058040
54100 → 4404058040	54101 → 4413058130
54110 → 4404058040	54111 → ϵ
54120 → 4419058190	54121 → 4414058140
54130 → 4410058100	54131 → 4424058240
54140 → 4415058150	54141 → 4411058110
54150 → 4416058160	54151 → 4417058170
54160 → 4415058150	54161 → 4410058100
54170 → 4416058160	54171 → 4421058210
54180 → 4419058190	54181 → 4420058200
54190 → 4403058030	54191 → 4418058180
54200 → 4418058180	54201 → 4418058180
54210 → 4422058220	54211 → 4423058230
54220 → 4410058100	54221 → 4421058210
54230 → 4421058210	54231 → 4421058210
54240 → 4413058130	54241 → 4403058030
55010 → 4405158051	55011 → 4402158021
55020 → 4401158011	55021 → 4403158031
55030 → 4404158041	55031 → 4402158021
55040 → 4412158121	55041 → 4409158091
55050 → 4401158011	55051 → 4406158061
55060 → 4407158071	55061 → 4407158071
55070 → 4408158081	55071 → 4406158061
55080 → 4407158071	55081 → 4402158021

Continued on next page

Table 9.2 – continued from previous page

55090 → 4419158191	55091 → 4404158041
55100 → 4404158041	55101 → 4413158131
55110 → 4404158041	55111 → ϵ
55120 → 4419158191	55121 → 4414158141
55130 → 4410158101	55131 → 4424158241
55140 → 4415158151	55141 → 4411158111
55150 → 4416158161	55151 → 4417158171
55160 → 4415158151	55161 → 4410158101
55170 → 4416158161	55171 → 4421158211
55180 → 4419158191	55181 → 4420158201
55190 → 4403158031	55191 → 4418158181
55200 → 4418158181	55201 → 4418158181
55210 → 4422158221	55211 → 4423158231
55220 → 4410158101	55221 → 4421158211
55230 → 4421158211	55231 → 4421158211
55240 → 4413158131	55241 → 4403158031
56010 → 4505058050	56011 → 4502058020
56020 → 4501058010	56021 → 4503058030
56030 → 4504058040	56031 → 4502058020
56040 → 4512058120	56041 → 4509058090
56050 → 4501058010	56051 → 4506058060
56060 → 4507058070	56061 → 4507058070
56070 → 4508058080	56071 → 4506058060
56080 → 4507058070	56081 → 4502058020
56090 → 4519058190	56091 → 4504058040
56100 → 4504058040	56101 → 4513058130
56110 → 4504058040	56111 → ϵ
56120 → 4519058190	56121 → 4514058140
56130 → 4510058100	56131 → 4524058240
56140 → 4515058150	56141 → 4511058110
56150 → 4516058160	56151 → 4517058170

Continued on next page

Table 9.2 – continued from previous page

56160 → 4515058150	56161 → 4510058100
56170 → 4516058160	56171 → 4521058210
56180 → 4519058190	56181 → 4520058200
56190 → 4503058030	56191 → 4518058180
56200 → 4518058180	56201 → 4518058180
56210 → 4522058220	56211 → 4523058230
56220 → 4510058100	56221 → 4521058210
56230 → 4521058210	56231 → 4521058210
56240 → 4513058130	56241 → 4503058030
57010 → 4505158051	57011 → 4502158021
57020 → 4501158011	57021 → 4503158031
57030 → 4504158041	57031 → 4502158021
57040 → 4512158121	57041 → 4509158091
57050 → 4501158011	57051 → 4506158061
57060 → 4507158071	57061 → 4507158071
57070 → 4508158081	57071 → 4506158061
57080 → 4507158071	57081 → 4502158021
57090 → 4519158191	57091 → 4504158041
57100 → 4504158041	57101 → 4513158131
57110 → 4504158041	57111 → ϵ
57120 → 4519158191	57121 → 4514158141
57130 → 4510158101	57131 → 4524158241
57140 → 4515158151	57141 → 4511158111
57150 → 4516158161	57151 → 4517158171
57160 → 4515158151	57161 → 4510158101
57170 → 4516158161	57171 → 4521158211
57180 → 4519158191	57181 → 4520158201
57190 → 4503158031	57191 → 4518158181
57200 → 4518158181	57201 → 4518158181
57210 → 4522158221	57211 → 4523158231
57220 → 4510158101	57221 → 4521158211

Continued on next page

Table 9.2 – continued from previous page	
57230 → 4521158211	57231 → 4521158211
57240 → 4513158131	57241 → 4503158031

As is hopefully clear from the table, the universal tag system based on Rogozhin's encoding is very large, to be more precise $\mu = 768$, $v = 2$. The fact that we need so many productions is a result of the method used. We do not have one tag system which is able to interpret description and input of a Turing machine in a direct way, as was the case for the first method, but we have to construct for each Turing machine separately a tag system, its number of productions depending on the number of states of the Turing machine. A further negative feature of this encoding is its exponential slow down. However, as mentioned earlier, Neary and Woods have been able to construct tag systems that simulate Turing machines in polynomial time, also applying this second method. I did not have the chance to look at their encoding in more details, and am very much embedded to Neary for having provided me with an estimate of the number of symbols the universal tag system would need. It is clear from this estimate that their encoding cannot be used to construct smaller tag systems. However, it should be noted that their goal was of course not to find small tag systems (as was neither Minsky's purpose) but efficient universal Turing machines and that it is probably possible to reduce the number of symbols.[14]
The largest problem in connection to this universal tag system is that it suffers

[14] Personal communication. To compute the number of symbols the tag system needs, Neary provided me with the following estimate. The simulation can be done by reducing the Turing machine first to a clockwise Turing machine, to a binary clockwise Turing machines, to a cyclic tag systems to a tag system with $v = 2$. To compute the number of symbols, it is very important to keep track of the increase in the number of states needed for every further reduction. Neary provided me with estimates for computing the number of states needed in the cyclic tag system to simulate a Turing machine, but we will skip the details of this calculation here. Let me merely note that if one starts with a Turing machine in n states, this number will definitely increase throughout the reductions. Then, if Q_{CT} denotes the number of states needed by the cyclic tag system – where Q_{CT} is 60 times the number of states needed by the binary clockwise Turing machine – to encode the Turing machine with Q states, then the number of symbols needed

from the disadvantage that a study of its behaviour can only be interesting if it is simulating Turing machines, and possibly tag systems. Indeed, if we feed the tag system a condition that is not structured in a particular way so that the tag system can perform the simulation, its behaviour is very predictable and it will either halt, become periodic or show unbounded growth depending on the word assigned to the letter x used in the encoding. As was already explained in Sec. 7.3.3, x is merely used as a kind of separator in the encoding and is not intended to ever become a relevant letter. Still, it is the symbol most frequently used in the words for the universal tag system. One merely has to look at how many times the number 58, encoding x, is used in its production rules to see this. If the system is started with a condition that allows x to be scanned, it will quickly become the dominant symbol and determine the behaviour of the tag system. Assigning a word with length smaller than v to x quickly leads to a halt. A similar reasoning can be applied in assigning a word of length v or a word longer than v to x, leading very quickly to periodicity rsp. unbounded growth.
To summarize, our universal tag system is exponentially slow, very large and furthermore completely uninteresting if it is not simulating other tag systems or Turing machines. It thus seems more interesting to look at tag systems directly, instead of indirectly studying them through the universal tag system described here.

9.2.3 Conclusion

To conclude this section, although (small) universal systems are basic to the theory of solvable and unsolvable decision problems, and can help to determine limits of unsolvability, it is clear that if one starts from a study of the behaviour of (classes) of systems, these universal systems have no special advantage over the systems they are able to represent, since they are, although universal, in a way too "specialized". Of course, this should not come as a surprise because universal machines are usually not intended to be implemented on a computer.

in the tag system to simulate the cyclic tag system is equal to $190(Q_{CT} + 122)$. This number of symbols is clearly very large, but it must be emphasized that this is merely an estimate.

9.3 Studying the "universe of discourse". Small universal systems, Busy Beavers and Collatz-like functions.

> At present it seems technically challenging to further reduce the size of our machines so we suspect that a radically different approach is required.
>
> Turlough Neary and Damien Woods, 2005.[15]

In the introduction of the previous section, we already pointed out that shifting attention from constructing certain systems in a given class that encode specific functions to the discourse of the systems, abstracting away from interpretation, can be a useful approach to study limits of solvability and unsolvability. We will now look at some examples from the literature where the analysis of the behaviour of systems belonging to a given class is basic to certain results in this context. Several of the examples to be discussed make more explicit the connection between the general unsolvability of a class of systems and concrete instances of systems from the class.

It is important to emphasize that this approach seems to become particularly useful in studying relatively small classes of computational systems. Indeed, the examples to be discussed involve proving or conjecturing theoretical results for very small (classes of) systems. Although *constructing* a given system that encodes a specific function is still an important approach in this context, one could say that a study based on the discourse of specific (classes of) systems becomes more and more important the smaller the systems under consideration. Especially if one wants to prove a given class – between the known limits of unsolvability and solvability – universal the usual method for constructing a universal system seems to fall short. The last example we will discuss in this section illustrates how other methods, based on the analysis of the behaviour, could be used to prove specific instances universal.

[15] [NW06d], p. 29

9.3.1 The Busy Beaver Game

As was said, in this section we will discuss an approach to unsolvability (solvability) that relies on the actual discourse of (classes of) systems. We should not forget though that, even if e.g. the developments put forward by Post, Church and Turing are part of a theoretical branch of mathematics, we already traced in each of their work a close connection between the discourse of the systems they each considered and the general theoretical results for these classes of systems. In this sense, as was already pointed out, one cannot strictly separate this approach from any other so-called more theoretical approach.

In Sec. 4.2, we showed how the computer has allowed us to study the behaviour of the objects of mathematics to an extent hardly possible without it. In this way, the computer has given us direct access to the behaviour of the systems it is itself a physical realization of. A study of the limits of the computer itself inspired Tibor Rádo 's formulation and negative solution of the Busy Beaver problem. As he states in one of his papers on the Busy Beaver Game ([Rád63], p. 76):[16]

> Let us note that our main objective is to observe the phenomenon of non-computability in its simplest form, so that we can use the insight we achieve to see better what tasks we can delegate to computers. Actually, the comments to be presented here originated with the writer's studies relating to the optimal design of automatic systems, and specifically with efforts to use computers to the limit of their capabilities for this purpose.

There are many reasons why Rádo's work is, for me, closest to the ideas I have tried to explain this far, this quote merely being one of many showing how important it was to him to search for methods to "observe the phenomenon of non-computability in its simplest form".

[16] Tibor Rádo was already rather old when he got involved with the subject. He was actually more famous for solving Plateau's problem, the problem to show the existence of a surface of minimal area with a given boundary curve. Dipping a frame in the shape of the curve into a soap solution will produce a film which takes the form of this minimal surface. It was only in 1962, that the paper containing the formulation of the Busy Beaver Game as well as the proof of its unsolvability, was published. Three years later Rádo died.

The definition of the Busy Beaver function and the proof of its non-computability were indeed intended as more simple or intuitive examples of unsolvable decision problems. As Rádo states in the abstract of his [Rád62], p. 877:

> The construction of non-computable functions used [...] is based on the principle that a finite, non-empty set of non-negative integers has a largest element. Also, this principle is used only for sets which are exceptionally well-defined by current standards. No enumeration of computable functions is used, and in this sense the diagonal process is not employed. Thus, it appears that an apparently self-evident principle, of constant use in every area of mathematics, yields non-constructive entities. The purpose of this note is to present very simple instances of non-computable functions.

As we already know, the generalized Busy Beaver game is to find, for a given class of Turing machines with n states and m symbols, the one that will produce the largest number of 1's before halting, when started with a blank input tape. Thus, the function $\Sigma(m,n)$, is indeed based on the principle of finding the largest element of a finite set of non-negative integers, a principle that occurs in every area of mathematics. In this sense, Rádo's function is "*defined in an extremely primitive sense*" ([Rád62], p. 877). Furthermore, the proof of its non-computability does not use a diagonal argument, but exactly this principle of largest elements. This resulted in a very simple if not primitive proof as Rádo calls it at a given time,[17] the non-computability of the Busy Beaver function being rooted in the fact that, in a way, it can be proven to grow faster than any computable function.

Proof of the non-computability of the BB-function (after Rádo)

Since Rádo merely considered binary machines, the number of symbols is not taken into account, and the classes are defined through the number of states. The function $\Sigma(n)$ denotes the maximum number of 1's produced by machines with n states when started with blank input tape, *before halting*. $S(n)$ denotes the maximum number of moves (shifts) made by one of the machines with n

[17] "*[...] the proof turned out to be surprisingly primitive.*" ([Rád63], p. 78)

states before it halts, when started with a blank tape.[18]
For two functions $f(x)$ and $g(x)$,

$$f(x) > -g(x)$$

is used to denote that $f(x) > g(x)$ for x greater than a certain minimal value. Rádo proved that for any computable function f:

$$\Sigma(x) > -f(x)$$

This result indeed implies that the non-computability of the function Σ results from the fact that the function grows faster than any computable function, since for any function $f(x)$, the value of $\Sigma(x)$ will be greater than the value of $f(x)$ (with x greater than a certain minimal value).
Now, consider any computable function $f(x)$. The function $F(x)$ is then defined as follows:

$$F(x) = \sum_{i=0}^{c} [f(i) + i^2] \tag{9.1}$$

and is also computable. Then:

$$F(x) \geq f(x) \tag{9.2}$$

$$F(x) \geq x^2 \tag{9.3}$$

$$F(x+1) > F(x) \tag{9.4}$$

On the basis of Turing's thesis, since $F(x)$ is computable, there exists a binary Turing machine M_F that computes F. Let us denote the number of states of M_F by C. Now, given the successor function $O(x) = x+1$ over the integers. Then we can also construct a binary Turing machine $M^{(x)}$ which, when started on a blank tape, prints $x+1$ consecutive 1's and then halts, scanning the rightmost of these 1's. For any x, such a Turing machine can be constructed with $x+1$ states. Now consider the following Turing machine $M_{FF}^{(x)}$ its operation described by the following scheme:

$$M_{FF}^{(x)} : M^{(x)} \rightarrow M_F \rightarrow M_F$$

[18] It is important to note that Rádo uses the quintuple description of Turing machines so that the machines always makes a move (left or right) for each configuration, at each step.

This machine when started on a blank tape, first prints a string of $x+1$ consecutive 1's, then, beyond a 0 to the right of this string, it prints $F(x)+1$ consecutive 1's, and finally, again beyond a 0 to the right of that string, it will print $F(F(x))+1$ 1's. After this is done, it halts. Clearly, $M_{FF}^{(x)}$ will have $1+x+2C$ states, and it is thus one of the machines to be taken into account when trying to determine $\Sigma(1+x+2C)$. Furthermore, we know its score which is equal to:

$$3+x+F(x)+F[F(x)]$$

Thus the maximum score $\Sigma(1+x+2C)$ is at least as large as the score of $\Sigma(1+x+2C)$, and we thus have the following inequality:

$$\Sigma(1+x+2C) \geq 3+x+F(x)+F[F(x)] \tag{9.5}$$

Now, since clearly $x^2 > -(1+x+2C)$ and $F(x) \geq x^2$ (from Eq. 9.3), we have:

$$F(x) > -(1+x+2C) \tag{9.6}$$

Since $F(x)$ is monotone increasing (see Eq. 9.4), the following follows from Eq. 9.6:

$$F[F(x)] > -F(1+x+2C) \tag{9.7}$$

From Eq. 9.5 and 9.7 it then follows that:

$$\Sigma(1+x+2C) > -F(1+x+2C) \tag{9.8}$$

hence (by Eq. 9.2) we get:

$$\Sigma(1+x+2C) > -f(1+x+2C) \tag{9.9}$$

in replacing $1+x+2C$ by n we finally get:

$$\Sigma(n) > -f(n) \tag{9.10}$$

Rádo has thus proven that for any computable function $f(n)$, (10) holds and $\Sigma(n)$ is thus a non-computable function. Furthermore, since with every printing operation the machine moves to the left or right, we also have:

$$S(n) \geq \Sigma(n) \tag{9.11}$$

thus from (10) and (11) we get:

$$S(n) \geq f(n) \tag{9.12}$$

for every computable function $f(n)$, thus $S(n)$ is also a non-computable function.

As is clear, the proof of the non-computability of $\Sigma(n)$ does not use a diagonal argument, but is rooted in one of the most basic principles of mathematics i.e. the idea of a largest element in a finite set of integers. In this way, the Busy beaver function can be considered as a more natural example of a non-computable function, relying as it does on an almost primitive idea used in every area of mathematics.

The proof of the non-computability of the BB-function did not stop Rádo from doing further research on the BB-function. On the contrary, he and a number of other researchers began to develop methods to determine $\Sigma(n)$ for particular values of n. So why would one try to determine specific values of a function that is proven to be non-computable in general? As far as Rádo is concerned there is a very clear answer: he began to wonder whether it is possible to find specific n for which $\Sigma(n)$ is non-computable, hoping that calculating $\Sigma(n)$ from below, starting from $n = 1$, might help to gain a better understanding of this problem.

In the following quote, Rádo explicitly states the problem of finding particular values for n that render $\Sigma(n)$ non-computable, and situates it in a more general setting ([Rád63], p. 80):

> [...] the (formal) non-computability of the function $\Sigma(n)$ [...] is directly traceable to the definition of $\Sigma(n)$ for each individual n as the largest element of a non-empty, finite set of non-negative integers [...] Now if one phrases this type of definition in terms of logical formulas, it is seen that we are faced with logical expressions of the type $\forall\exists\forall$ [...] It is well known that logical expressions of this type correspond, generally, to unsolvable decision problems. Hence, if $\Sigma(n_0)$, for example is computable for some particular n_0, then one would expect that the reason should be some *particular feature* exhibited by machines with n_0 cards, rather than a general theorem applicable to every n. This expectation is perhaps strengthened by the observation that in the extensive researches on

> solvable cases of decision problems [...] the issue is the identification of *special features* which make a particular decision problem of this type solvable. Let us alone recall the halting problem [...], not only is this problem undecidable for all Turing machines, but, [...] there exists an *individual* Turing machine whose halting problem is undecidable. Hence one would rather expect that in a similar manner there may exist particular and individual positive integer n_0 for which $\Sigma(n_0), S(n_0)$ are non-computable.

At first sight it seems rather counterintuitive to ask for specific values n for which $\Sigma(n)$ is non-computable. Indeed, one merely has to check a finite number of Turing machines, i.e. those with n states with one and the same input: a blank tape. However, given the general unsolvability of the halting problem, how will we decide for each machine individually that it will halt for this one specific input?

As we of course know, the general unsolvability of the halting problem means that there is no *general* method to decide for every Turing machine whether it will halt. In the context of finding particular values of n for which $\Sigma(n)$ is computable or non-computable this general theorem – important though as it is – will not bring us very far, since it does not say anything about particular decision methods for particular machines with particular inputs. As a consequence, since we do not have a general method, we must search for more particular methods that make it possible to decide for each machine individually from a given class, whether it will halt when started with a blank input tape. The value of $\Sigma(n)$ for specific n can be calculated iff. we have been able to prove for every machine individually from the class of machines with n states, that it will halt or not halt. This is exactly what Rádo is asking for in the quote: he asks for specific features exhibited by machines from a given class, rather than a general theorem, that makes it possible to calculate Σ for the class. The existence of certain decidability criteria as e.g. proven by Margenstern and Pavlotskaya [MP95, Pav73] is a good example of how particular features that are not implied by the general theorem can be used to determine solvable classes.

Conversely, if we want to find a particular value n for which $\Sigma(n)$ is uncomputable, we must also find a way to show that there exist classes of Turing machines with n states, for which there are specific features that make it impossible to decide for each machine individually whether it will halt when started with a blank input tape. In other words, for $\Sigma(n)$ to be non-computable for particular n, there must exist machines with n states for which one can prove that there exists no effective method to predict whether that machine will halt when started with a blank input tape. We already know that there exist specific instances of Turing machines with an unsolvable halting problem, i.e. universal machines. However, as we already mentioned in the previous section, none of the machines we have looked at, offers any challenge whatsoever with respect to the Busy Beaver Game. One of the problems here *might* be that for now the only way to prove for a given specific machine that it is unsolvable is to prove it universal. As is stated by Sutner ([Sut03], p. 366–367)

> As a matter of experience, concrete and natural r.e. problems in mathematics and computer science all appear to be either decidable or complete.

If one would find a method to prove *a given instance* unsolvable, without it being Turing complete, one would have proven that it is of an intermediate degree of unsolvability. Maybe if one would be able to find such proofs, one would be able to determine particular values n that render $\Sigma(n)$ non-computable, but of course this is a mere speculation from our side. Until now it is still an open problem to prove for specific machines (with or without particular inputs) that they are unsolvable without being universal (i.e. Turing complete). As is stated by Michel, whose research will be discussed later on in this section [Mic04]:

> It is well known that there are recursively enumerable sets that are neither m-complete, nor recursive. So, there are Turing machines that are not universal, but have an undecidable halting problem. [...] Presently, we can settle the halting problem for a given Turing machine either by producing an algorithm to prove it decidable, or by simulating a universal machine to prove it undecidable. When facing an instruction table for a Turing machine which is neither decidable, nor universal, we have no method available to prove it undecidable, and no more method to prove it not universal. Therefore, studying the undecidability line independently of the universality line would require a breakthrough in computability science.

As was said, in order to gain a better understanding of how one could find specific n for which $\Sigma(n)$ is non-computable, Rádo (in collaboration with Shen Lin) began to compute specific $\Sigma(n)$.

Computing $\Sigma(n)$

The first most obvious way to find specific values n for which $\Sigma(n)$ is non-computable, is to try to effectively calculate $\Sigma(n)$ for increasing n. With n given one must then determine for each machine with n states individually whether it will halt or not. Trying to do this by hand, very quickly becomes an unreasonable job, since the number of machines $N(n)$ with n states is equal to:

$$N(n) = [4(n+1)]^{2n}$$

For $n = 1$, it is easily determined that $\Sigma(n) = 1$. Also the case where $n = 2$ is rather easy to solve by hand, $\Sigma(n) = 4, S(n) = 6$. However, to determine $\Sigma(3)$ or $S(3)$ with $N(3) = 16777216$, the computer seems to become an indispensable instrument. Even with a computer it is very hard to determine $\Sigma(3)$ if one has not some kind of method to exclude a large number of machines. Together with Shen Lin, Rádo began to develop several methods to compute $\Sigma(3)$ and $S(3)$. These methods were programmed. The computer could then be used to exclude machines as possible winners, and to study the behaviour of those that were not excluded by the initial methods. In this way Rádo indeed implemented the idea of finding particular features of machines to decide whether they will halt on a blank input tape.

A first reduction of the number of machines with 3 states ($N(3) = 16777216$) was accomplished through a method called tree normalization, and can be described as follows. First of all, given a machine M, consider its mirror image M^*, the machine obtained by replacing each right shift in M by a left shift, and each left shift by a right shift. Clearly, M^*'s behaviour is equivalent to that of M, if M halts after t steps, having printed x 1's, M^* will halt in the same number of steps, having printed the same number of 1's. Given this symmetry, we merely have to look at those machines which go to the right when in state 1, scanning a 0 (or, equivalently, those that go to the left).

Secondly, since the machines are always started in state 1, scanning a 0, if the instruction is to go to the halting state (called state 0) then the machine will halt after one step. If the machine goes to state 1 from state 1 after having scanned a 0, it gets into a simple loop. Therefore, we only have to take into account those machines that go to a new state from state 1 after having scanned a 0, different from the halting state. Since the labels of the states are arbitrary, we can furthermore restrict attention to those machines going from state 1 to 2.

A final reduction is achieved by only taking into account those machines that print a 1 in state 1, scanning a 0. If it does not print a 1 in state 1, let us look at the further operations of the machine, until it prints its first 1. Let us then restart the machine in this state. In this new start the machine will halt iff. the original machine did, and both machines will produce the same number of 1's.[19]

Taking all these considerations together, Lin and Rádo could thus restrict their attention to machines for which the first instruction to be performed is to print a 1, go to the right and then to state 2. This approach was generalized by Machlin [Kop81] and Brady [Bra83], but the details will not be discussed here. Using this normalization, Rádo and Lin were able to reduce $N(3)$ to 82844. While this is a considerable reduction, we are still left with a large number of machines. This number was further reduced as follows. Each of the individual machines was run on a computer. Those that halted in less than 21 steps were also discarded, storing their number of steps and number of 1's printed before halting. The remaining machines were then run, printing out the behaviour of the first 50 in the list.

The behaviour of each of these 50 machines was then further examined, to determine whether they show patterns indicating that the particular machine will never stop, and has entered an infinite loop. From the analysis of these machines, they observed a certain recurrent pattern, i.e. simple loops. This pattern was programmed, and the remaining machines showing this kind of pattern could also be excluded. "*As a matter of luck*", it turned out that the recurrent pattern found disposed of all but 40 machines. These 40 "holdouts" were

[19] However, the new start will take fewer steps than the original, so this reduction might underestimate $S(n)$ by $n-1$. The solution of this problem will not be discussed here. The interested reader is referred to [MS90].

then further examined, and it turned out that all showed another kind of pattern we are already familiar with, i.e. Christmas trees, which made it possible to decide that these "holdouts" were also "never-stoppers". In this way Rádo and Lin were able to prove that $\Sigma(3) = 6$, $S(3) = 21$. Later, these methods were further refined by e.g. [Bra83], [Kop81] and [MS90], further differentiating between the different patterns, but we will not discuss the details of this research here.

More important here is the fact that in studying the Busy Beaver function, trying to determine $\Sigma(n)$, one starts from existing classes of machines, and, as a consequence, uses an approach combining a more abstract analysis of the rules with an analysis of the behaviour of the machines. Within this approach one does not care about what kind of functions are actually computed. Rather one tries to determine for each machine individually whether it will halt or not, combining human and machine work, developing both theoretical as well as more heuristic methods.

As is noted by Brady, after having used the normalization method to exclude a large number of machine to calculate $\Sigma(4)$, one has to rely on pure heuristic methods, that nonetheless lead to rigorous proofs ([Bra83], p. 647):

> The four-state case has previously been reduced to solving the blank input tape halting problem of only 5820 individual machines. In this final stage of the k = 4 case, one appears to move into a heuristic level of higher order where it is necessary to treat *each machine* as representing a *distinct theorem*. [...] The proof techniques, embodied in programs, are entirely heuristic, while the inductive proofs, once established by the computer, are completely rigorous and become the key to the proof of the new and original mathematical results: $\Sigma(4) = 13$ and $S(4) = 107$.

Once one has excluded a considerable number of machines through (a generalized method of) tree normalization, each machine has to be treated individually. The only way left for Brady to determine $\Sigma(4)$, as was the case for $\Sigma(3)$, is to analyze the behaviour of several machines individually, trying to find every possible pattern that implies infinite loops, and then program the patterns found in a way the pattern can be detected for other machines. Using these methods, Brady was able to calculate $\Sigma(4) = 13$ and $S(4) = 107$. This result was also found independently by Kopp (Machlin's girl name) using similar methods [Kop81].

On the problem of determining specific n for which $\Sigma(n)$ is uncomputable

Returning to Rádo and Lin, after having described the methods they used to solve the case $\Sigma(n)$, they make explicit one of the problems involved when trying to calculate $\Sigma(n)$ for particular n ([LR65], p. 199):

> We may stress here a certain point of interest. Even though only 40 holdouts were left, it was not clear a priori that it can be decided as to whether they are never-stoppers or not, for a given machine may exhibit such a bizarre operating record or exhibit patterns that occur only after a prohibitive number of shifts that no human being could be expected to decide that it will never stop. It is also entirely conceivable that we may have on our hands a machine which is undecidable for some logical reason.

Suppose we are working on a given class of Turing machines with n states and we are left with only 1 holdout. We have looked at the behaviour of the machine for billions of steps, still we have not detected any clear pattern that allows us to eliminate it from our list, rendering the values $\Sigma(n)$ and $S(n)$. Neither has it halted. So how will we ever calculate $\Sigma(n)$ and $S(n)$?

There are only two possible ways out of this problem. One can try to show by one or the other rigorous method that the holdout will in the end halt or get into an infinite loop. Or, one concludes one has found a particular value for n for which $\Sigma(n)$ and $S(n)$ are non-computable, because you have proven that there exists no effective method to decide that it will halt when started with a blank tape. Although it might halt some day, or get into an infinite loop, there is no other method but to wait and see. Rádo was well-aware of the problems involved for determining specific values n for which $\Sigma(n)$ is non-computable or computable, and understood how this problem is not only a problem for the machines and formalisms themselves, but also for us humans ([LR65], p. 211–

212):[20]

> The reader may surely realize that if one attempts to apply the method described above to the problem $BB-1963$, for example, then difficulties of prohibitive character are bound to arise. In the first place, the number of cases becomes astronomical, and the storage and execution for the computer programs involved will defeat any efforts to use existing computers. Even if we assume that somehow we manage to squeeze through the computer the portion of our approach involving partial recursive patterns, the number of holdouts may be expected to be enormous. Over and beyond such "physical"difficulties, there is the basic fact of noncomputability of $\Sigma(n)$, which implies that no single finite computer exists that will furnish the value of $\Sigma(n)$, for every n. In the absence (at present) of a formal concept of "noncalculability" for individual well-defined integers, it is of course not possible to state in precise form the conjecture that there exist values of n for which $\Sigma(n)$ is *not* effectively calculable.

As was said, in order to get a more formal grip on the idea of proving $\Sigma(n)$ noncomputable for specific values of n, Rádo and Lin began to explicitly calculate $\Sigma(n)$ for particular n:

> Our interest in these very special problems was motivated by the fact that at present there is no formal concept available for the "effective calculability" of individual well-defined integers like $\Sigma(4), \Sigma(5), \ldots.$ (We are indebted to Professor Kleene of the University of Wisconsin for this information) We felt therefore that the actual evaluation of $\Sigma(3), SH(3)$ may yield some clues regarding the formulation of a fruitful concept for the effective calculability (and noncalculability) of individual well-defined integers.

Instead of trying to make progress on this problem in a more theoretical way, Rádo and Lin started from the execution of the systems themselves, because the theory itself seemed to fall short. As far as we can see, this indeed seems to be the best way to at least make a start for tackling such problems.

[20] The Busy-Beaver game thus offering a clear challenge for Dr. Copeland's ideas about hypercomputability

9.3.2 Busy Beavers and Collatz-like Problems

After Rádo and Lin computed $\Sigma(3)$ and $S(3)$, several other researchers began to examine the Busy Beaver problem, trying to compute $\Sigma(n)$ for particular n. I will not go into the details of further research on the Busy Beaver Game here.[21] Let me merely note that after Machlin-Kopp and Brady settled the question for the actual values of $\Sigma(4)$ and $S(4)$, no new such values have been determined for any other n, nor for any class of Turing machines with more than two symbols, there “only” being many conjectured values or proven lower bounds. This further illustrates the intractability of the Busy Beaver problem. An overview of the current records can be found at the website of Pascal Michel, http://www.logique.jussieu.fr/ michel/bbc.html.

In two papers [Mic93], [Mic04] Pascal Michel proved that certain instances of Collatz-like functions including the famous $3n+1$-problem, are reducible to very small Turing machines. For some of these reductions, Michel did not use the more “traditional” methods for encoding functions into Turing machines, i.e. constructing a Turing machine that computes the function, but started from conjectured Busy Beaver winners, and proved the reduction through an analysis of the behaviour of the machines.

These results are important here for two reasons. First of all, as was said, for some of the reductions, Michel started from an analysis of the behaviour of particular machines. Secondly, these reductions can be used to gain new insights in the limits of solvability and unsolvability in Turing machines. Before further discussing this work by Pascal Michel, it is important to define Collatz-like functions. We will also discuss a result by John Conway. He proved that Collatz-like functions are generally unsolvable, i.e. it is possible to construct a universal Collatz-like function.

[21] A bibliography on the Busy Beaver Game can be found on the internet [Wij].

Collatz-like functions and the Polygame

Let $C : \mathbb{N} \to \mathbb{N}$ be defined by:

$$C(n) = \begin{cases} \frac{n}{2} & \text{if x is even} \\ 3n+1 & \text{if x is odd} \end{cases}$$

The $3n+1$-problem is to determine for any $n \in \mathbb{N}$, whether $C(n)$ will end in a loop 4, 2, 1 after a finite number of iterates. Most mathematicians agree that this is indeed the case for any natural number n, but the result was never proven and is still in a conjectural phase. There is clear heuristic support for the truth of this conjecture. It has been verified for any number $n \leq 224\text{X}2^{50} \approx 2 \cdot 52\text{X}10^{17}$. Furthermore, there is a heuristic probabilistic argument supporting the conjecture. Choose an odd integer n_0 at random and iterate the function T until another odd integer n_1 is found. Then in $\frac{1}{2}$ of the time, $n_1 = (3n_0)/2$, $\frac{1}{4}$ of the time $n_1 = (3n_0)/4$, ...If one presupposes that the function T is sufficiently "mixing" in that successive odd integers in the trajectory of n behave as though they were drawn at random $(\bmod 2^k)$ from the set of odd integers $(\bmod 2^k)$ for all k, then the expected growth in size between two consecutive odd integers is the multiplicative factor $\frac{3}{4}$. This argument suggests that on the average the iterates in a trajectory tend to shrink in size.[22]

Despite the existing heuristic support for the conjecture, it is still not proven. This is due to the known intractability of the $3n+1$-problem, as one e.g. sees in looking at the behaviour of the number of iterates needed before entering the loop for increasing integers. The $3n+1$-problem, also known as the Collatz problem, after its inventor Lothar Collatz,[23] is one of these problems in mathematics the solution of which seems as difficult to find as the formulation of the problem is simple. For years several researchers have studied the $3n+1$-problem and some of its generalizations without any success of finding a solution. Besides its known intractability, it is considered interesting because it is connected to several other branches of mathematics, including ergodic theory,

[22] This argument was first formulated by Crandall [Cra78]. The explanation of the argument given here is based on Lagarias' general overview of the $3n+1$-problem [Lag85].

[23] Lothar Collatz first defined this function in the thirties. There are still several other names for the problem.

Markov chains and computability theory. I will not discuss the details of the research on this problem.[24] To illustrate the difficulty of the problem, let me add the following statement by Kakutani[25]

> For about a month everybody at Yale worked on it, with no result. a similar phenomenon happened when I mentioned it at the University of Chicago. A joke was made that this problem was part of a conspiracy to slow down mathematical research in the U.S.

In [Mic04] Pascal Michel considers generalized functions of C, called Collatz-like functions. These are based on the following equivalent form of the $3n+1$-function:

$$\begin{aligned} C(2m) &= m, \\ C(2m+1) &= 3m+2. \end{aligned}$$

Given integers $d \geq 2; a_0, a_1, ..., a_{d-1}; r_0, r_1, ..., r_{d-1}; x \in \mathbb{N}$ a Collatz-like function is defined as follows:[26]

$$G(n) = \begin{cases} m_0 & \text{If } n \equiv 0 \bmod d \\ m_1 & \text{If } n \equiv 1 \bmod d \\ \vdots & \\ m_{d-1} & \text{If } n \equiv (d-1) \bmod d \end{cases}$$

where m_i is either undefined or denotes an operation of the following form:

$$\frac{a_i(n - m_i)}{d} + r_i$$

Similar generalizations were already considered by Conway. In 1972 he presented a paper *Unpredictable Iterations* at a conference on number theory [Con72]. He proved that there is no general decision procedure for Collatz-like functions,

[24] Good introductions on Collatz-like functions can be found in [Lag85, Lag95], an annotated bibliography is available on the net via Arxiv at: http://www.cecm.sfu.ca/organics/papers/lagarias/paper/html/paper.html [Lag06]. Note that [Lag95] is a more recent and seriously extended version of [Lag85].

[25] Quoted in [Lag95], from a private conversation dated 1981, Kakutani describing what happened after he circulated the problem around 1960

[26] It should be noted that Michel extends these functions to functions of pairs of integers.

i.e. a procedure that decides for any given Collatz-like function whether it will yes or no result in the value 1 after a finite number of iterates for each integer n. The formulation of these generalized Collatz-like functions for which Conway proved the result, is different from the one given above. He considered functions $g(n)$:

$$g(n) = a_i n \quad n \equiv i \bmod P$$

where $a_0, ..., a_{p-1}$ are rational numbers chosen such that $g(n)$ is always integral, i.e. the denominator of each a_i must be a divisor of the greatest common divisor of P and i.[27]

Conway proved this class of functions unsolvable, by showing that any register machine can be represented by a Collatz-like function, but did not provide an explicit construction of a universal Collatz-like function in this paper. In [Kas92], Kaščák provided a method for constructing a universal Collatz-like function. Its encoding is rather complicated: one first has to reduce a universal Turing machine to a two register machine. Reduce that machine to a specially constructed 6 register machine, which is then reduced to a 1 register machine with 66 instructions (but needing exponential encodings). Finally, it is shown how this machine can be reduced to a Collatz-like function. The resulting Collatz-like function has a modulus 396. About 15 years after the publication of his *Unpredictable Iterations* also Conway constructed a universal function, that can be reformulated in terms of Collatz-like functions, called the *Polygame*, by using his encoding methods from [Con72].

> In 1987, Conway published the paper *FRACTRAN- A Simple Universal Computing Language for Arithmetic* that explores his 1972 result, containing a specific example of a universal, and thus generally unsolvable, Collatz-like function. In a subsection called *Avoid brand X*, Conway writes ([Con87], p. 8)
>
>> Works that develop the theory of effective computation are often written by authors whose interests are more logical than computational, and so they seldom give elegant treatments of the essentially computational parts of this theory. Any effective enumeration of the

[27] It is important to point out that these functions can be rewritten in the form for Collatz-like functions given above, by setting the modulus P equal to the product of the denominators.

> computable functions is probably complicated enough to spread over a chapter, and we might read that “of course the explicit computation of the index number for any function of interest is totally impracticable.” Many of these defects stem from a bad choice of the underlying computational model. Here we take the view that it is precisely because the particular computational model has no great logical interest that it should be carefully chosen. The logical points will be all the more clear when they don’t have to be disentangled by the reader from a clumsy program written in an awkward language, and we can then “sell” the theory to a wider audience by giving simple and striking examples explicitly.

As is very clear from this quote, Conway’s Fractran, based on Collatz-like function, was developed from a very specific point of view. To Conway, it is not at all obvious that Turing machines or partial recursive functions are the dominant framework in computability theory. Indeed, as was argued in previous chapters, despite the theoretical equivalences between e.g. Turing machines and tag systems, it is very important to be aware of the many intricate differences between these different formalisms, and one must be very careful in choosing one computational model over the other, depending as it does on what one wants to do.
Let us now turn to Fractran, a programming language based on the *fraction game*. The game is played with a given list of fractions

$$f_1, f_2, ..., f_k$$

and a starting integer n. You repeatedly multiply the integer you have at any stage by the first f_i in the list for which the answer is integral. If there is no such f_i, the game stops. Using this kind of framework, Conway defines several games called the primegame, the pigame and the polygame. The primegame uses a sequence of fractions, such that if the game is started with 2, the sequence of powers of 2 that is generated by the iterative application of the primegame is the sequence of prime numbers.
The pigame is such that when started at 2^n, the next power of 2 to appear is the n-th bit in the decimal expansion of π. The polygame is a universal game, and can be rewritten in terms of Conway’s definition of Collatz-like functions. It is described by the following list of fractions:

$$\frac{583}{559}\ \frac{629}{551}\ \frac{437}{527}\ \frac{82}{517}\ \frac{615}{329}\ \frac{371}{129}\ \frac{1}{115}\ \frac{53}{68}\ \frac{43}{53}\ \frac{23}{47}\ \frac{341}{41}$$

$$\frac{41}{43}\ \frac{47}{41}\ \frac{29}{37}\ \frac{37}{31}\ \frac{299}{29}\ \frac{47}{23}\ \frac{161}{15}\ \frac{527}{19}\ \frac{159}{7}\ \frac{1}{17}\ \frac{1}{13}\ \frac{1}{3}$$

Now, define the function $f_c(n) = m$, if polygame, when started at $c2^{2^n}$ stops at

2^{2^m} and otherwise leave $f_c(n)$ undefined. Then it can be shown that every computable function appears among $f_0, f_1, f_2, \ldots$. For example, f_{37485} is equivalent to subtraction over the positive integers, where $1 \rightarrow 0$, and $n+1 \rightarrow n$. The pigame can be simulated by setting $c = 3 \cdot 5 \cdot 5^{2^{89 \cdot 101!}+2^{90 \cdot 101!}} \cdot 17^{101!-1} \cdot 23$. I will not go into the details of the proof of the universality of polygame here, nor will I give the details for finding specific c for specific computable functions. Let me merely note that one of the further advantages Conway sees in this game is the fact that the input of polygame is always one single integer ([Con87], p. 8):

> The entire configuration of a FRACTRAN machine at any instant is held as a single integer – there are no messy "tapes" or other foreign concepts to be understood by the fledging programmer

Of course, there is one clear disadvantage to Conway's fractran. When started, the input of Polygame grows exponentially with n. Furthermore, the values c needed to compute particular functions can be very large.

Although the example of Polygame can be compared to the construction of universal Turing machines as discussed in the previous section, we wanted to mention it here because it is rooted in a problem further removed from mathematical logic, and still, through the general unsolvability of Collatz-like functions, very closely connected to this domain.

Small Busy Beaver machines and Collatz-like functions.

Now that we know what a Collatz-like function is, we can finally look at their connection with Turing machines. In [Mic93], Pascal Michel proved that the $3n+1$-problem is reducible to the class TM(6, 3) of Turing machines. In [Mar00], Margenstern proved that it can be reduced to the classes TM(11,2), TM(5,3), TM(4,4), TM(3,6) and TM(2,10) and calls the set of these machines the present $3n+1$-line. This line is drawn by connecting the points of the machines known to be able to compute the $3n+1$ function, the machines being arranged in a symbol-state diagram (see fig. 9.1). The line is situated between the present solvability and universality line. Margenstern furthermore mentions that Baiocchi has improved upon this result, through reduction to the classes TM(10,2), TM(3, 5), and TM(2, 8).

None of these encodings however was obtained by an analysis of a given machine, i.e. the machines were constructed. Michel also proved that the halting problem for some other Turing machine classes depend on the decision problem of some other Collatz-like functions. He proved this for the classes TM(5,2) [Mic93], TM(2,4), TM(3,3) and TM(5,2) [Mic04] and calls this set of machines the present Collatz-like line, situated between the present $3n+1$-line and solvability line. The method used for these last reductions however is in contrast with the methods he used to reduce the $3n+1$-problem. The machines were not constructed with the aims of computing certain Collatz-like functions. On the contrary, Michel started from machines the behaviour of which had already been studied by other researchers in the context of the Busy Beaver Game. The Collatz-like functions considered, are now defined over pairs of integers, instead of over one single integer.

Given a Turing machine M over a finite alphabet Σ, and let Σ^* denote the set of finite words from alphabet Σ. If $x \in \Sigma^*$, let us define x^0 as the empty word, and for any $n >\geq 1, x^{n+1} = x^n x$. an infinite number of 0's to the the right string of 0's is denoted by 0^ω, similarly ${}^\omega 0$ denotes an infinite number of 0's to the left string of 0's. A configuration on the tape of a machine M is then denoted as ${}^\omega 0 x(Za) y 0^\omega$ where Z is the state the machine is in, $a \in \Sigma$ is the symbol scanned, $x, y \in \Sigma$. If C_1 and C_2 are two configurations of M, then $C_1 \vdash (p) C_2$ if the machine goes from C_1 to $C_2 in$p* steps. M, then $C_1 \vdash (p) END$ is used if configuration C_1 leads in p steps to the halting state H.

The following instruction table describes the Busy Beaver record holder $M_{BB(4,2)}$ in the class of Turing machines with 4 symbols and 2 states $(\Sigma(4,2) = 90, S(4,2) = 7195)$:

	0	1	2	3
A	1RB	2LA	1RA	1LA
B	3LA	1RH	2RB	2RA

Michel proved the following proposition.

Proposition 1 *: Let us denote the following configuration of $M_{BB(4,2)}$. For every $n \geq 0, C_1(n,0) =^{\omega} (A0)2^n0^{\omega}$, and $C_1(n,1) =^{\omega} (A0)2^n30^{\omega}$ Then for every $k \geq 0$,*

$$\begin{array}{lll}
C_1(3k,0) & \vdash (15k^2+7k+3) & C_1(5k+1,1) \\
C_1(3k+1,0) & \vdash (15k^2+22k+11) & END \\
C_1(3k+2,0) & \vdash (15k^2+27k+13) & C_1(5k+4,0) \\
C_1(3k,1) & \vdash (15k^2+28k+16) & END \\
C_1(3k+1,1) & \vdash (15k^2+33k+19) & C_1(5k+5,0) \\
C_1(3k+2,1) & \vdash (15k^2+43k+33) & C_1(5k+7,1)
\end{array}$$

This proposition indeed shows that the machine $M_{BB(4,2)}$ computes a Collatz-like function over two integers. Michel thus concludes that the halting problem for $M_{BB(4,2)}$ involves a study of the functions g defined by:

$$\begin{array}{lcl}
g(3k,0) & = & (5k+1,1) \\
g(3k+1,0) & = & \text{undefined} \\
g(3k+2,0) & = & (5k+4,0) \\
g(3k,1) & = & \text{undefined} \\
g(3k+1,1) & = & (5k+5,0) \\
g(3k+2,1) & = & (5k+7,1)
\end{array}$$

As is noted by Michel, the behaviour of g is still an open problem in that it is unknown whether g will always lead to an undefined value (although Michel conjectures that it does).

What makes this proposition so interesting here is that "*[t]he result is given by a tedious analysis of the behaviour*" of the machine [Mic04]. This illustrates that starting from an analysis of the behaviour of specific machines, can indeed be a very useful approach, one that clearly does not remain restricted to the heuristic domain, but leads to rigorous results. As was said before, this approach seems particularly interesting if one studies ever smaller systems, as is clear from the example given here, and might thus contribute to lowering the limits of unsolvability, or increasing the limits of solvability. As was said, Michel found similar results for some other small machines, that are not all current Busy Beaver record holders, but I will not give the details of the results.

Looking at the $3n+1$ line in Turing machines as well as the universality line,

Michel concludes that both lines could be lowered by future work. He also notes that the present Collatz-like line is already on its lowest level, any lower level implying their solvability, with the possible exception of 4-state, 2-symbols machines. Michel furthermore conjectures that these smaller machines – simulating Collatz-like functions in a way standing in clear contrast with e.g. the simulation of tag systems in Minsky's 4 by 7 machine – can be shown to be solvable. Still, given the known intractability of many Collatz-like functions this reduction to small Turing machines shows how hard it might be to prove these classes solvable. The more "traditional" reduction of the $3n+1$ problem to small machines by Baiocchi, Margenstern and Michel further strengthens this.
While Michel's work shows that *both* the more "traditional" encoding techniques, as well as an analysis of the behaviour of Turing machines, can be useful to understand the problems that might be involved in proving a given class of Turing machines solvable, depending as it does on problems known to be hard to solve, we still do not have an example of some kind of machine, considerably smaller than the known universal ones, that is proven to be universal by using methods that clearly differ from the ones discussed in the previous section. Such examples have been established rather recently by Matthew Cook [Coo04], proving the existence of small "universal" machines in the following classes (5, 2), (4,3), (3,4) and (2,7). The proofs are based on a similar result, already mentioned, in the domain of cellular automata, i.e. the proof of the "universality" of rule 110.

9.3.3 On the "universality" of cellular automaton rule 110.

In [Wol02] Wolfram gives the description of the proof of the universality of a 1-dimensional, 2-state cellular automaton (CA) with radius 1, known as rule 110, which is due to Matthew Cook [Coo04]. In Sec. 9.1 we already pointed out that Wolfram has identified four general classes of behaviour for cellular automata, the class 4 automata considered as the most "exciting" ones, containing the so-called "complex" automata which are considered to be "universal". Rule 110 is in class 4, and the proof of its "universality" is used by Wolfram to support his more philosophical so-called Principle of Computational Equivalence, of which one of the implications is that something is either universal or solvable, where this "something" is not restricted to the abstract world, but applies equally well to the physical world.[28] Given the "universality" and the formal simplicity of rule 110, Wolfram goes on to conclude that there is a very low threshold in our world for something to attain universality and in this sense to be of a highest degree of "complexity", i.e., nothing can exceed the observed complexity of rule 110's behaviour.

In turning our attention to the proof of the "universality" of rule 110, it should be noted that Cook's construction is very ingenious and deserves special attention here. As was said, using the methods of this proof, Cook was able to prove that there exist very small "universal" Turing machines. We will first give a description of the general structure of the proof, followed by a discussion of the method used. It will then be explained why the word universality has been placed between double quotes here.

General structure of the proof

Wolfram classified rule 110 as class 4 CA. This class contains CA that show behaviour lying at the edge between very simple repetitive behaviour and random behaviour, automata which are neither too repetitive nor too random and are for this reason considered "complex".[29] What one observes in looking at the evolution of CA 110, is a complex interaction between several structures, on the background of a simple repetitive pattern.

The proof of the universality of rule 110 is rooted in its ability to simulate anything calculable by *cyclic tag systems*. This variant on tag systems was developed by Matthew Cook and shown to be universal through reduction of tag systems to cyclic tag systems. A cyclic tag system is defined over a 2-symbol alphabet. Furthermore one has a finite list of n words $w_1, w_1, ..., w_n$, which have to be tagged sequentially, starting from the first word. If the last word in the list has been tagged, one starts again with the first word. Now given an initial string

[28] Note that Wolfram's principle thus excludes the possibility of natural examples of intermediate degrees, as is pointed out by Sutner [Sut05].

[29] The terminology used is of course that used by Wolfram. Thus, if we use the word random here, we mean random as interpreted by Wolfram, i.e., statistical randomness.

A. If its left-most symbol is 1, w_0 is tagged at the end of the string, and the first symbol is erased, if the first symbol is 0, the empty word is tagged at the end of *A*, and the first symbol is erased. In general, given a string produced at iteration step x, if its first symbol is 1, tag the word $w_{x \bmod n}$ at the end of the string and erase the first symbol. If the leftmost symbol is 0, the empty word is tagged and the first symbol is erased.

To show the universality of rule 110, we need an *infinite number of certain localized patterns* embedded in an *infinitely repeating background*. The background pattern has length 14, and repeats itself every 7 iteration steps and is equal to: 00010011011111. There are three localized patterns which are basic to the proof.

The first structure $S1$ shifts two cells to the right, and repeats itself every three iterations. It is the sequence 0001110111. The second structure $S2$ shifts 8 cells to the left and also repeats itself after 3 iterations. The last structure $S3$ is stationary, it never moves to the left or right, it repeats itself every 6 iteration steps. The representation of the cyclic tag system has three main components. First of all, there is the data string, representing the strings produced in the cyclic tag systems, which always remains stationary, i.e. it repeats itself every n steps, but does not move to the left or right. Secondly, we need an infinite sequence of production rules. I.e. in rule 110 it is necessary that the encoding of the sequence of words to be used is repeated infinitely often to the right of the data string. These rules pass through the data string, coming from the right. Finally, there is a infinitely repeated sequence of clock pulses which start from the left and move to the right. They are used to make sure that the right words are tagged at the end of the data string. The spacing between these three components is basic for the simulation to work. The encoding of the initial condition is very complicated and must be such that the different localized structures can interact with each other at the perfect time.

The data string processed by the cyclic tag system is encoded through the repetition of $S3$ for a certain number of times equal to the length of the data string. The letters of the string, 0 and 1 are differentiated from each other by varying the horizontal space between two consecutive such $S3$'s. Furthermore, it should be noted that the encoding of the data string is reversed, i.e. the leftmost symbol in the data string processed by the cyclic tag system is the rightmost structure $S3$ in rule 110, and vice versa. Now in simulating one iteration step of the cyclic tag system, rule 110 must be able to always destroy the encoding of the first symbol of the data string (the rightmost structure $S3$), and to tag the right word if the first symbol encoded is 1, and the empty word if the first symbol is 0. Now, as was said, a structure $S2$ always moves to the right, repeating itself. This structure is used in the encoding of the sequence of words defining the cyclic tag system. Each word w_i is represented by the repetition of $S2$ for a certain number of times equal to the length of w_i, where 0 and 1 are again differentiated by varying the spaces between the $S2$. Furthermore, every word is separated by the so-called rule separator, a localized structure which moves to the left at the same speed as $S2$.

Now, the encoding of the set of words of the cyclic tag system must be repeated infinitely often to the right of the data string, constantly shifting further and further to the left. Then, when a rule separator, coming from the right, collides with the encoding of the first symbol in the data string, i.e. its leftmost $S3$, this symbol will be destroyed. Depending on the spacings between the leftmost $S3$ and to the left of it, the rule separator in colliding with this spacing, is transformed into one of two new structures. If the spacing between these two rightmost $S3$ is such that the rightmost symbol is identified as 0, the rule separator is changed into a structure that blocks the incoming word from the right. This transformed rule separator is destroyed when the next rule separator enters from the right.

If the spacing between the rightmost and next symbol was that used for the encoding of 1, the rule separator is transformed into a new structure that does not block the incoming word. This new rule separator will also have been destroyed when the new separator enters from the right, however not before having allowed a series of structures to pass through towards the left, i.e. the encoding of the word to be tagged. These structures will then be tagged at the leftmost symbol of the data string. This is done by using the above mentioned clock pulses, which are encoded though $S1$. This $S1$ is repeated infinitely many often to the left of the data string, always moving to the right. If the encoding of our word, allowed to pass through by the transformed rule separator, reaches

the left end of the data string, these clock pulses are used to tag the right bits in the proper encoding. If the encoding of a bit from this word is 1, the clock pule meeting it will transform this *S*2 and its spacing in a stationary *S*3 with the proper spacing needed for the encoding of a 1 in the data string. A similar procedure is performed when the incoming symbol is 0.
Using this kind of transformations, rules coming from the right, colliding with the data string, the clock pulses coming from the left, to arrange the right tagging operation, it can be shown that any cyclic tag system can be simulated by rule 110.

Discussion of the proof

As is hopefully clear, the methods used by Cook differ significantly from those discussed in the previous section. Although one needs perfectly encoded initial conditions, the respective structures being spaced such that their collisions are synchronized, the method used is clearly based on a “tedious analyses of the behaviour” of rule 110. Indeed, in order to achieve what Cook has done one must be very familiar with the things rule 110 is capable of. The proof itself does not start from the instruction table of rule 110, it starts from the structures which can be produced by rule 110 and how these structures evolve depending on what kind of other structures they ‘collide’ with within the CA. Evidently, Cook did not find this proof by starting from cyclic tag systems to construct a universal CA which happened to be rule 110. Rather it was on the basis of a long sequence of observations, that the conjecture of the universality of rule 110 was made. Then Cook started to investigate rule 110, probably having tested different kind of structures and their collisions to finally end up with this very complicated proof. The significance of the computer in this context can hardly be underestimated. We have thus new support for the idea that focussing on the behaviour of certain computational systems rather than on the specific functions they compute, can lead to important results in the context of limits of unsolvability. Again it is clear that this method is used when a very small system is involved the behaviour of which indicates that it might be a very powerful computational tool, while its rules can give us hardly any information here.

Despite the fact that this proof by Cook further supports some of the ideas put forward here, there is one problem with the proof: it doesn't satisfy Davis's definition of universality. Let us recall that the reason for Davis to write his note was to give a definition for universal devices, for which the encoding can always be done by a non-universal machine. The proof that Davis's definition of universality, in terms of what is now often called Turing completeness, indeed satisfies this condition, is based on the fact that the encoding can always be done by recursive functions, which in their turn can be proven to be strongly computable. I.e. they can be computed by a Turing machine that always halts.[30] Now, in order to generate the proper input for rule 110 to simulate a given cyclic tag system, we cannot use a Turing machine that strongly computes the encoding function, since the input for rule 110 is infinite and can thus not be the output of a machine that halts in a finite number of steps. The "universality" of rule 110 is thus not covered by Davis's definition of universality. The following reasoning clarifies the significance of the problem. Before starting rule 110 simulating a cyclic tag system, the initial condition must be encoded by say a Turing machine. However, rule 110 can never be started because it has to wait until the production rules and the clock pulse have been encoded an infinite number of times. The only other possibility left is that neither clock pulses nor production rules are repeated infinitely many often. The simulation can then only succeed if one knows in advance what will happen with the cyclic tag system, i.e. if one knows that it will halt, or become periodic. However, in order to know this one must first solve the halting problem for cyclic tag systems. And even if we would be able to solve the halting problem, we would still need infinite repetition of production rules and clock pulses when cyclic tag systems are involved that never halt nor become periodic.

Moreover, a major problem for me personally with this kind of encoding is that the class of computational systems the universality of rule 110 is based on, cyclic tag systems and indirectly tag systems, seem not to be capable of this kind of encoding, since these systems do not allow for this kind of infinite en-

[30] It is maybe interesting to note that Post's definition of 1-giveness also required that a given problem should be encoded by a finite procedure that always terminates (See Sec. 3.1.3, p. 135).

codings. Of course one could say that, virtually, there are infinitely many 0's to the right of a tag string, but this still doesn't help us any further here. If we would have an encoding of the production rules of one or the other system by repeating them infinitely often, we are confronted with two problems. First of all, tagging something at the *end* of an *infinite string*, can at least be called problematical. Secondly, even of we would allow this, it would take infinite time before the word tagged at the end of our infinite string would be processed and the tagging operation would thus have no effect at all on the future behaviour of the process.

Tag systems and cyclic tag systems are not the only class of computational systems for which this kind of encoding is impossible. For example, also register machines do not allow such encoding, since one would have to use an infinite number of registers, or an infinite number representing the infinite sequence of production rules. In general, one can conclude that the special kind of encoding used by Cook can only be applied to computational systems which operate on infinite tapes in some or the other way, like Turing machines or cellular automata. Tag systems, on the other hand, seem to have a kind of "built-in" protection against such problematic intrusions of the infinite.

To conclude, while the proof given by Cook of the universality of rule 110 is very interesting from the perspective of this section, based as it is on the behaviour of this system, there are some clear problems with the encoding used, involving the infinite repetition of the production rules. Still Cook's proof is worth more research and is rather ingenious. I for myself am not completely sceptic about the proof, because, without taking into account Davis's (realistic) definition of universality and the remarks about this encoding in relation to tag systems, one cannot neglect the fact that, intuitively, rule 110 is indeed able to simulate the computations of any cyclic tag system, be it under conditions which are not that intuitive. The system needs to be remembered time and time again about what rules it is actually simulating, and is not able to self-reproduce the rules in a finite way.

9.3.4 Conclusion

This chapter started from the idea that small universal systems might enhance our understanding of the connection between the general unsolvability of a class of systems, and the actual discourse of each of these systems. Although small universal systems are very important in this context – as particular examples of systems with an unsolvable decision problem they help us to draw limits of unsolvability – we concluded from Sec. 9.2 that a study of the behaviour of these machines offers us no extra advantage over a study of the behaviour of other systems not known to be universal or solvable. In this section we showed that research closely connected to limits of solvability and unsolvability often involves some kind of study of the behaviour of particular systems and shows itself very useful to prove certain theoretical results in this context.

As is clear from the examples discussed, research based on an analysis of the behaviour of certain (classes of) systems, shows itself particularly important when smaller systems are concerned. Indeed, once one starts to work with smaller systems, starting from existing instances rather than deliberate constructions, an analysis of behaviour is often the only way out to get the desired result or to at least build up an intuition of the problems involved. It is also clear that the computer is often an indispensable instrument in this context. It is the computer that has made the behaviour of these several computational systems it is the physical realization of, accessible to us humans to an extent impossible before. The fact that one needs a computer to study the behaviour of these systems is not merely connected to our slowness as far as computations are concerned, but is also rooted in the fact that the systems whose behaviour is studied, are by no means trivial to predict: if we would know what would happen with a given system when started with certain initial conditions, we wouldn't need our computer to study its behaviour because we would know it in advance. In other words, the computer becomes an indispensable part of the process of finding mathematical results in the research context considered here, when systems are concerned for which it is hard to make predictions on the basis of an analysis of their description. As is clear, these comments are very close to some of the remarks made by Lehmer and von Neumann in this context (See Sec. 4.2).

This last feature explicates the link between the general theoretical result of a class of systems being unsolvable, and their actual discourse. For example, the problem posed by Rádo to find particular values n that render $\Sigma(n)$ computable or uncomputable, is more related to problems connected to the execution of particular machines. In fact, as we showed, it is the analysis of the behaviour of the machines, made accessible through the computer, that makes it possible to compute particular values of $\Sigma(n)$. Of course, the fact that we must shift attention to the behaviour of the machines, is a consequence of the general unsolvability of the halting problem for Turing machines, but this is exactly where the connection between discourse and theoretical results lies.

It is now time to relate this discussion to Post's form of "Tag".

9.4 On the limits of solvability and unsolvability in tag systems.

As was shown in the previous sections, both theoretical constructions encoding certain functions, as well as a study of the behaviour of specific (classes of) systems, are two important approaches in the context of studying limits of solvability and unsolvability. Often it is the combination of the more theoretical encodings, such as the reduction of the $3n+1$-problem to relatively small Turing machines, and other more theoretical results, with a study of the behaviour of certain classes of systems that helps to study these limits. To our mind, you cannot do without the theoretical encodings and results nor without the analyses of behaviour to make further progress in this domain.

Tag systems, lying at the basis of many known small universal systems, will be the main subject of this, rather lengthy, section. In Sec. 2.2.5 it was shown that tag systems played an important role in Post's work. They were not only significant with respect to the formulation of his important systems in normal form, but furthermore first led him to the idea that there might exist unsolvable decision problems in logic and mathematics. As was argued, Post "experimented" with these systems. He tried out several cases in order to deduce more general properties, i.e. a study of the behaviour of these systems has been basic for his theoretical conclusions. As Post did not have a computer, he had to use paper and pen to study his tag systems, a truly exhausting task.

As was pointed out by Post, he considered the class of tag systems with $\mu = 2, v > 2$ intractable – possibly rooted in his experiences with **T1** – while he characterized the class with $\mu > 2, v = 2$ as being of "bewildering complexity". In the previous chapter we already saw that Post had every reason to consider the class $\mu = 2, v > 2$ intractable, the fact that **T1** is still not known to be solvable nor unsolvable only further strengthens this observation. In this section we will show that Post was also completely correct in calling the class $\mu > 2, v = 2$ complex, i.e. we will prove that the solvability of the class $\mu = 3, v = 2$ depends on a solution of the $3n+1$-problem (Sec. 9.4.1). Given the simplicity of the encoding used in the proof, we will have an argument supporting Post's observation concerning the connection between tag systems and number theory.

In a next section (Sec. 9.4.2) we will prove that the class of tag systems with $\mu = v = 2$ is solvable. After a summary of the known limits of solvability and unsolvability in tag systems and Turing machines, we will tackle the question of whether there exist universal tag systems in the class $\mu = 2, v > 2$. We will describe an abstract method that might be used to encode a universal two-symbolic tag system, and the reader is warned here in advance that this method is rather intricate and speculative, if not a bit obscure. Combining these more theoretical results on the limits of solvability and unsolvability in tag systems with some of the experimental results from the previous chapter, we will connect our results on tag systems with the previous discussions and argue that their limits of unsolvability are very low relative to Turing machines.

9.4.1 Tag systems and Collatz-like problems

> The tag problem has a tantalizing resemblance to another famous little stinker, generally known as the $3X+1$ problem.[...] Like the tag problem,, the $3X+1$ problem is unsolved. Is there any close connection between them?

Brian Hayes, 1986.[31]

In Sec. 9.3.2 we considered Collatz-like functions, the definition of which is based on a reformulation of the $3n+1$-problem, with $C(2m) = m, C(2m+1) = 3m+2$. In this section we will show that C can be reduced to a tag system T_C with $\mu = 3, v = 2$. The method will be generalized to any Collatz-like function, thus proving that any Collatz-like function can be reduced to a tag system. Since Conway proved that Collatz-like functions are generally unsolvable we thus obtain an alternative proof of the unsolvability of tag systems.[32]

Reduction of the $3n+1$-function to a tag system

In this section we will prove the following theorem:

Theorem 9.4.1 *The function $C(n)$ is reducible to a tag system T_C with $\mu = 3$, $v = 2$.*

Let A^i denote a string A repeated i times, $A \overset{\circ}{\rightarrow} B$ is the string B produced from A, after all the letters from A have been erased. Let the alphabet be $\Sigma = \{\alpha, c, y\}$ and $n \in \mathbb{N}$. Then, each iteration of $C(n)$ corresponds to the production of a string $\alpha^{C(n)}$ from a string α^n in T_C. The production rules are:

$$\begin{array}{ccc} \alpha & \rightarrow & cy \\ c & \rightarrow & \alpha \\ y & \rightarrow & \alpha\alpha\alpha \end{array}$$

Now, if n is of the form $2m$, T_C produces $\alpha^{\frac{n}{2}}$ from α^n:

$$\begin{array}{lcl} \alpha^n & \overset{\circ}{\rightarrow} & (cy)^{\frac{n}{2}} \\ (cy)^{\frac{n}{2}} & \overset{\circ}{\rightarrow} & \alpha^{\frac{n}{2}} \end{array}$$

[31] [Hay86], p. 27

[32] The results to be described here, are based on a paper *Tag systems and Collatz-like functions* [Mol07] I submitted to Theoretical Computer Science and is now under revision. They were also presented at CIE06, as part of a general talk on the usefulness of small universal systems [Mol06c].

If n is of the form $2m+1$, T_C produces $\alpha^{3(\frac{n-1}{2})+2}$ ($= \alpha^{3m+2}$) from α^n:

$$\begin{array}{lcl} \alpha^n & \overset{\circ}{\rightarrow} & y(cy)^{\frac{n-1}{2}} \\ y(cy)^{\frac{n-1}{2}} & \overset{\circ}{\rightarrow} & \alpha^{3(\frac{n-1}{2})+2} \end{array}$$

This encoding allows for efficient simulation of $C(n)$ for any n. If n is even, C_T needs n iterations, with n uneven, $n+1$, to simulate one iteration of $C(n)$. The reason for the simplicity of this encoding is that $C(n)$ relies on modulo operations, while tag systems themselves can be regarded as some kind of modulo systems. Indeed, the encoding is based on this one feature of tag systems. Consider a string A of length $|A|$, and let $A \overset{\circ}{\rightarrow} B$. Clearly, the length of B depends on $|A|$ mod v, in that the "original" length of B (the addition of the lengths of the words produced from A) will be decreased by the additive complement of $|A|$ mod v.[33]In this respect, $|A|$ mod v determines what sequence of letters in B will and will not be scanned by the tag system. This feature is not only basic to our encoding, but is also the main ingredient in Minsky's and Cocke's proof of the universality of tag systems with $v = 2$ (See Sec. 6.1.1). To return to our encoding of C in T_C, if $|\alpha^n|$ is even, $|\alpha^n| \overset{\circ}{\rightarrow} (cy)^{\frac{n}{2}}$, with $|(cy)^{\frac{n}{2}}|$ mod $v = 0$, guaranteeing that only the letter c will be scanned in B. Similarly, since $|(cy)^{\frac{n}{2}}|$ is even, no letter from $\alpha^{\frac{n}{2}}$ will have been erased after all the letters of $|(cy)^{\frac{n}{2}}|$ have been erased. In case $|\alpha^n|$ is uneven, $|\alpha^n| \overset{\circ}{\rightarrow} B$, with $|B|$ mod $v = 1$, the first leading c being erased when the last α in α^n has been scanned. As a result, the tag system will scan the sequence of letters y. Although, taking together all the y's results in $\alpha^{3(\frac{n-1}{2})+3}$, the oddness of $y(cy)^{\frac{n-1}{2}}$ guarantees that the leading α will be erased after the last y has been scanned, thus leading to the desired result.

It should be noted here that T_C satisfies the minimal condition discussed by Maslov (Sec. 6.1.1). Indeed, $l_{min} = v-1$ and $l_{max} = v+1$.

Furthermore, the problem to decide for any n, whether $C(n)$ will ever lead to 1 after a finite number of iterations, reduces to the question of whether T_C will ever produce α. In other words, the $3n+1$-problem can be reduced to a reachability problem for T_C.

[33] For the definition of additive complement, see Sec. 6.3.2.

Generalization of the method to arbitrary Collatz-like functions

By generalizing and slightly changing the encoding from the previous section, we were able to prove the following theorem:

Theorem 9.4.2 *Given an arbitrary Collatz-like function $G(n)$, with modulus d. Then, there is always a tag system T_G with $v = d, \mu \leq 2d+3, \Sigma = \{h, \alpha, \alpha_0, \beta_0, \beta_1, ..., \beta_{d-1}, b_0, b_1, ..., b_{d-1}\}$ that simulates $G(n)$ for any n.*

Note that μ and v are completely determined by the modulus. The symbol h functions as a kind of halting symbol, used for those cases when $G(n)$, $n = dm + i$, $0 \leq i < d$, is undefined for i. It is also important to note that the encoding of the present section needs the extra symbols $\alpha_0, \beta_0, \beta_1, ..., \beta_{d-1}$.
Each iteration of G over a number n corresponds to the production of a string $\alpha_0\alpha^{G(n)}$ from a string $\alpha_0\alpha^n$. The production rules for α_0, α are:

$$\begin{array}{lcl} \alpha_0 & \rightarrow & \beta_{d-1}\beta_{d-2}...\beta_0 \\ \alpha & \rightarrow & b_{d-1}b_{d-2}...b_0 \end{array}$$

If $G(n)$ is defined, with $n = dm + i$, $0 \leq i < d$, the production rules for β_i and b_i are :

$$\begin{array}{lcl} \beta_i & \rightarrow & (\alpha)^j\alpha_0(\alpha)^{r_i} \\ b_i & \rightarrow & (\alpha)^{a_i} \end{array}$$

where j is the additive complement of $(i+1)$ relative to d [i.e. : $-(i+1)$ mod d evaluated to its least positive remainder], with $i = n$ mod d.
If $G(n)$ is undefined, $n = dm + i$, $0 \leq i < d$, the production rules for β_i and b_i are:

$$\begin{array}{lcl} \beta_i & \rightarrow & h \\ b_i & \rightarrow & h \end{array}$$

The production rule for h is:

$$h \rightarrow \epsilon$$

Now, applying the production rules of T_G to a given string $\alpha_0\alpha^n$, in case $G(n)$ is defined, we get:

$$\alpha_0\alpha^n \overset{\circ}{\rightarrow} \beta_i\beta_{i-1}...\beta_0(b_{d-1}b_{d-2}...b_0)^{\frac{n-i}{d}} \tag{9.13}$$

Note, that we again use the property, mentioned in Sec. 9.4.1, that the length of a string B produced from a string A, through $\overset{\circ}{\to}$, is completely determined through $|A| \bmod v$, i.e. if the additive complement c of $|A| \bmod v > 0$, then the first c letters of the first word(s) produced from A will be erased, when the last letter of A has been scanned. Note that because the number of letters erased is equal to c, the order of the indices of the letters in the words produced from $\alpha_0, \alpha, \beta_i, b_i, 0 \leq i < d$ is reversed, so that we are able to keep track of the remainder. Furthermore, by adding the extra symbol α_0, the rules assure that $b_{d-1}b_{d-2}...b_0$ will be repeated $m = \frac{n-i}{d}$ times.
After the application of one iteration on the string produced in (9.13), T_G produces:

$$b_i b_{i-1}...b_0(b_{d-1}b_{d-2}...b_0)^{\frac{x-i}{d}-1}(\alpha)^j \alpha_0(\alpha)^{r_i} \tag{9.14}$$

From (9.14), T_G produces

$$\underbrace{b_i b_{i-1}...b_0(\alpha)^j}_{d} \alpha_0(\alpha)^{a_i(\frac{n-i}{d}-1)+r_i} \tag{9.15}$$

after (n-i)/d - 1 iterations. As is clear, the symbol β_i produced through α_0 is used to assure the tag system will start scanning α_0 after one iteration of G has been completed, through the addition of j times α, since

$$i + 1 + j = d.$$

Furthermore, β_i is used to add r_i if $G(n)$ is defined and $r_i > 0$. The letter b_i is used to perform the multiplication of m with a_i, since b_i is repeated $m = (n-i)/d$ times.
From (9.15) T_G finally produces:

$$\alpha_0(\alpha)^{a_i\frac{n-i}{d}+r_i} \tag{9.16}$$

after one more iteration.
If we apply the production rules to a string $\alpha_0\alpha^n$, in the case $G(n)$ is undefined, the production given in (9.13) remains unchanged. Then

$$\beta_i\beta_{i-1}...\beta_0(b_{d-1}b_{d-2}...b_0)^{\frac{n-i}{d}} \overset{\circ}{\to} h^{\frac{n-1}{d}+1} \tag{9.17}$$

From (9.17) we finally get:

$$h^{\frac{n-1}{d}+1} \overset{\circ}{\rightarrow} \epsilon \qquad (9.18)$$

leading the tag system to a halt.

The encoding of Collatz-like functions into tag systems is thus very straightforward, the input n for G being directly encoded as a string of length $n+1$. As was the case for the reduction of the $3n+1$-problem, the simulation of Collatz-like functions is efficient, where one iteration of $G(n)$ maximally takes $2(\lfloor n/d \rfloor + 1)$ iterations in T_G.

Given the fact that any Turing machine can be reduced to a Collatz-like function, the reduction of the present section serves as another proof of the existence of a universal tag system. Furthermore, the unsolvable problem to determine whether a Collatz-like function G, given an integer n, will ever produce the number 1 after a finite number of steps, reduces to the reachability problem to determine for any tag system T_G whether it will ever produce the string $\alpha_0\alpha$.

In comparing the encoding of the present section with that from Sec. 9.4.1, it is clear that the encoding of the present section leads to the simulation of the $3n+1$-problem in a larger tag system, with $\mu = 6$. This is due to the use of the symbol α_0. One might thus wonder whether there is a condition under which a tag system T_G, encoding a function $G(n)$ using α_0, can be reduced to a smaller tag system T'_G, without α_0.[34] The following theorem gives such a condition as well as the production rules of T'_G, which are based on the encoding of the $3n+1$-problem from Sec. 9.4.1 in T_C.

Theorem 9.4.3 *Given a Collatz-like function $G(n)$ with modulus d, where for each n, $G(n)$ is either undefined or equal to $\frac{a_i(n-i)}{d}+r_i$, $i = 0,1,...,d-1$. Then $G(n)$ can always be reduced to a tag system T'_G with $v = d, \mu \leq 2+d, \Sigma = \{h, \alpha, b_0, b_1, ..., b_{d-1}\}$ iff. for every i defined, $\overline{i} < a_i$, if $i > 0$, $r_i = a_i - \overline{i}$, if $i = 0, r_i = 0$, where $\overline{i}$ is the additive complement of i. For each i defined, the production rules of T'_G are: $\alpha \rightarrow b_0 b_{d-1}..b_2 b_1; b_i \rightarrow \alpha^{a_i}$. For i undefined, the production rules are $b_i \rightarrow h$; $h \rightarrow \epsilon$*

The details of the proof are left to the reader.

[34] I am indebted to Pascal Michel for pointing out this problem to me.

Discussion of the result

The $3n+1$-problem is known as a highly intractable problem, and, as we already argued, proving the reducibility of the $3n+1$-problem to classes of machines considerably smaller than the known universal ones, can serve as an indication of the difficulties that might be involved in proving machines from this class solvable. Given the intractability of the $3n+1$-problem, Margenstern [Mar00] conjectured that all classes of Turing machines to which the $3n+1$-problem can be reduced, i.e. all points on the $3n+1$-line, contain a machine with an unsolvable halting problem or an unsolvable reachability problem or an unsolvable modified reachability problem (a conjecture that assumes of course nothing about the status of the $3n+1$-problem). In drawing from this research, the reduction of the $3n+1$-problem to a tag systems with $\mu = 3$, $v = 2$, strongly suggests that proving the solvability of this class of tag systems will be very hard. It is also important to note that the tag system T_C is considerably smaller than the size of the known Turing machines to which the $3n+1$-problem can be reduced. Furthermore, whereas the class of tag systems TS(3, 2), where TS(μ, v) denotes the class with μ symbols (and production rules) and a shift number v (cfr. Sec. 6.1.1), contains T_C, the class of Turing machines TM(3, 2) and TM(2,3) is known to be solvable. This result suggests that the limits of unsolvability might be significantly lower in tag systems as compared to those for Turing machines.

The simplicity and efficiency of the encoding of the $3n+1$-problem to T_C, and the general encoding scheme for Collatz-like functions to tag systems serves as an indication that tag systems might be used as a kind of bridge between problems in number theory and problems in computer science or computability theory. Indeed, as was shown, this encoding is so simple because tag systems themselves are a kind of remainder systems. We have not been able to further explore this connection, but it seems possible that in further investigating this connection, one might be able to find new methods to prove certain properties for tag systems, drawing from remainder arithmetic. For now however, this connection remains rather intuitive and is in need of further investigation. Maybe a search in the Post archive might lead to something. Indeed, as is clear from two of the quotes from Sec. 2.2.5, Post must have made such a connection. To be more exact, there are three quotes in which Post mentions this connection. In the third quote, Post states ([Pos65], p. 386):

> Now we mentioned [...] how an extended attempt to solve the simplified form of this finiteness problem "Tag" led to ever increasing difficulties, with all the complexities of number theory in the offing.

In one of the quotes from Sec. 2.2.5, Post also indicates that he considered the regularity of always removing v elements, i.e. the shift number, as responsible for "*the intrusion of number theory in the development* [...]".[35]
We are not sure whether there are any notes left on Post's more detailed research on tag systems, but are planning to visit the archive hoping we might find such notes. The fact that the connection between tag systems and remainder arithmetic seems so obvious, serves as a further argument for the significance of doing more research on tag systems, although we cannot predict the outcome of such research.

9.4.2 Solvability of the class $\mu = v = 2$.

> **PROBLEM**. Choose any two-symbol, two-state machine and show that it is *not* universal. Hint: Show that its halting problem is decidable by describing a procedure that decides whether or not it will stop on any given tape. D. G. Bobrow and the author did this for all (2,2) machines [1961, unpublished] by a tedious reduction to thirty-odd cases (unpublishable).
>
> Marvin Minsky, 1967.[36]

During his research on tag systems, Post proved the solvability of the class of tag systems $\mu = v = 2$. Although he mentions this result in [Pos65], the proof was never published. Now, I am completely sure that when Post says that he had this proof, he really had this proof. Still, to convince the sceptical reader, it was considered important to find such a proof. Since Post understood this result as

[35] [Pos65], p. 382

[36] [Min67], p. 281

the major success of his project to prove the problem of "tag" solvable, during his Procter fellowship, it was also a challenge and an experience to search for this proof. In a footnote Post writes that "*the special case $\mu = \nu = 2$ involved considerable labor.*"[37] We are not completely sure how to interpret this quote, we have not seen Post's proof, but as will become clear through the proof, the one we found indeed involves considerable labor.

In Sec. 6.2.1 we described how Post differentiates between three classes of behaviour a tag system can converge to, i.e., a tag system can halt, it can become periodic, or it can show unbounded growth. The reachability and halting problem, the two forms of the problem of tag, can be proven solvable, if one can determine for any initial condition, for a given tag system, that it will lead to one of these three classes of behaviour after a finite number of steps. Remember that in case of unbounded growth, one should be able to prove that for any given number n the tag system will always produce a string A_i of length $l_{A_i} > n$ after a finite number of iterations i, such that no string $A_j, j > i$, will ever be produced again for which $l_{A_j} \leq n$.

In the proof following hereafter, we will show that for any tag system from the class $\mu = \nu = 2$, one can indeed determine whether it will become periodic, halt or show unbounded growth after a finite number of steps, and we will thus prove the following theorem:

Theorem 1 *For any given tag system T, if $\mu = \nu = 2$ then the halting problem and the reachability problem for T are solvable.*

First of all, it should be noted that we only have to consider those cases with $l_{\mathbf{min}} < 2, l_{\mathbf{max}} > 2$, given the theorem proven by Wang mentioned in Sec. 6.1.1. In the remainder, we assume that $l_{\mathbf{max}} = l_{w_1}, l_{\mathbf{min}} = l_{w_0}$, the symmetrical case of course being equivalent to this case.

There are three global cases to be taken into account, i.e., $w_0 = \epsilon, w_0 = 1, w_0 = 0$. Each of these cases is subdivided into several subcases, determined by the following parameters: the parity of w_1,[38] the length l_{w_1} of w_1 and the total number of 1's in w_0 and w_1 (indicated as #1). It should be noted that, contrary to classes

[37] [Pos65], p. 362

[38] The parity of a number x is the property of being even or odd.

of Turing machines TM(m, n), the three global cases to be considered contain an infinite number of tag systems. In this sense it has been basic for this proof that it is possible to determine certain threshold values for the last two of these parameters, i.e., l_{w_1} and #1. If the values of these parameters are larger than a given number the infinite class of tag systems determined by the parameters will always show unbounded growth except for a specific class of initial strings. If these values are smaller or equal to these parameters, the tag systems will always halt or become periodic, except for a determined class of initial conditions.

There is one specific method that has been basic to solve the majority of cases to be considered, i.e., the *table method* (See Sec. 7.3.4). Remember that what one basically does with this method is to look at a certain number of substrings that can be produced theoretically in a given tag system, by starting from the possible productions from the respective words $w_0, ..., w_{\mu-1}$. The table method is applied to the tag system by looking at all possible strings v that can be produced from each of the words w_i, $0 \leq i < \mu$, by concatenating the words corresponding to the letters of each of the v different sequences in each of the w_i, sequences which are determined by the shift w_i is entered with, i.e., the number of leading letters in w_i that are erased but not scanned. If one of these new strings produced is equal to one of the words w_i it is marked. If all strings produced in this way are marked or equal to ϵ it follows that the tag system will always halt or become periodic, since the length of the strings that can be produced from the respective words is bounded. If this is not the case, the same procedure is applied to all strings left unmarked and not equal to ϵ,...

As will become clear in the proof, the table method is not only useful if, for a given tag system, all the strings become marked or are equal to ϵ at a given time, but can also be used to e.g. prove that a tag system will either halt or show unbounded growth.

In should be noted that from now on, $\dot{x}$ denotes that x is uneven, similarly, a non-dotted number x denotes an even number. Furthermore l_{w_0} and l_{w_1} are abbreviated as l_0 rsp. l_1. In our outline, we will separate the three global cases, which are in their turn to be subdivided into the respective subcases.

Case 1. $w_0 = \epsilon$

Case 1.1. #1 = 0. Irrespective of the length of w_1 it is trivial to prove that tag systems from this class will always halt, since only 0's can be scanned.

Case 1.2. #1 = 1, $l_1 \equiv 0 \bmod 2$. Let $w_1 = 0^{\dot{x}_1}10^{y_1}$. The following table proves the lemma:

	w_1
S_0	HALT
S_1	w_1✓

The row headed with S_0 (shift 0) gives the string produced from a given string S (in this case w_0 or w_1) when the first letter of the string S is scanned by the tag system. Similarly, the row headed S_1 (shift 1) gives the possible productions from a given string S when its first letter is erased without being scanned.
As is clear from the table, a tag system from this class will either halt or become periodic. It will always become periodic when at least one 1 is scanned in the initial condition, such that the first letter in w_1 resulting from this 1 will not be scanned. This can be determined through the parity of the length of the initial condition. In all other cases, tag systems from this class always halt. A similar proof can be given for the case $w_1 = 0^{x_1}10^{\dot{y}_1}$.

Case 1.3. #1 = 1, $l_1 \equiv 1 \bmod 2$ The table that can be constructed for this class of tag systems, is identical to the previous table, with $w_1 = 0^{\dot{x}_1}10^{\dot{y}_1}$. Despite the table being identical, tag systems from this class can be proven to always halt. Given an arbitrary number n of 1's scanned in the initial condition, which are separated by a certain number of 0's, such that all w_1's produced from these 1's will be entered with a shift 1. After all the letters of the initial condition have been scanned, the following string is produced:

$$\underbrace{0^{x_2}10^{\dot{y}_1}0^{\dot{x}_1}10^{\dot{y}_1}....0^{\dot{x}_1}10^{\dot{y}_1}}_{n} \qquad (9.19)$$

where $x_2 = x_1$ or $x_2 = x_1 - 1$. From (9.19) the tag system will then produce the following string:

$$\underbrace{0^{x'_2}10^{\dot{y}_1}0^{\dot{x}_1}10^{\dot{y}_1}....0^{\dot{x}_1}10^{\dot{y}_1}}_{\lfloor\frac{n}{2}\rfloor} \tag{9.20}$$

Clearly, whatever the number of 1's in the initial condition might be, they will more or less be reduced by a factor 1/2, for each application of $\overset{\circ}{\rightarrow}$, thus ultimately leading to the production of ϵ. This is the case, because for every pair of w_1's one of them will be erased due to the fact that $\dot{y}_1 + \dot{x}_1$ is always even. It thus follows that tag systems from this class will always halt, whatever the initial condition might be. A similar proof can be given for the case $w_1 = 0^{x_1}10^{y_1}$.

Case 1.4. $\#1 = 2$**,** $l_1 \equiv 0 \bmod 2$**.** To prove the case we have to differentiate between two subcases, i.e. the case with $w_1 = 0^{x_1}10^{y_1}10^{z_1}$ and $w_1 = 0^{\dot{x}_1}10^{\dot{y}_1}10^{z_1}$ (the proof for the case with $w_1 = 0^{\dot{x}_1}10^{y_1}10^{\dot{z}_1}$ is similar to the first case, the proof with $w_1 = 0^{x_1}10^{\dot{y}_1}10^{\dot{z}_1}$ is similar to the second case).

Subcase 1.4.1. $w_1 = 0^{x_1}10^{y_1}10^{z_1}$**.** The first case is proven through the following table:

Table 9.5: $w_1 = 0^{x_1}10^{y_1}10^{z_1}$

	w_1
S_0	w_1✓
S_1	w_1✓

From this proof it follows that any tag system from this class of cases will always become periodic, except when no 1 is scanned in the initial condition, then it always halts.

Subcase 1.4.2. $w_1 = 0^{\dot{x}_1}10^{\dot{y}_1}10^{z_1}$**.** The proof of the solvability of the second case follows from the following table:

Table 9.6: $w_1 = 0^{\dot{x}_1}10^{\dot{y}_1}10^{z_1}$

	w_1	$w_1 w_1$	...	$(w_1 w_1)^{n-1}$
S_0	HALT	HALT	...	HALT
S_1	$w_1 w_1$	$w_1 w_1 w_1 w_1$	....	$(w_1 w_1)^n$

Tag systems from this class will either halt or show unbounded growth depending on the parity of the length of the initial condition.

Case 1.5. #1 = 2**,** $l_1 \equiv 1 \bmod 2$**.** The tables for the proof are almost identical to those for case 1.4., except that now we have to consider the cases $w_1 = 0^{x_1}10^{\dot{y}_1}10^{z_1}$ (or similarly $w_1 = 0^{\dot{x}_1}10^{\dot{y}_1}10^{\dot{z}_1}$) and $w_1 = 0^{\dot{x}_1}10^{y_1}10^{z_1}$ (or similarly $w_1 = 0^{x_1}10^{y_1}10^{\dot{z}_1}$). Contrary to lemma 1.4.2., the tag system will always become periodic when $w_1 = 0^{x_1}10^{\dot{y}_1}10^{z_1}$, if at least two 1's are scanned in the initial condition. This is the case, because, for each two consecutive w_1's, the tag system produces two consecutive w_1's, since $z_1 + x_1$ is even. If only one 1 is scanned in the initial condition, the system will either halt or become periodic depending on the parity of the length of the initial condition. In case $w_1 = 0^{\dot{x}_1}10^{y_1}10^{z_1}$ tag systems from this class will always become periodic when at least one 1 is scanned in the initial condition since for every w_1 produced, one and only one w_1 will be produced. In all other cases, tag systems from this class halt.

Case 1.6. #1 = 3**,** $l_1 \equiv 0 \bmod 2$**.** Again we have to consider several cases, depending on the spacings between the 1's, i.e. the number of 0's between the consecutive 1's. If all spacings are uneven, i.e. if $w_1 = 0^{\dot{x}_1}10^{\dot{y}_1}10^{\dot{z}_1}10^{t_1}$ (or, $w_1 = 0^{x_1}10^{\dot{y}_1}10^{\dot{z}_1}10^{\dot{t}_1}$) the proof is similar to the one for case 1.4.2., the tag system leading to unbounded growth or a halt, depending on the parity of the initial condition and the position of the 1's in w_1. The proof of the second case follows from cases 1.4.1. and 1.4.2., since either two 1's are scanned or one 1. An example of such case is $w_1 = 0^{x_1}10^{\dot{y}_1}10^{z_1}10^{t_1}$. Based on the proofs from case 1.4., we conclude that tag systems from this second subclass will either become periodic or lead to unbounded growth, depending on the parity of the initial condition and the position of the 1's. Of course, all tag systems from this class will always halt, when no 1 is scanned in the initial condition.

Case 1.7. **$\#1 = 3$, $l_1 \equiv 1 \bmod 2$.** Again we have to differentiate between two cases: $w_1 = 0^{s_1}10^{x_1}10^{y_1}10^{t_1}$ (and all variants) or $w_1 = 0^{s_1}10^{\dot{x}_1}10^{\dot{y}_1}10^{t_1}$ (plus all variants).

Case 1.7.1. **$w_1 = 0^{s_1}10^{\dot{x}_1}10^{\dot{y}_1}10^{t_1}$.** If all 1's are unevenly spaced, i.e. if one 1 is scanned in w_1 all others will also be scanned, it can be proven that the tag system will always grow, when at least one w_1 is produced from the initial condition such that all its 1's are scanned, thus producing w_1^3. The following table shows that in case w_1^3 is produced, a tag system from this class will always lead to unbounded growth.

Table 9.7: Case $w_1 = 0^{s_1}10^{\dot{x}_1}10^{\dot{y}_1}10^{t_1}$

	w_1	w_1^3	w_1^6	...	w_1^{2m}	w_1^{2m+1}
S_0	w_1^3	w_1^6	w_1^9	...	w_1^{3m}	w_1^{3m+3}
S_1	w_1✓	w_1^3✓	w_1^9	...	w_1^{3m}	w_1^{3m}

Note that although in shift 1, w_1^3 produces w_1^3, the tag system will not become periodic. Indeed, if the tag system produces the string w_1^3 and it is entered with a shift 1, the next time w_1^3 is produced, it will be entered with a shift 0, given the fact that its length is odd.

Case 1.7.2. **$w_1 = 0^{s_1}10^{x_1}10^{y_1}10^{t_1}$.** Tag systems from this class, i.e., those for which only two 1's will be scanned in the same shift, will always lead to unbounded growth if at least one 1 is scanned in the initial condition. The table proving the result, will not be given here, since it can be easily replaced by the following reasoning. First of all, note that once two consecutive $w_1^\prime s$ have been produced, the tag system will always lead to unbounded growth grow, since two consecutive w_1's always lead to the production of at least 3 consecutive w_1's. If only one 1 has been scanned in the initial condition, leading to the production of one time w_1, the tag system will always lead to the production of two times w_1 because of the fact that l_1 is odd. Indeed, either w_1 will immediately lead to the production of two consecutive w_1's (depending on the parity of the length of the initial condition), or two consecutive w_1's will be produced the next time w_1 is produced. If no 1 is scanned in the initial condition, tag systems from this class always lead to a halt.

Case 1.8. $\#1 > 3$**,** $l_1 \equiv 0 \bmod 2$**.** In general, any tag system with $\#1 > 3$ from this class will either halt become periodic or lead to unbounded growth. For each #1, there are always three different cases to be taken into consideration. In the first case, all 1's are separated by an odd number of 0's. In generalizing Table 9.6 it follows that tag systems from this class will always halt or lead to unbounded growth, depending on the length of the initial condition and the position of the 1's in w_1.

The second case applies when $w_1 = 0^{s_1}10^{x_1}10^{\dot{x}_2}10^{\dot{x}_3}1...0^{\dot{x}_n}10^{t_1}$, when either one 1 is scanned or $\#1-1$ 1's are scanned, depending on the parity of the length of the initial condition. It easily follows from case 1.5. that a tag system from this class will either become periodic or show unbounded growth depending on the parity of the length of the initial condition and the position of the 1's in w_1.

The last case concerns those cases where, whatever shift w_1 is entered with, at least two 1's will be scanned. Clearly, tag systems from this class will always lead to unbounded growth, except when no 1 is scanned in the initial condition, since whatever shift w_1 is entered with, it will always lead to the production of at least two w_1's.

Of course for all cases, a halt will result when no 1 is scanned in the initial condition.

Case 1.9. $\#1 > 3$**,** $l_1 \equiv 1 \bmod 2$**.** In case all 1's are unevenly spaced, all 1's being scanned if one 1 is scanned in w_1, the tag system will always show unbounded growth if at least one 1 is scanned in the initial condition. This follows from generalizing the proof from case 1.7.1. In case all 1's are unevenly spaced except for

1, tag systems from this class can also be proven to always lead to unbounded growth, if at least one 1 is scanned in the initial condition. The result follows from the proof of case 1.7.2. If at least two 1's are scanned, whatever shift w_1 is entered with, it is clear that also in this case the tag systems will always lead to unbounded growth, if at least one 1 is scanned in the initial condition.
To summarize, all tag systems from this class will always lead to unbounded growth, except when no 1 is scanned in the initial condition, leading to a halt.

Case 2. $w_0 = 1$**.**

Case 2.1. $\#1 = 1$**.** In this case the length of w_1 is a determining factor to predict the behaviour of a tag system from this class. We have to differentiate between the following two cases: $2 < l_1 < 5$ or $5 \leq l_1$.

Subcase 2.1.1. $2 < w_1 < 5$**.** If $2 < l_1 < 5$ the tag system will always become periodic, except when the initial condition is equal to 0, then it will halt. Note that since $w_0 = 1, \#1 = 1, w_1$ does not contain 1, and consists merely of 0's. The result follows from the following tables. It is important to note that in the case $l_1 = 3$, although w_1 can lead to the production of w_0 and thus to a halt, this will never occur given the parity of w_1:

Table 9.8: Case $l_1 = 3$

	w_0	w_1	$w_0 w_0$
S_0	w_1	$w_0 w_0$	w_1✓
S_1	HALT	w_0✓	w_1✓

Table 9.9: Case $l_1 = 4$

	w_0	w_1	$w_0 w_0$
S_0	w_1	$w_0 w_0$	w_1✓
S_1	HALT	$w_0 w_0$	w_1✓

Subcase 2.1.2. $5 \leq w_1$**.** If $l_1 = 5$ tag systems from this class always become periodic if the initial condition is equal to: 1, 00, 10, 01, 11, 000, 001, 110, 100, 011, 010, 0000 and 0101. If it consists of only one 0, it will halt. This can easily be checked by hand. In all other cases it leads to unbounded growth. This follows from the following table:

Table 9.10: Case $l_1 = 5$

	w_0	w_1	w_0^2	w_0^3	w_1^2	w_0^5	...	w_0^n
S_0	w_1	w_0^3	w_1✓	w_1^2	w_0^5	w_1^3	...	$w_1^{\lceil n/2 \rceil}$
S_1	HALT	w_0^2	w_1✓	w_1✓	w_0^5	w_1^2✓	...	$w_1^{\lceil n/2 \rceil} - 1$✓

Although the table seems to allow for periodicity, the fact that w_1 is odd guarantees that once w_1^2 is produced, the tag system will always lead to unbounded growth. This can be easily checked by hand.

Clearly, if $l_1 > 5$, the tag systems will always show unbounded growth (if the length of the initial condition is longer than 1). The proof can be found by constructing a table similar to Table 9.10 and is left to the reader. Note that once $l_1 > 7$, the proof becomes very simple, since whatever shift w_1 is entered with, it will always lead to the production of at least 4 1's, and thus to the production of $w_1 w_1$.

Case 2.2. #1 = 2, $l_1 = 3$**.** It can be determined for any tag system from this class that it will either halt or become periodic. There are three different tag systems to be taken into account here:

$$0 \rightarrow 1 \quad 1 \rightarrow 100$$
$$0 \rightarrow 1 \quad 1 \rightarrow 010$$
$$0 \rightarrow 1 \quad 1 \rightarrow 001$$

In the following tables it is shown that all three tag systems will always become periodic, except when the initial condition is equal to 0. It should again be noted that although w_1 can lead to the production of w_0 and thus to a halt, this will never occur given the parity of w_1

Table 9.11: Case $0 \to 1, 1 \to 100$

	w_0	w_1	$w_1 w_0$	$w_0 w_1$
S_0	w_1	$w_1 w_0$	$w_1 w_0$✓	$w_1 w_0$✓
S_1	HALT	w_0✓	$w_0 w_1$	$w_1 w_0$✓

Table 9.12: Case $w_0 = 1, w_1 = 010$

	w_0	w_1	$w_0 w_0$
S_0	w_1	$w_0 w_0$	w_1✓
S_1	w_1	w_1✓	w_1✓

Table 9.13: Case $w_0 = 1, w_1 = 001$

	w_0	w_1	$w_0 w_1$	$w_1 w_0$
S_0	w_1	$w_0 w_1$	$w_1 w_0$	$w_0 w_1$✓
S_1	w_1	w_0✓	$w_0 w_1$✓	$w_0 w_1$✓

Case 2.3. $\#1 = 2, l_1 > 3$**.** For any tag system from this class it can be determined that it will either halt, become periodic or lead to unbounded growth. To prove this, we will only consider the case $l_1 = 4$ in more detail. We will first prove that once the tag system produces $w_1 w_0$, scanning the first letter of w_1 the system will always grow. There are four different tag systems in this class, i.e. $w_1 = 1000, w_1 = 0100, w_1 = 0010, w_1 = 0001$. We will only prove the first case, the

proofs of the other cases being similar to the first case. The following table illustrates why the tag system will always lead to unbounded growth, once it has produced $w_1 1$, scanning the leftmost letter in $w_1 1$.

Table 9.14: Case $0 \rightarrow 1, 1 \rightarrow 1000$

	$w_1 1$	11	w_1	$w_1 1 w_1$
S_0	$w_1 1 w_1$	w_1	$w_1 1 \checkmark$	$w_1 1 w_1 11$
S_1	11	w_1	$11 \checkmark$	$11 w_1 1$
	$w_1 1 w_1 11$	$w_1 1 w_1 11 w_1$	$w_1 1 w_1 11 w_1 11$	....
S_0	$w_1 1 w_1 11 w_1$	$w_1 1 w_1 11 w_1 11$	$w_1 1 w_1 11 w_1 11 w_1$	....
S_1	$11 w_1 1 w_1$	$11 w_1 1 w_1 w_1 1$	$11 w_1 1 w_1 w_1 1 w_1$	....
	$w_1 1 (w_1 11)^{n-1}$	$w_1 1 (w_1 11)^{n-1} w_1$	$11 w_1 1$	$w_1 11$
S_0	$w_1 1 (w_1 11)^{n-1} w_1$	$w_1 1 (w_1 11)^{n}$	$w_1 \underbrace{w_1 1 w_1}$	$w_1 1 w_1$
S_1	$11 w_1 (1 w_1 w_1)^{n-2} 1 w_1$	$11 w_1 (1 w_1 w_1)^{n-1} 1$	$w_1 11$	$11 w_1$
	$11 w_1$	$w_1 w_1 1$	1111	$w_1 w_1$
S_0	$w_1 w_1 1$	$\underbrace{w_1 1 w_1} 1 w_1$	$w_1 w_1$	$\underbrace{w_1 1 w_1} 1$
S_1	$w_1 11 \checkmark$	1111	$w_1 w_1$	$1111 \checkmark$
	$11 w_1 1 w_1$	$w_1 11 w_1 1$	$11 w_1 11$	$w_1 11 w_1$
S_0	$w_1 \underbrace{w_1 1 w_1} 11$	$\underbrace{w_1 1 w_1} 1 w_1 1$	$w_1 \underbrace{w_1 1 w_1}$	$\underbrace{w_1 1 w_1} w_1 1$
S_1	$w_1 11 w_1 1$	$11 w_1 11$	$w_1 11 w_1$	$11 w_1 11 \checkmark$

Although the table seems to allow for some periodicity, it should be noted that this can never occur once $w_1 1$ is produced and entered with a shift 0. Suppose our initial condition is $w_1 1$, then:

$$
\begin{aligned}
&w_1 1 \overset{\text{S1}}{\rightarrow} w_1 1 w_1 \overset{\text{S0}}{\rightarrow} 11 w_1 1 \overset{\text{S1}}{\rightarrow} w_1 \underbrace{w_1 1 w_1}\\
&\overset{\text{S0}}{\rightarrow} \underbrace{w_1 1}\underbrace{w_1 1 w_1 11} \overset{\text{S0}}{\rightarrow} \underbrace{w_1 1 w_1}\underbrace{11 w_1 1 w_1} \overset{\text{S0}}{\rightarrow} \underbrace{w_1 1 w_1 11}\underbrace{w_1 w_1 1 w_1 11}\\
&\overset{\text{S0}}{\rightarrow} \underbrace{w_1 1 w_1 11 w_1} 1111 \underbrace{w_1 1 w_1} \overset{\text{S0}}{\rightarrow} \underbrace{w_1 1 w_1 11 w_1 11} w_1 w_1 \underbrace{11 w_1 1}\\
&\overset{\text{S0}}{\rightarrow} \underbrace{w_1 1 w_1 11 w_1 11} 1111 \underbrace{w_1 11}.
\end{aligned}
$$

where $\overset{\text{Sx}}{\rightarrow} X \overset{\text{Sy}}{\rightarrow} Y$ means that Y results from X, if X is entered with shift Sx, i.e., its first x letters are erased without being scanned. Whatever shift the last string produced in this sequence is ever entered with, the tag system will always lead to unbounded growth. If the shift 0 remains a constant, the table shows that this must indeed happen. If the shift changes at one time to 1, it is guaranteed that the string produced will contain $w_1 1 w_1 w_1 1 w_1$ as a substring (See table). Since the length of $w_1 1 w_1$ is uneven, whenever the first sequence of $w_1 1 w_1$ is entered with a shift 0, $w_1 1 w_1$ will be entered with a shift 1, and vice versa and it is thus guaranteed that the system must lead to unbounded growth (See Table). This reasoning still does not result in a proof of the fact that this tag system will always lead to unbounded growth once $w_1 1$ is produced, and entered with a shift 0, since we have merely shown it for the case where the initial condition is equal to $w_1 1$. If we would e.g. add only one 0 to the condition, the shifts completely change. Still, our reasoning remains valid. Indeed, based on the table, one can deduce that there are only three possibilities for periodicity:

$$
\begin{aligned}
&1111 \overset{S1}{\rightarrow} w_1 w_1 \overset{Sx}{\rightarrow} 1111\\
&11 w_1 11 \overset{S1}{\rightarrow} w_1 11 w_1 \overset{S1}{\rightarrow} 11 w_1 11\\
&w_1 11 \overset{S1}{\rightarrow} 11 w_1 \overset{S1}{\rightarrow} 11 w_1
\end{aligned}
$$

Any other path through the table that does not lead to any of these periodic strings, will lead to unbounded growth, producing a string containing $w_1 1 w_1 w_1 1 w_1$ as a substring. However, these periods can only be produced if the tag system is started with initial conditions of one of the following forms:

$$\begin{array}{l} 111(1111)^n \\ 0001000(w_1 w_1)^n \\ 1100011(11 w_1 11)^n \\ 000111000(w_1 11 w_1)^n \\ 00011(w_1 1)^n \\ 11000(11 w_1^n) \end{array}$$

or any combination of these strings, with $n = \{0, 1, ...\}$. As is clear, for neither of these conditions can $w_1 1$ be entered with a shift 0. Any other initial condition, containing $w_1 1$ entered with a shift 0, will lead to unbounded growth. Indeed, although any such condition might still allow for the production of a string containing one of these periodic strings as a substring, they will always be combined with strings that grow or strings for which the shift does not remain constant. This follows from the productions as given in the table. Indeed, if we look at the different paths in the table leading to periodicity, it is clear that the lengths of the strings leading to the periodic strings change from odd to even (and vice versa). It is this fact that makes it impossible for strings to become periodic, except when the initial condition is in one of the forms as described above. If the initial condition is not a combination of these periodic strings, the fluctuation of the parity of the lengths of the several substrings, makes it impossible for the whole string to become periodic.

To summarize, the tag system analyzed through the table will always show unbounded growth, once $w_1 1$ is produced by the system, scanning its leftmost letter. The system only halts when the initial condition is equal to 0. The system becomes periodic for the following initial conditions: 000, 1, 00, 01 and the set of periodic strings described above. All other conditions will lead to the production of $w_1 1$, entered with a shift 0 and thus to growth. A similar proof can be found for the remaining cases for which $l_1 = 4$.

The general solvability of the class of tag systems with $\#1 = 2$, $w_0 = 1$, $l_1 > 4$ follows from the following considerations. First, suppose the 1 in w_1 is at an uneven position. Then, if w_1 is entered with a shift 0, it will always lead to a string longer than w_1, consisting of w_1 and $\lfloor \frac{l_1 - 1}{2} \rfloor$ times 1. If w_1 is entered with a shift

1, it will result in a string consisting of $\lfloor \frac{l_1}{2} \rfloor$ 1's.
Then, if $l_1 = 5$, the tag system will always show unbounded growth if $w_1 w_1$ is produced at least once from the initial condition. Indeed, if $w_1 w_1$ is produced, it will always lead to the production of a string consisting of one time w_1 and four times 1 (whatever shift $w_1 w_1$ is entered with) a string which will lead to the production of at least two times w_1 and two times 11, a string which clearly leads to unbounded growth.
If $l_1 = 6$ then the tag system will also always lead to unbounded growth, if $w_1 w_1$ is produced as a substring at least once. The proof is similar to the case $l_1 = 5$ and is left to the reader.
If $l_1 > 6$, once w_1 is produced from the initial condition, tag systems from this class will always lead to unbounded growth. This is the case because, whatever shift w_1 is entered with, it always leads to the production of at least 4 1's, and thus to the production of at least two w_1's or the production of one w_1 plus at least three 1's, in its turn leading to the production of at least two w_1's. To summarize, for all tag systems from this class, with $l_1 > 4$, it is clear that all but a finite number of initial conditions will always lead to unbounded growth. The remaining (finite number of) initial conditions can be easily calculated for each of the subcases, and lead to either a halt or periodicity in a finite number of steps.

Case 2.4. #1 > 2**.** We have to distinguish two cases, $l_1 = 3$ and $l_1 > 3$

Subcase 2.4.1. $l_1 > 3$ For $l_1 > 3$, if all 1's are unevenly spaced – implying that if one 1 is scanned, the other(s) will also be scanned – each of the tag systems from this class will either halt, become periodic or show unbounded growth after a finite number of steps.
In case $l_1 = 4$, there are two possible tag systems to be taken into consideration: either $w_1 = 1010$, or $w_1 = 0101$. Both tag systems will halt if the initial condition is equal to 0. For all other conditions, the tag systems will always lead to unbounded growth or become periodic. Indeed, once w_1 is produced, and this will always happen for initial conditions different from 0, the tag system cannot halt. This is the case because w_1 always leads to the production of either $w_0 w_0$

or $w_1 w_1$. If $w_0 w_0$ is produced, it will always lead to the production of w_1, while $w_1 w_1$ will lead to the production of w_0^4 or w_1^4. Whether any of these two tag systems will become periodic or show unbounded growth then depends on the length of the string S_1 produced from the initial condition after all the relevant letter of the initial condition have been processed. Clearly, in case $w_1 = 0101$, if l_{S_1} odd, the system will show unbounded growth, if l_{S_1} even, it will become periodic. For the case $w_1 = 1010$, if l_{S_1} even, the system will show unbounded growth, if l_{S_1} odd, it will become periodic. This can easily be checked through the table method.

If $l_1 > 4$, tag systems in this class, with the 1's in w_1 unevenly spaced, will always lead to unbounded growth, except when the initial condition is equal to 0. This result follows from the proof of case 2.3. for those cases with $l_1 > 4$. The details of the proof are left to the reader.

If all 1's are unevenly spaced except for one, it is trivial to prove that tag systems from this class will always lead to unbounded growth once w_1 is produced from the initial condition, i.e., for all initial conditions except for 0. Indeed, whatever shift w_1 is entered with, it will always lead to the production of at least one w_1 plus one 1.

If w_1 is such that whatever shift it is entered with, at least two 1's are scanned, it trivially follows that any tag system from this class will always lead to unbounded growth once w_1 is produced from the initial condition, i.e., it will lead to unbounded growth with any initial conditions except for 0. Indeed, whatever shift w_1 is entered with, it will always lead to the production of at least two w_1's.

Subcase 2.4.2. Case $l_1 = 3$ We still have to show that in case $l_1 = 3$, one can determine that a tag systems will either halt, become periodic or lead to unbounded growth. There are four different tag systems in this class:

$$\begin{array}{ll} 0 \to 1 & 1 \to 110 \\ 0 \to 1 & 1 \to 101 \\ 0 \to 1 & 1 \to 011 \\ 0 \to 1 & 1 \to 111 \end{array}$$

As for the last tag system of this list, it trivially follows that it will always lead to unbounded growth, except when the initial condition is equal to 0.
To prove the remaining cases, let us look at some of the possible productions from w_1, for the first tag system from the list with $w_1 = 110$ ($\dot{n}$ denotes that n is odd):

Table 9.15: Case $0 \to 1, 1 \to 110$

	w_1	$w_1 1$	w_1^2	$w_1 1 w_1$
S_0	$w_1 1$	$w_1 1 \checkmark$	$w_1 1 w_1$	$(w_1 1)^2$
S_1	$w_1 \checkmark$	w_1^2	$w_1^2 1$	w_1^3
	$(w_1 1)^2$	$w_1^2 1$	$w_1 1 w_1^2$	$(w_1 1)^2 w_1$
S_0	$(w_1 1)^2 \checkmark$	$w_1 1 w_1^2$	$(w_1 1)^2 w_1$	$(w_1 1)^3$
S_1	w_1^4	$w_1^2 1 \checkmark$	$w_1^4 1$	w_1^5
	$(w_1 1)^3$	w_1^3	$w_1 1 w_1^2 1$	$(w_1 1)^2 w_1^2$
S_0	$(w_1 1)^3 \checkmark$	$w_1 1 w_1^2 1$	$(w_1 1)^2 w_1^2$	$(w_1 1)^3 w_1$
S_1	w_1^6	$w_1^2 1 w_1$	$w_1^4 1 \checkmark$	$w_1^6 1$
	$(w_1 1)^n$	$(w_1)^{2n}$	$(w_1 1 w_1)^n, \dot{n}$	$(w_1 1 w_1)^n, n$
S_0	$(w_1 1)^n \checkmark$	$(w_1 1 w_1)^n$	$((w_1 1)^2 w_1^3)^{n-1} (w_1 1)^2$	$((w_1 1)^2 w_1^3)^n$
S_1	w_1^{2n}	$(w_1^2 1)^n$	$(w_1^3 (w_1 1)^2)^{n-1} w_1^3$	$(w_1^3 (w_1 1)^2)^n$
	$(w_1^2 1)^n, n$	$(w_1^2 1)^n, \dot{n}$	$w_1^{3n}, \dot{n}$	$(w_1 1 w_1^2)^n$
S_0	$(w_1 1 (w_1^2 1)^2)^{\frac{n}{2}}$	$(w_1 1 (w_1^2 1)^2)^{\frac{n-1}{2}} w_1 1 w_1^2$	$(w_1 1 w_1)^{\frac{3n-1}{2}} w_1 1$	$((w_1 1)^2 w_1)^n$
S_1	$(w_1^2 1 w_1 1 w_1^2 1)^{\frac{n}{2}}$	$(w_1^2 1 w_1 1 w_1^2 1)^{\frac{n-1}{2}} w_1^2 1$	$(w_1^2 1)^{\frac{3n-1}{2}} w_1$	$(w_1^4 1)^n$
	$((w_1 1)^2 w_1)^n, n$	$((w_1 1)^2 w_1)^n, \dot{n}$		
S_0	$((w_1 1)^3 w_1^5)^{\frac{n}{2}}$	$((w_1 1)^3 w_1^5)^{\frac{n-1}{2}} (w_1 1)^3$		
S_1	$(w_1^5 (w_1 1)^3)^{\frac{n}{2}}$	$(w_1^5 (w_1 1)^3)^{\frac{n-1}{2}} w_1^5$		

The table shows that the length of a string produced in this tag system can never shrink once w_1 is produced. Although the table allows for some periodicity, the tag system will always lead to unbounded growth, except when the initial condition is equal to 0, leading to a halt, or when the initial condition is of the form

$(w_1 1)^n$ always leading to periodicity. Although $w_1^2 1$ is self-reproducing when entered with a shift 1, it is not a periodic string, because its length is uneven, i.e. a string that is a concatenation of $w_1^2 1$ is not periodic. We thus conclude that this tag system will always grow unboundedly for all but a finite class of initial conditions. Similar proofs can be given for the other two cases of tag systems, with $l_1 = 3$. The details of the proofs are left to the reader.

Case 3. $w_0 = 0$**.**

Case 3.1. #1 = 0**,** $l_1 > 2$**.** It is trivial to prove that any tag system from this class will halt, since any sequence of 0's always leads to ϵ.
It should be noted that from now on, since $w_0 = 0$, any substring of 0's part of a given string produced by a tag system from the classes considered, will always ultimately lead to ϵ. In this respect, the size of any number of consecutive 0's is in a way irrelevant. Of more significance is the parity of such sequences of 0's. In the remaining sections, the sequence of 0's preceding the first 1 in w_1 and the sequence of 0's following the last 1 in w_1 will, respectively, be denoted through the indexed variables s_n and t_n (we will not e.g. use 0^{t_n} to avoid confusing notations). The intermediate sequences of 0's, separating two 1's will be denoted through indexed variables x_n, y_n and z_n. Note that for any s_n, t_n, x_n, y_n, z_n: $s_{n+1} < s_n, t_{n+1} < t_n, x_{n+1} < x_n, y_{n+1} < y_n$.

Case 3.2. #1 = 1**,** $l_1 > 2$**.** For all tag systems from this class it can be determined that they will always halt. In the following table, it is shown that the lengths of any string produced by any of these tag systems, are bounded, irrespective of the length of w_1.

	w_0	w_1	$s_2 w_1 t_2$	...	$s_n w_1 t_n$
S_0	HALT	$s_3 t_3 \to$ HALT	$s_5 t_5 \to$ HALT	...	$\epsilon \to$ HALT
S_1	0	$s_2 w_1 t_2$	$s_4 w_1 t_4$	...	w_1

From this table, it is clear that the lengths of the strings produced in these tag systems, are bounded. The reason why these systems will always halt, follows from the fact that, whatever the length of w_1, the system will ultimately produce a string in which no 1's will be scanned. This is due to the fact that the 0's surrounding the 1 in w_1 always lead to ϵ, while any sequence of 0's following a 1 will at some point become even thus leading to the erasure of the 1, following the previous 1. A more detailed proof of this last feature, i.e., that any odd sequence of 0's always leads to the production of an even sequence of 0's, will be given in the proof of case 3.3.2.1.

Case 3.3. #1 = 2**,** $l_1 > 2$**,** $l_1 \equiv 0 \bmod 2$**.** It can be determined for any tag system from this class that it will either halt, become periodic or lead to unbounded growth. We have to take into account two cases. The 1's can be unevenly or evenly spaced i.e. $w_1 = t_1 1 x_1 1 s_1$ or $w_1 = \dot{t}_1 1 \dot{x}_1 1 s_1$.[39]

Subcase 3.3.1. $w_1 = t_1 1 x_1 1 s_1$**.** The following table proves that any tag system from this class either halts or becomes periodic after a finite number of steps.

Table 9.17: Case $w_1 = t_1 1 x_1 1 s_1$

	w_0	w_1	A_1	A_2	...	A_n
S_0	w_0	$t_2 w_1 x_2 s_2 = A_1$	$t_4 A_1 x_4 s_4 = A_3$	$t_6 A_1 x_6 s_6$	...	$t_k x_k A_n s_k = t_p x_p A_1 s_p \checkmark$
S_1	HALT	$t_3 x_3 w_1 s_3 = A_2$	$t_5 x_5 A_2 s_5 = A_4$	$t_7 x_7 A_2 s_6$	...	$t_l A_{n+1} x_l s_l = t_q A_2 x_q s_q \checkmark$

As is clear from the table, a tag system from this class will always become periodic – except for those initial conditions in which no 1 is scanned – since the number of 0's surrounding A_1 becomes bounded, while, whatever shift w_1 is entered with, it will lead to the production of w_1.

[39] The proofs for the other possible combinations, $w_1 = \dot{t}_1 1 x_1 1 \dot{s}_1$ or $w_1 = t_1 1 \dot{x}_1 1 \dot{s}_1$ are identical to the proofs for these two forms.

Subcase 3.3.2. $w_1 = t_1 1 \dot{x}_1 1 \dot{s}_1$ For any tag system from this class it can be determined that it will either halt, become periodic or grow. The proof of this case is more complicated, and we have to subdivide the case in two cases: t_1, $\dot{x}_1$ or $\dot{s}_1 > 1$; $t_1 = 0$, $\dot{x}_1 = 1$, $\dot{s}_1 = 1$.

SubSubcase 3.3.2.1. t_1, $\dot{x}_1$ or $\dot{s}_1$ > 1. For any tag system from this class it can be determined that it will either halt or become periodic. Set $w_1 = t_1 1 \dot{x}_1 1 \dot{s}_1$. In shift 1, the tag system will produce a sequence of 0's from w_1, ultimately leading to a halt. In shift 0, we get:

$$A_1 = t_2 w_1 \lfloor \dot{x}_1 / 2 \rfloor w_1 s_2 \tag{9.21}$$

Depending on the shift, if $\dot{s}_1 + \lfloor \dot{x}_1 / 2 \rfloor + t_1$ is even, we get:

$$t_3 A_1 0^{n_1} \tag{9.22}$$

or:

$$t_3 0^{n_1} A_1 \tag{9.23}$$

It thus follows that if $\dot{s}_1 + \lfloor \dot{x}_1 / 2 \rfloor + t_1$ even, the tag system must ultimately become periodic, since the lengths of the possible strings produced from w_1 in this case are bounded, but never produce the empty string. Note that the tag system will always become periodic if at least one w_1 is produced, such that the tag system will scan the first letter of w_1. In any other case, it halts.
If $\dot{x}_1 + \lfloor \dot{x}_1 / 2 \rfloor + t_1$ uneven, the tag system produces:

$$A_2 = t_4 A_1 \lfloor \dot{x}_1 / 4 \rfloor A_1 s_3 \tag{9.24}$$

from (9.21), or a string merely consisting of a certain number of 0's (ultimately converging to ϵ), depending on the shift. If $\dot{x}_1 + s_2 + \lfloor \dot{x}_1 / 2 \rfloor + t_2 + t_1$ even, we get:

$$t_5 A_2 0^{n_2} \tag{9.25}$$

or:

$$t_5 0^{n_2} A_2 \tag{9.26}$$

again depending on the shift. Thus if $\dot{x}_1 + s_2 + \lfloor \dot{x}_1/2 \rfloor + t_2 + t_1$ even, the tag system will always halt or become periodic. A halt occurs, if no A_2 is produced. If this is not the case, but $\dot{s}_1 + s_2 + (x_1 - 1)/4$ uneven, the tag system produces:

$$A_3 = t_6 A_2 \lfloor (x_1 - 1)/8 \rfloor A_2 s_4 \tag{9.27}$$

from (9.24), or a sequence of 0's depending on the shift.

Generally, tag systems from this class will always become periodic or halt once a sequence $\dot{s}_1 + s_2 + s_3 + ... + s_n + \lfloor (x_1 - 1)/2^n \rfloor + t_n + ... + t_2 + t_1$, separating two consecutive A_{n-1} in A_n ($n \in \mathbb{N}, A_0 = w_1$) becomes even. Indeed, given a string $A_n = t_i A_{n-1} \lfloor \dot{x}_1/2^n \rfloor A_{n-1} \dot{s}_i$, with $\dot{s}_1 + s_2 + s_3 + ... + s_n + \lfloor \dot{x}_1/2^n \rfloor + t_n + ... + t_2 + t_1$ even, the tag system will produce either $t_i A_n 0^{n_j}$ or $t_i 0^{n_j} A_n$, with the number of 0's surrounding each A_n being bounded. Once this string A_n is produced in a tag system it must thus become periodic.

Now, it can be proven that for any w_1 in the class of tag systems we are considering, there always exist an n such that the number of 0's $\dot{s}_1 + s_2 + s_3 + ... + s_n + \lfloor \dot{x}_1/2^n \rfloor + t_n + ... + t_2 + t_1$ separating two consecutive A_{n-1} in A_n is even. The first thing to be noted is that, since the first 0 in $\dot{s}_1 + s_2 + s_3 + ... + s_n + \lfloor \dot{x}_1/2^n \rfloor + t_n + ... + t_2 + t_1$ is always erased, the number of 0's $\dot{s}_1 + s_2 + s_3 + ... + s_n + \lfloor \dot{x}_1/2^n \rfloor + t_n + ... + t_2 + t_1 - 1$ must be even if the number of 0's separating the two consecutive A_{n-1} in A_n is uneven. We thus have:

$$0\underbrace{000...000}_{i \equiv 0 \bmod 2} \tag{9.28}$$

Now, given an initial condition consisting of n 0's, $n > 1$ with n even. Processing such string, will always lead to the production of a string with uneven length. Either an uneven number of 0's is produced, or the last sequence produced consists of two 0's, leading to one 0 and thus uneven length. Since any sequence of 0's ultimately converges to 0, we can thus conclude that $\underbrace{000...000}_{i \equiv 0 \bmod 2}$ ultimately becomes uneven, and we can thus conclude that for any w_1 there always exist an n such that the number of 0's $\dot{s}_1 + s_2 + s_3 + ... + s_n + \lfloor \dot{x}_1/2^n \rfloor + t_n + ... + t_2 + t_1$ separating two consecutive A_{n-1} in A_n is even. We have thus proven that tag systems from this class either become periodic or halt. A halt occurs when all A_i have been entered with a shift 0, before A_n is produced.

SubSubcase 3.3.2.2. $t_1 = 0$**,** $x_1 = 1$**,** $s_1 = 1$ It can be proven that the only tag system in this class, with $w_1 = 1010$, will either halt or lead to unbounded growth. Clearly, if w_1 is entered with shift 0 we get $w_1 w_1$, if entered with shift 1, it will lead to a string of two 0's, and thus ultimately to ϵ. Now given A_n. Since there is always only one 0 between two consecutive A_{n-1} in A_n, the number of 0's separating such A_{n-1} is always uneven, and it is thus always the case that any A_n will either lead to the production of a string consisting of 0's, or A_{n+1} with $l_{A_{n+1}} > l_{A_n}$. In other words, this tag system will always show unbounded growth, once all 0's separating consecutive 1's from the initial condition have been erased, and there is at least one w_1 remaining in this string produced. Otherwise it will halt.

Case 3.4. $\#1 = 2$**,** $l_1 > 2$**,** $l_1 \equiv 1 \bmod 2$**.** It can be determined for any tag system from this class that it will always halt or become periodic. Again we have to consider two cases, depending on the parity of the spacing between the two 1's, i.e. $w_1 = \dot{t}_1 1 x_1 1 s_1 s$ and $w_1 = t_1 1 \dot{x}_1 1 s_1$.[40]

Subcase 3.4.1. $w_1 = \dot{t}_1 1 x_1 1 s_1$ The table that proves the result is identical to Table 9.17, and it thus follows that any tag system from this class will either halt or become periodic. It will always become periodic once w_1 is produced and entered with a shift 1, in all other cases it halts. A similar proof can be given for the case $w_1 = t_1 1 x_1 1 \dot{s}_1$.

Subcase 3.4.2. $w_1 = t_1 1 \dot{x}_1 1 s_1$**.** For any tag system from this class, it can be determined it will either halt or become periodic. We have to differentiate between two cases: t_1, x_1 or $s_1 > 1$ and $t_1 = 0$, $x_1 = 1$, $s_1 = 0$. We will not give the proof for the first case, since it is almost identical to the (rather complicated) proof of case 3.3.2.1.

In case $t_1 = 0$, $x_1 = 1$, $s_1 = 0$ we only have to consider one tag system, with $w_1 =$

[40] The proofs for the two other possible w_1 are almost identical to the proofs of these two forms.

101. For this tag system it can be determined that it will either halt or become periodic. This is shown through the following table:

Table 9.18: Case $w_0 = 0, w_1 = 101$

	w_0	w_1	$w_1\,w_1$	$w_1\,w_1\,w_0$
S_0	w_0✓	$w_1\,w_1$	$w_1\,w_1\,w_0$	$w_1\,w_1\,w_0\,w_0$
S_1	w_0‘✓	w_0✓	$w_0\,w_1\,w_1$	$w_0\,w_1\,w_1$✓
	$w_1\,w_1\,w_0\,w_0$	$w_0\,w_1\,w_1\,w_0$	$w_0\,w_0\,w_1\,w_1$	$w_0\,w_1\,w_1$
S_0	$w_1\,w_1\,w_0\,w_0$✓	$w_0\,w_0\,w_1\,w_1$	$w_0\,w_1\,w_1\,w_0$✓	$w_0\,w_0\,w_1\,w_1$✓
S_1	$w_0\,w_1\,w_1\,w_0$	$w_1\,w_1\,w_0\,w_0$✓	$w_0\,w_0\,w_1\,w_1$✓	$w_1\,w_1\,w_0$✓

This tag system will always become periodic, once it has produced w_1, entered with a shift 0. In all other cases it will halt.

Case 3.5. $\#1 > 2$**,** $l_1 > 2$**,** $l_1 \equiv 0 \bmod 2$**.** It can be determined for each tag system from this class that it will lead to unbounded growth, become periodic or halt. To prove this, we merely have to show this in detail for the case $\#1 = 3$. There are two possible cases here: all 1's are unevenly spaced, i.e.,$w_1 = t_1 1 \dot{x}_1 1 y_1 1 s_1$, or, only two of them are unevenly spaced, i.e., $w_1 = t_1 1 \dot{x}_1 1 \dot{y}_1 1 \dot{s}_1$.[41] The third case we will consider here, is the generalization of the results for $\#1 = 3$ to $\#1 > 3$.

Subcase 3.5.1. $w_1 = t_1 1 \dot{x}_1 1 y_1 1 s_1$**, #1 = 3.** Depending on the shift w_1 is entered with, the tag system will produce one of the following two strings:

$$A_1 = t_2\, w_1 \left\lfloor \dot{x}_1/2 \right\rfloor w_1 \left\lfloor y_1/2 \right\rfloor s_2 \tag{9.29}$$

or:

$$B_1 = 0^{n_1}\, w_1\, s_2 \tag{9.30}$$

[41] The proofs for all other w_1's, with all or not all 1's evenly spaced, are of course almost identical to the proofs to follow.

this second string, being again a case of w_1 (surrounded by 0's) thus allowing for the possibility of periodicity. If A_1 is produced and $s_1 + \lfloor \dot{x}_1/2 \rfloor + t_1$ is odd one of the following strings is produced from A_1:

$$A_2 = t_3 A_1 \lfloor \dot{x}_1/4 \rfloor A_1 \lfloor y_1/4 \rfloor s_4 \tag{9.31}$$

or:

$$B_2 = t_4 B_1 \lfloor \dot{x}_1/4 \rfloor B_1 \lfloor y_1/4 \rfloor s_5 \tag{9.32}$$

else, if $s_1 + \lfloor \dot{x}_1/2 \rfloor + t_1$ is even, we get:

$$C_1 = t_3 A_1 \lfloor \dot{x}_1/4 \rfloor B_1 \lfloor y_1/4 \rfloor s_5 \tag{9.33}$$

or:

$$D_1 = t_4 B_1 \lfloor \dot{x}_1/4 \rfloor A_1 \lfloor y_1/4 \rfloor s_4 \tag{9.34}$$

If B_2 is produced, and the sequence of 0's between the two consecutive $B_1's$ is odd it is clear that the tag system will produce either a string of the same form as B_2, the number of 0's between two consecutive B's being decreased [see (9.35) and (9.36)], or a string of a form similar to A_2, depending on the shift. If this sequence of separating 0's is even, we get one of the two following strings from B_2:

$$E_{1,B_2} = t_5 \lfloor 0^{n_1}/2 \rfloor B_1 s_6 \lfloor \dot{x}_1/8 \rfloor \lfloor 0^{n_1}/2 \rfloor A_1 \lfloor y_1/8 \rfloor s_7 \tag{9.35}$$

or:

$$F_{1,B_2} = t_5 \lfloor 0^{n_1}/2 \rfloor A_1 s_7 \lfloor \dot{x}_1/8 \rfloor \lfloor 0^{n_1}/2 \rfloor B_1 \lfloor y_1/8 \rfloor s_6 \tag{9.36}$$

Now, in the proof case 3.3.2.1. we showed that any number of 0's = $\dot{s}_1 + s_2 + ... + s_n + \lfloor \dot{x}_1/2^n \rfloor + t_n + ... + t_2 + t_1$ will in the end always become even. Similarly, we can prove for this case that, starting from A_1, there always exist n rsp. m such that in A_n respectively B_m the sequence of 0's separating the two A_{n-1}'s rsp. the two B_{n-1}'s becomes even. The proof is identical to that for case 3.3.2.1. We can thus conclude that once A_1 is produced, the strings that can be produced from a certain $A_{n-1}, n > 1$ will ultimately be in one of the two following forms:

$$C_n = t_{i_1} A_{n-1} \lfloor \dot{x}_1/2^n \rfloor B_{n-1} \lfloor y_1/2^n \rfloor s_{j_1} \tag{9.37}$$

or:

$$D_n = t_{i_2} B_{n-1} \left\lfloor \dot{x}_1 / 2^n \right\rfloor A_{n-1} \left\lfloor y_1 / 2^n \right\rfloor s_{j_2} \tag{9.38}$$

Similarly, the strings that can be produced from a certain $B_i, n > 1$, will ultimately either lead to a string of a form similar to an A_j [and will thus lead to one of the forms (9.37) or (9.38)] or to one of the two following forms:

$$E_{n,B_i} = 0^{n_{i,1}} B_n 0^{n_{j,1}} A_n 0^{n_{h,1}} \tag{9.39}$$

or:

$$F_{n,B_i} = 0^{n_{i,2}} A_n 0^{n_{j,2}} B_n 0^{n_{h,2}} \tag{9.40}$$

Now, given strings of the form C_n, D_n, E_{n,B_i} and F_{n,B_i}. If the sequence of consecutive 0's between the several A's and B's is odd, these string will always grow, the proofs being left to the reader. If this sequence of separating 0's is even, the strings will also always grow. In case of C_n or D_n, the strings produced from either of these strings will always consist of a string $A_n > A_{n-1}$ and a string B_{n-1}, independent of the shift.
Given strings of the forms E_{n,B_i} or F_{n,B_i}, the strings produced from these strings will also always consist of a string $A_{n+1} > A_n$ and a string B_n, independent of the shift. It is important to note, that once strings of the forms $C_n, D_n, E_{n,B_i}, F_{n,B_i}$ are produced, the sequence of 0's separating the several A's and $B's$ can also be proven to always become even, and so these new strings will in their turn lead to growth. We can thus conclude that once a tag system from this class produces a string of the form A_1 it will always grow. The system will always become periodic if strings B_1 are produced, such that the number of 0's separating consecutive $B_1's$ remains odd, the first B_1 in a string produced by the tag system being entered with a shift 1.
We still have to consider the case where all 1's are unevenly spaced. Let us suppose $w_1 = t_1 1 \dot{x}_1 1 \dot{y}_1 1 \dot{s}_1$. We can then determine for any tag system in this form that it will either halt, become periodic or lead to unbounded growth.

Subcase 3.5.2. $w_1 = t_1 1 \dot{x}_1 1 \dot{y}_1 1 \dot{s}_1$**, #1 = 3** We have to consider two cases. The first case concerns one tag system, i.e. the tag systems for which $t_1 = 0, x_1 =$

$1, y_1 = 1, s_1 = 1$. It can be proven that this tag system, with $w_1 = 101010$ will either halt or lead to unbounded growth. The proof is identical to Case 3.3.2.1. with $w_1 = 1010$ and we will thus not repeat this here.

For the second case, tag systems from this class will produce the following string from w_1:

$$A_1 = t_2 w_1 \lfloor x_1/2 \rfloor w_1 \left\lfloor y_1/2 \right\rfloor w_1 s_2 \tag{9.41}$$

or a string consisting of 0's, depending on the shift. If $\dot{s}_1 + \lfloor x_1/2 \rfloor + t_1$ even, $\dot{s}_1 + \left\lfloor y_1/2 \right\rfloor + t_1$ uneven from A_1 we get:

$$t_3 A_1 0^{n_1} \tag{9.42}$$

or:

$$t_3 0^{n_2} A_1 \lfloor (y_1 - 1)/4 \rfloor A_1 s_3 \tag{9.43}$$

depending on the shift A_1 is entered with. Similarly, if $\dot{s}_1 + \lfloor x_1/2 \rfloor + t_1$ uneven, $\dot{s}_1 + \left\lfloor y_1/2 \right\rfloor + t_1$ even we get:

$$t_3 0^{n_3} A_1 s_3 \tag{9.44}$$

or:

$$t_3 A_1 \lfloor x_1/4 \rfloor A_1 0^{n_4} \tag{9.45}$$

depending on the shift. If both $\dot{s}_1 + \lfloor x_1/2 \rfloor + t_1$ and $\dot{s}_1 + \left\lfloor y_1/2 \right\rfloor + t_1$ are even, the tag system produces either:

$$t_3 A_1 0^{n_5} A_1 s_3 \tag{9.46}$$

or:

$$t_3 0^{n_6} A_1 0^{n_7} s_3 \tag{9.47}$$

depending on the shift A_1 is entered with. If both $\dot{s}_1 + \lfloor x_1/2 \rfloor + t_1$ and $\dot{s}_1 + \left\lfloor y_1/2 \right\rfloor + t_1$ are uneven, the tag system either produces a string solely consisting of 0's, or:

$$A_2 = t_3 A_1 \lfloor x_1/4 \rfloor A_1 \left\lfloor y_1/4 \right\rfloor A_1 s_3 \tag{9.48}$$

from A_1. For this last case, we know from case 3.3.2.1. that there always exists an A_n – produced from a sequence A_{n-1} starting from A_1 – such that at least one of the sequences of 0's between two consecutive A_{n-1}'s will be even.[42] As

[42] These sequences are: $\dot{s}_1 + s_2 + s_3 + ... + s_n + \lfloor x_1/2^n \rfloor + t_n + t_{n-1} + ... + t_2 + t_1$ and $\dot{s}_1 + s_2 + s_3 + ... + s_n + \left\lfloor y_1/2^n \right\rfloor + t_n + t_{n-1} + ... + t_2 + t_1$.

long as none of these sequences becomes even, strings produced from A_1 can lead to the empty string ϵ when one of the A_i leading to A_n is entered with a certain shift. Whether a halt occurs within tag systems from this class then depends on the parity of 0's separating consecutive w_1 in the initial condition, as long as no A_n is produced.

We now still have to prove that in all other cases tag systems from this class will always lead to unbounded growth grow or become periodic. Let us suppose that at least one of the sequences $\dot{s}_1 + s_2 + s_3 + ... + s_n + \lfloor x_1/2^n \rfloor + t_n + t_{n-1} + ... + t_2 + t_1$ or $s_1 + s_2 + s_3 + ... + s_n + \left\lfloor y_1/2^n \right\rfloor + t_n + t_{n-1} + ... + t_2 + t_1$ has become even, once A_n, is produced, with $n \in \mathbb{N}$ and $A_0 = w_1$. In properly replacing all indices, the strings that can be produced from A_n will all be in one of the (generalized) forms (9.42) – (9.47). In case at least one string is produced of forms (9.43), (9.45) or (9.46) it easily follows that the tag system will always lead to unbounded growth, since it can be proven that each of the sequences of 0's separating two consecutive A_{n-1} in A_n will ultimately become even. Let us suppose this is already the case for A_n. Then, if the first A_{n-1} is entered with a shift 0, the second A_{n-1} will be entered with a shift 1 and vice versa, thus resulting in the production of a string consisting of three A_{n-1}, of which two are again separated by an even number of 0's. If the number of 0's separating two A_{n-1}'s in A_n only becomes even, when A_{i+n} is produced, *at least* three A_{n-1}'s will be produced from A_{i+n}. It thus follows that once any tag system from this class produces strings of the forms (9.43), (9.45) or (9.46) it will always grow. The proof that any such sequence separating two consecutive A_{n-1} in A_n will in the end become even, is (almost) identical to the proof of case 3.3.2.1. and will not be given here.

If strings of the forms resulting from (9.42), (9.44) or (9.47) by replacing the indices in the proper way, are produced, tag systems from this class will either grow or become periodic. If a string is produced in a tag system, consisting of a combination of these forms, the tag system can only remain periodic if the parity of the number of 0's separating consecutive A_n remains constant, such that every such form is entered with a shift that guarantees its periodicity. In all other cases, tag systems from this class will always grow, since in the end at least one of the forms (9.42), (9.44) or (9.47) will lead to the production of a form

(9.43), (9.45) or (9.46).[43]

Subcase 3.5.3. #1 > 3**.** On the basis of the proofs of the solvability of the cases $w_1 = t_1 1 x_1 1 y_1 1 \dot{s}_1$, $\#1 = 3$ and $w_1 = t_1 1 \dot{x}_1 1 y_1 1 s_1$, $\#1 = 3$ we can now prove that we can determine for any tag system with $\#1 > 3$, $w_0 = 0$, $l_1 > 3$, $l_1 \equiv 0 \bmod 2$, that it will either halt, become periodic or lead to unbounded growth. Clearly if whatever shift w_1 is entered with, at least two 1's will be scanned, it can be determined that the tag system will either halt or show unbounded growth, a halt only occurring when no 1 is scanned in the initial condition.
The two other cases are that either all 1's are unevenly spaced, or all but one are unevenly spaced. The proofs of these two cases immediately follow from the two cases $w_1 = t_1 1 \dot{x}_1 1 \dot{y}_1 1 \dot{s}_1$, $\#1 = 3$ and $w_1 = t_1 1 \dot{x}_1 1 y_1 1 s_1$, $\#1 = 3$.

Case 3.6. #1 > 2**,** $l_1 > 2$**,** $l_1 \equiv 1 \bmod 2$**.** The proof of this case is very similar to the proof of case 3.5., and is thus left to the reader.
Given Cases 1–3 we have thus proven theorem 9.4.2

□

Discussion of the proof

As is clear from the proof, proving the solvability of the halting and reachability problem for the class TS(2, 2) indeed involves considerable labor, despite the fact that once one has established some basic methods, the proofs for the several cases become rather straightforward. Most probably some of the proofs might be simplified. For example,the solvability of the cases 1.2, 1.4, 1.6., 1.8. easily follows from theorem 6.3.1, which states that the decision problem for any tag system for which the length of the respective words and v are not relative prime can be reduced to a certain number (the G.C.D. of v and the lengths of the words) of other tag systems. It follows from this theorem that the halting and reachability problem for all the tag systems with $w_0 = \epsilon$, $l_{w_1} \equiv 0 \bmod 2$ from

[43] In order to determine whether a tag system will become periodic or not, one merely has to run the tag system until the number of 0's surrounding each of the A_n in any of these forms has become constant or until a form (9.43), (9.45) or (9.46) is produced.

the class TS(2, 2) can be reduced to the halting and reachability problem of tag systems with $v = 1$. Since Wang has proven that these problems are solvable for any tag system with $v = 1$ (See Sec. 6.1.1), the result easily follows.

As was said, there is one important difference between proving this class of tag systems solvable as compared to solvability proofs for Turing machines: one has to prove the result for an infinite class of tag systems. In this respect, it was fundamental in this proof that there always exists a class of boundary cases, marking the difference between tag systems that will always halt or become periodic (except for a determined class of initial conditions), and tag systems that will show unbounded growth (except for a determined class of initial conditions). These boundary cases are partially determined by the total number of 1's in w_1 and w_0, since every time a 1 is scanned a string produced in a tag system must become longer. In this respect this proof seems closely connected to constraint 3 from Sec. 7.

Although we already concluded that this constraint can never be valid in general, it is clear that this method might be applied to certain infinite classes of tag systems to prove them solvable even if we have not yet been able to develop a general method to do this. Now, in assuming that this constraint can indeed be refined in a way that it can be used as a kind of criterium for differentiating between solvable classes and unsolvable classes, it might be used to seriously simplify our proof of the solvability of the class TS(2, 2). If we do not take into account the case for which $w_0 = \epsilon$, i.e., the more easy case, it can in fact be proven that for the class of tag systems TS(2, 2) there is but a finite subclass of tag systems for which the constraint is valid, i.e., the class $l_1 = 3, \#1 = \#0 = 2$, which can be easily proven solvable through the table method.[44] For the classes $v \geq 2, \mu > 2$ or $v > 2, \mu \geq 2$, there is always an infinite class of tag systems for which the constraint can be applied. We consider this as a fundamental difference between the class TS(2,2) and the classes TS(3,2), TS(2,3) and suspect that further research on this constraint might help to considerably simplify the

[44] We will not prove this here, because the proof is very trivial. Remember that for the constraint to work in this case, the following equation $x + (l_1 - 2)x = l_1 + 1$ must have a solution over the integers. Clearly, this equation is only solvable if either $l_1 = 2$ or $l_1 = 3$. The case $l_1 = 2$ was already excluded from the proof, given Wang's condition.

proof of Theorem 1.

The significance of constraint 3 is furthermore illustrated by the role of the parameters #1 and #0 in the proof. In fact, the estimates one can calculate on the basis of constraint 3 for determining the frontier values for #1 and #0, marking the difference between tag systems that lead to periodicity or a halt and those that lead to unbounded growth (except for some specific class of initial conditions), gives a good approximation for case 1. Using the equation:

$$\#0(l_0 - v) + \#1(l_1 - v) > 0 \tag{9.49}$$

we get that $\#1 > 2$ for case 1 to obtain tag systems that show unbounded growth.[45] Except for some of the subcases (those for which $l_1 \equiv 0 \bmod 2$) this estimate indeed fits the results of the proof for the case. In cases 2 and 3, we get $l_1\#1 > l_1 + \#1 + 1$. For case 3, the estimate only works by approximation. I.e. it gives a correct estimate in the sense that if $\#1 > 2$ one should expect unbounded growth. However, it also implies that if $l_1 > 3$, $\#1 = 2$ unbounded growth should also be expected. This does not fit the results completely, and illustrates some of the problems connected with the constraint. As for case 2, we get the same estimate. In this case, the estimate on the basis of the constraint is completely incorrect since we already get unbounded growth once $l_1 > 4$, $\#1 = 1$ (except for some special initial conditions). Of course, there is a clear explanation for the fact that tag systems from this class will show unbounded growth once l_1 exceeds a certain value. This is due to the fact that $w_0 = 1$. Although scanning a 0 has the immediate effect of a decrease in the length, if the letter 1 produced by w_0 is scanned, it has, indirectly an effect of growth. This example illustrates that constraint 3 is in need of a further refinement.

To conclude, although it is clear that constraint 3 could be used to simplify the proof in this section, it is equally clear that this will only be possible after it has been more seriously investigated.

[45] Indeed, since $\#0 = l_1 - \#1$, $v = 2$, the equation becomes $-2l_1 + \#1 l_1 = x$

9.4.3 Universality in tag systems.

The known limits of solvability and unsolvability in tag systems and Turing machines: a comparison

One of the first things to be noted with respect to limits of unsolvability in tag systems, is that their known limits of unsolvability are very high. Although in Minsky's encoding, v remains equal to 2, the number of symbols (and thus production rules) needed to simulate a Turing machine is rather high. Indeed, given a Turing machine with 2 symbols, and n states, the tag system needs $16n$ symbols. Applying this to the smallest universal machine known with two symbols, i.e. UTM(2, 18) (mentioned in [Nea06]), we get a universal tag system with $v = 2$, $\mu = 288$, the smallest class of universal tag systems thus being TS(288, 2). Even if we would accept Matthew Cook's universal Turing machine with 2 symbols, 7 states, based on the proof of the universality of rule 110, we get universality in the class TS(112, 2) which is still rather large.
The limits of solvability on the other hand are very low. Given the proof from Sec. 9.4.2 as well as the results by Wang and Post [Pos65, Wan3a], the classes of tag systems known to be solvable are TS(1,1), TS(2, 1), TS(1, 2) and TS(2,2).
In Fig. 9.1 and 9.2 an overview is given of the known limits of solvability and unsolvability in Turing machines rsp. in tag systems. As is clear from these figures, the gap between known solvable and unsolvable classes in Turing machines is significantly smaller than in tag systems. The $3n+1$-line however in tag systems is lower as compared to that in Turing machines. Furthermore, given our experimental results from the previous chapter, there is no clear reason to assume that the class of tag systems with $\mu = 2, v > 2$ is solvable. Add to this the fact that tag systems lie at the basis of some of the smallest universal systems known and the question naturally follows whether it is not possible to lower the unsolvability line in tag systems to classes that are equal or close to the classes TS(2, 3) and TS(3,2). The first most obvious thing to try to lower this unsolvability line is to construct smaller universal tag systems.

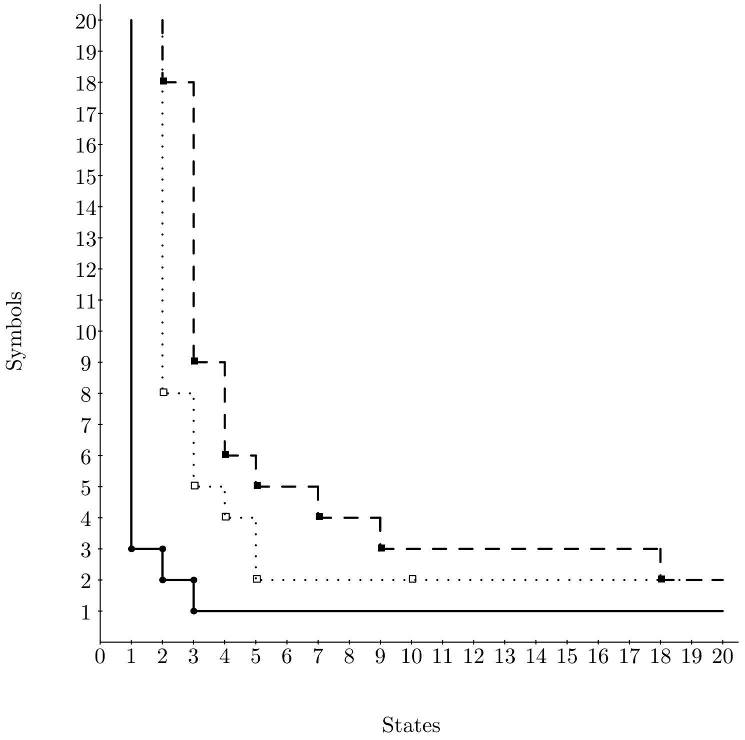


Figure 9.1: Limits of solvability and unsolvability in Turing machines. A full line denotes the solvability line, with ▪ indicating the known solvable classes. The dotted line denotes the $3n+1$-line, □ denoting the class of Turing machines to which the $3n+1$-function was reduced. The dashed line is the current unsolvability line, i.e. universality line, where ■ denotes the classes for which a universal machine has been constructed.

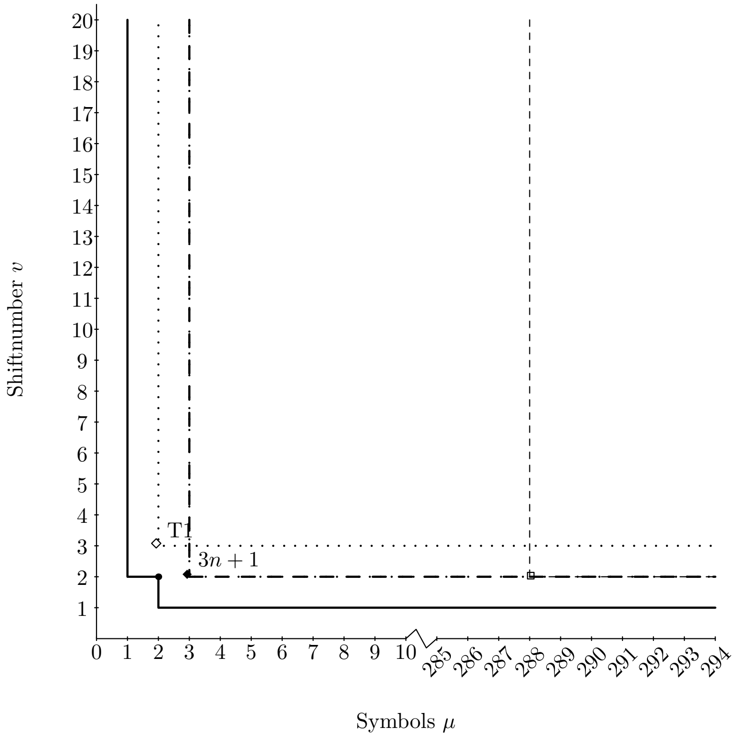


Figure 9.2: Limits of solvability and unsolvability in Tag systems. The full line indicates the solvability line, with ▪ indicating TS(2,2). The dotted line is the line indicating the line formed by Post's tag system **T1** – indicated by ◊ – the dashed dotted line indicates the $3n+1$-line in tag systems, ♦ indicating the class of tag systems to which the $3n+1$-problem was reduced. The dashed line is the current universality line in tag systems, with □ denoting the class in which a universal tag system was constructed.

Universality in tag systems with $\mu = 2$?

Given my experiences with 2-symbolic tag systems, I became more and more convinced that the class of tag systems with $\mu = 2$ is unsolvable, and I began to search for methods to encode the large universal tag system, based on Minsky's encoding and thus indirectly on the encoding of Turing machines, to this class of tag systems. Maybe it would have been more convenient if I had first tried to simply find rather small universal tag systems, without them being necessary binary, but due to my stubbornness with respect to proving the universality of a tag system with $\mu = 2$, I never spend any time on this, possibly, more interesting problem.
In this section, we will, regretfully, not give a proof of the universality of the class of tag systems with $\mu = 2$ but sketch two methods that might lead to such proof and argue that although an explicit encoding is lacking right now, there are clear reasons why one should expect universality on this level. It should be noted that the first method gives a kind of general scheme of how such an encoding might work, while the second is more specific, using the existence of different periodic structures in tag systems as discussed in Sec. 8.4. For both methods, the goal for now is to simulate a universal tag system based on Minsky's encoding.

§1. Method 1 To explain how this encoding might work, it is interesting to shortly illustrate how it is possible to encode any class of Turing machines TM(m, n) into the class TM(2, s).[46] Given for example Minsky's universal 4-symbols, 7-states machine. We can then encode each of the four symbols in binary form. For example set B to 11, x to 01, 1 to 10 and 0 to 00 and then rewrite the machine table. I will not go into the details of how to rewrite such a table. Let me merely point out that it is a rather trivial though laborious matter to do this if one does not take into account the number of states. In a similar way, one could try to encode any tag system from a class TS(μ, v) to a class TS(2, v') by encoding the m symbols into some kind of binary form. At first, drawing from the method for reducing n-symbolic Turing to 2-symbolic Turing machines, it seemed rather straightforward to prove the result. However, when applied in

[46] This was proven by Shannon [Sha56].

a direct fashion, this method cannot work for tag systems. Let me explain this through an example.

Given a tag system with 4 symbols, $v = v_4$ that we want to encode into a tag system with 2 symbols, with shift number v. Let us suppose that each of the four symbols has to be encoded by two symbols. Given v, our four symbols could then e.g. be encoded as:

$$\begin{aligned}
0 &\rightarrow 0x_{0,1}...x_{0,v-1}0y_{0,1}...y_{0,v-1}\\
1 &\rightarrow 1x_{1,1}...x_{1,v-1}0y_{1,1}...y_{1,v-1}\\
2 &\rightarrow 0x_{2,1}...x_{2,v-1}1y_{2,1}...y_{2,v-1}\\
3 &\rightarrow 1x_{3,1}...x_{3,v-1}1y_{3,1}...y_{3,v-1}
\end{aligned}$$

with $x_{i,j}, y_{i,j} \in \{0,1\}$. There is however one problem with this kind of encoding: how will one simulate in this the erasure of the first v_4 symbols from the 4-symbolic tag system? This kind of encoding never erases encodings of symbols. If we for instance concatenate the encoding for 2 and 3, the tag system will process both encodings.

Let us denote the encoding of v_4, the respective symbols and words of a tag system we want to simulate in the 2-symbolic tag system as: $\overline{v_4}, \overline{a}_0, ..., \overline{a}_{\mu-1}, \overline{w}_{a_0}, ..., \overline{w}_{a_{\mu-1}}$. Then, the only possible way to solve this erasure problem is that all $l_{\overline{a}_i}$, are such that no combination of a length smaller than $\overline{v_4}$ of these $\overline{a}_i$ is such that the length of this combination is divisible by v, while a combination of length $\overline{v_4}$ is. Suppose that we e.g. have the following sequence of symbols in our two-symbolic tag system:

$$\underbrace{\underbrace{x_{1,1}x_{1,2}x_{1,3}...x_{1,n_{a_{1,i}}}}_{\overline{a}_{1,i}}\underbrace{x_{2,1}x_{2,2}x_{2,3}...x_{2,n_{\overline{a}_{2,i}}}}_{\overline{a}_{2,i}}\underbrace{x_{3,1}x_{3,2}x_{3,3}...x_{3,n_{\overline{a}_{3,i}}}}_{\overline{a}_{3,i}}......\underbrace{x_{s,1}x_{s,2}x_{s,3}...x_{s,n_{\overline{a}_{n,i}}}}_{\overline{a}_{n,i}}}_{l_{\overline{a}_{1,i}\ \overline{a}_{2,i}\ \overline{a}_{3,i}...\overline{a}_{n,i}}\equiv 0 \text{ mod } v, s=\overline{v_4}} \tag{9.50}$$

with $x_{i,j} \in \{0,1\}$. This encoding should be such that the tag system will only begin to scan the first symbol in any $\overline{a}_{i,j}$ once $j = s+1$. In this way one can control the simulation of v_4. Indeed, it is only when the encoding of $a_{s,i}$ has been scanned, that the next letter scanned will again be the first symbol of $\overline{a}_{s+1,i}$, and the system can again start scanning a sequence of 0's and 1's.

The encoding of the a_i should be thus that given such a string of the form (9.50) the sequence of 0's and 1's scanned throughout this sequence results in a sequence of words w_0 and w_1 such that their combination results in the encoding of $w_{a_{1,i}}$. Although this kind of encoding seems theoretically possible we are confronted with the problem that given a tag system of μ symbols, any combination of 0's and 1's being encodings of the sequence of symbols $a_{i,2}...a_{i,\overline{v}}$ following the encoding of a specific symbol $a_{1,i}$ should lead to a combination of words w_0 and w_1 that results in the encoding of $w_{a_{i,1}}$. For example, if the tag system we want to simulate contains 10 symbols, with $v = 3$. Suppose that we would encode each of these symbols by binary combinations of length 4 in a binary tag system, with shift number v_2. We must now determine v_2 and the length of the encodings for our binary tag system such that the lengths of the encodings and v_2 are such that any combination of 3 encodings is divisible by v_2 while no shorter combination is. A solution to this problem is to encode any symbol by a binary string of length 10, and set $v_2 = 3$. Then, for each such encoding, 3 or 4 letters will be scanned. Given a string in this binary tag system consisting of a number of encodings of length 10. When the first 0 or 1 in this string is entered, and thus the encoding of the first symbol, the tag system indeed has to go through 3 encoded symbols before the first letter of an encoded symbol is scanned again. However, since each encoding of a symbol can be followed by any combination of 2 encoded symbols, while we have 10 different symbols, any combination of 2 encodings following the encoding of the first symbols, should lead to the same combination of 0's and 1's, to correctly encode the word corresponding to the encoding of the first symbol. For this specific example, this means we have 100 possible combinations of encodings of symbols that can follow the first symbol![47]

Still, given the encoding of the universal tag system, this kind of encoding is not necessary a problem, since the number of combinations of two symbols is highly restricted. Indeed, for the simulation to work, for each of the symbols, there is only one combination. For example, any A_i is always followed by an x,

[47] Indeed, if we denote our symbols from the 10-symbolic tag system as the decimal digits, going from 0 to 9, it immediately follows that we have 100 different combinations of these digits, of length 2 (starting with 00, ending with 99.

so the encoding should only work for $A_i x$. If such an encoding would be possible we would have found a two-symbolic tag system that can simulate any Turing machine, and is thus universal. For this abstract method to work however, we have to find the right encodings for all symbols a_i such that any of the possible combinations of two symbols $\overline{a}_i$ in the binary tag system that is possible in the universal tag system when started with the right initial conditions, leads to the right combination of words w_0 and w_1. We have not really searched for such an encoding, but we suspect that any such encoding can only be found with the help of a computer. Indeed, since one needs 288 different encodings which each have to lead to the right encodings – where each resulting encoding in its turn has to lead to the right encoding – it might be a rather intractable problem to find such an encoding if one would not use some kind of search algorithm that searches for the right words w_0 and w_1 and a shift number v, fulfilling certain constraints. In this respect it should be noted that although this encoding does not start from an existing tag system, but is used to construct one, the hypothetical algorithm that might be used to find such encoding will most probably involve some kind of analysis of behaviour of the tag systems constructed.

§2. Method 2 The idea of the second method resulted from the observation that a tag system can produce several periodic structures (See Sec. 6.1.2 and 8.4). For example, in Post's tag system **T1** one can construct an infinity of periodic structures which are all combinations of a finite number of periodic structures. The method to be described here might be used to simulate a universal tag system constructed through Minsky's encoding. As is clear from the description of Minsky's encoding (See Sec. 6.1.1), there are three basic operations such universal tag system must be able to perform: it must be able to recognize whether a given substring is of even or uneven length, it must be able to half or double substring and it must be able to produce new symbols.

It is important to remember here, that every instruction (print symbol, move to left or right, goto state t') of a Turing machine, performed in state t scanning symbol s, is simulated in the tag system by starting from a sequence:

$$A_{t,s}x(\alpha_{t,s}x)^n B_{t,s}x(\beta_{t,s}x)^m$$

from which the tag system, after the instructions have been performed, produces a sequence:

$$A_{t',s'}x(\alpha_{t',s'}x)^{n'} B_{t',s'}x(\beta_{t',s'}x)^{m'}$$

where t' and s' are the new state and the symbol scanned, and m' and n' are the result of doubling and halving m rsp. n (or vice versa, depending on whether the machine step simulated included a move to the left or to the right).

In Minsky's encoding, $v = 2$, and the use of pairs of symbols is thus an important aspect of the encoding. Although the second symbol of a pair is not scanned, it is fundamental for the recognition of the parity of a substring. In the theoretical encoding to be discussed here, we will not differentiate between first and second symbol of a pair: in this encoding, a pair of symbols that leads to a string of half or double length is encoded as a binary string that halves or doubles through the production rules of the tag systems. As a consequence, the differentiation between first and second symbol is completely arbitrary.

The parity of a given substring is recognized, in that an even number of a given class of binary strings will lead to a different shift as compared to an odd number of binary strings. It should also be noted that where one has to differentiate between several $\alpha_{t,s}x$ and $\beta_{t,s}x$ depending on t and s in Minsky's encoding, this will not be done in the encoding to be discussed here, since any $\alpha_{t,s}x$ and $\beta_{t,s}x$ is encoded in the same way. We will thus use αx and βx. Furthermore in Minsky's encoding each symbol $A_{t,s}, \alpha_{t,s}, B_{t,s}, \beta_{t,s}$ results in a sequence of productions of new symbols – $(\alpha_{t,s}x)$ e.g. first produces $c_{t,s}xc_{t,s}x$ or $s_{t,s}$, these symbols in their turn producing new symbols in their turn producing $\alpha_{t',s'}x$ – finally leading to $A_{t',s'}x, (\alpha_{t',s'}x), B_{t',s'}x, (\beta_{t',s'}x)$. In our encoding we will not differentiate between the original pair of symbols and the sequence of symbols it leads to. In other words, every time we e.g. use αx this not only indicates $\alpha_{t,s}x$ but also any symbol resulting from it e.g. $s_{t,s}s_{t,s}$, $d_{t,s,1}d_{t,s,0}$ (or $d_{t,s,0}d_{t,s,1}$) and $\alpha_{t',s'}$. This is done for the ease of notation. It will always be clear from the context what stage of the simulation we are discussing.

As is the case with Minksy's encoding, in general there will be two classes of symbols: the pairs of symbols αx and βx that have to be halved or doubled and the symbols Bx and Ax that are used as symbol producing symbols and furthermore assure that, irrespective of whether a certain substring has become odd or even, after the machine's configuration has been changed, the new resulting sequence is of the form $A_{t,s}x(\alpha_{t,s}x)^n B_{t,s}x(\beta_{t,s}x)^m$.

In our theoretical encoding the symbols Ax and Bx will play a very special role: they will serve as a kind of shift controlling devices and are furthermore used to simulate the operation of printing symbols when necessary. Indeed, the role of $A_{t,s}x$ and $B_{t,s}x$ is such that the sequences of αx and βx are entered with the right shifts for a certain number of times. Furthermore, it are the symbols $A_{t,s}x$ and $B_{t,s}x$ that are used for making the right state transitions depending on whether in the simulation a 0 or a 1 was identified (by using the parity of a given substring).

As was said, this encoding relies on the existence of certain periodic structures in several tag systems, such as those which are the most common in **T1**. The idea of using such periodic structures here is based on the fact that the periods of these structures can remain constant, while their length is a product of the period.[48] For example, there is an infinite number of strings with period 6, e.g. all strings $(10111011101000000)^n$, with $n > 0$. One can then wonder whether it is possible for a tag system to produce $(10111011101000000)^{2n}$ or $(10111011101000000)^{n/2}$ from $(10111011101000000)^n$, once $(10111011101000000)^n$ is entered with a shift different from 0. In other words, given $(10111011101000000)^n$, can one find a sequence of shifts which lead to a string with the same period, of double or halved length. In general, given a tag system T, this problem is to find periodic structures with letters $x_1x_2...x_t$, $x_i \in \Sigma$, such that T can produce $(x_1x_2...x_t)^{2n}$ or $(x_1x_2...x_t)^{\lfloor n/2 \rfloor}$ from $(x_1x_2...x_t)^n$ when entered with a certain shift.

For the encoding to work, we can use periods of type 1, and the method thus only works for tag systems for which this type exists. It should however be noted that a similar kind of encoding might be found for periods of type 3 since these

[48] For more details, the reader is referred to the discussion on periodic structures in tag systems of Sec. 6.1.2 and 8.4.

structures easily allow for the construction of an infinite number of periodic strings, having certain properties also shared by periods of type 1, properties that are basic for the encoding to work. We have not studied this possibility, so for now we will restrict our attention to periods of type 1.

In order to check whether the halving and doubling operation is possible for **T1**, I started from $(111000)^2$ and then applied the table method, described in Sec. 7.3.4, to this string. In other words, I looked at all the theoretically possible productions from $(111000)^2$. I did not construct this table by hand, but used an algorithm implementing the method, to generate these production, and then checked whether there existed any path through the resulting table that lead to 111000 or $(111000)^4$ (or any of the other 5 variants of a period 6, i.e. 011100, 001110, 000111, 100011, 110001). This is indeed the case . For example, 111000 is produced from $(111000)^2$ as follows:

$$\xrightarrow{S0} (w_0^3 w_1^3)^2 \xrightarrow{S0} w_1^2 w_0 w_1 w_0^2 \xrightarrow{S0} (w_1^2 w_0^4)^2 \xrightarrow{S1} w_1^2 w_0^5 w_1 w_0^3 \xrightarrow{S2} w_1^3 w_0^3 w_1 w_0^2 \xrightarrow{S2}$$
$$w_0 w_1^3 w_0^4 \xrightarrow{S0} w_1^2 w_0 w_1 w_0^3 \xrightarrow{S1} w_1^2 w_0^5 \xrightarrow{S1} w_1^3 w_0^3$$

where $\xrightarrow{Si} y \xrightarrow{Sj} x$ means string x results from string y, when y is entered with a shift i and all its relevant letters have been scanned. Let us call the sequence of shifts leading from string A to string B the path from A to B. In table 9.19 some of the paths leading from $(111000)^2$ to period 6 structures of double or half length are shown. These are put in bold.

Table 9.19: Some possible productions from the periodic structure $(111000)^2$.

	$w_1^3 w_0^3 w_1^3 w_0^3$, L =1
S_0	$w_1^2 w_0 w_1 w_0^2 w_1^2 w_0 w_1 w_0^2$
S_1	$w_1^3 w_0^3 w_1^3 w_0^3$
S_2	$w_0 w_1^3 w_0^3 w_1^3 w_0^2$

	$w_1^2 w_0 w_1 w_0^2 w_1^2 w_0 w_1 w_0^2$, L =2
S_0	$w_1^2 w_0^4 w_1^2 w_0^4$
S_1	$w_1^5 w_0 w_1^5 w_1$
S_2	$w_0 w_1 w_0 w_1 w_0^3 w_1 w_0 w_1 w_0^2$

	$w_1^2 w_0^4 w_1^2 w_0^4$, L =3
S_0	$w_1^2 w_0^5 w_1 w_0^3$
S_1	$w_1^3 w_0^2 w_1^2 w_0^4$
S_2	$w_0 w_1 w_0^3 w_1^3 w_0^2$

	$w_1^5 w_0 w_1^5 w_1$, L =3
S_0	$w_1^2 w_0 w_1^3 w_0^3 w_1^3 w_0 w_1 w_1$
S_1	$w_1^3 w_0 w_1^5 w_0 w_1^3 w_0^2$
S_2	$w_0 w_1^3 w_0 w_1 w_0 w_1^3 w_0 w_1^3$

	$w_1^2 w_0^5 w_1 w_0^3$, L =4
S_0	$w_1^2 w_0^4 w_1^2 w_0^2$
S_1	$w_1^3 w_0^3 w_1 w_0^2$
S_2	$w_0 w_1 w_0^7$

	$w_1^3 w_0^2 w_1^2 w_0^4$, L =4
S_0	$w_1^2 w_0 w_1 w_0^3 w_1 w_0^3$
S_1	$w_1^3 w_0^2 w_1^2 w_0^4$ ✓
S_2	$w_0 w_1^3 w_0 w_1^3 w_0^2$

	$w_0 w_1 w_0^3 w_1^3 w_0^2$, L =4
S_0	$w_0 w_1 w_0^2 w_1^2 w_0 w_1 w_0^2$
S_1	$w_0^4 w_1^3 w_0^2$
S_2	$w_1^2 w_0^3 w_1^3 w_1$

	$w_1^2 w_0 w_1^3 w_0^3 w_1^3 w_0 w_1 w_1$, L =4
S_0	$w_1^2 w_0^3 w_1^3 w_0^3 w_1^5$
S_1	$w_1^5 w_0 w_1 w_0^2 w_1^2 w_0 w_1 w_0 w_1 w_1$
S_2	$w_0 w_1 w_0 w_1^3 w_0^3 w_1^3 w_0^4$

	$w_1^3 w_0^3 w_1 w_0^2$, L =5
S_0	$w_1^2 w_0 w_1 w_0^2 w_1^2 w_1$
S_1	$w_1^3 w_0^3 w_1 w_0^2$ ✓
S_2	$w_0 w_1^3 w_0^4$

	$w_0 w_1 w_0^7$, L =5
S_0	$w_0 w_1 w_0^5$
S_1	w_0^7 ✓

S_2	$w_1^2 w_0^4$
	$w_1^2 w_0 w_1 w_0^3 w_1 w_0^3$, L =5
S_0	$w_1^2 w_0^5 w_1 w_0^2 \checkmark$
S_1	$w_1^5 w_0^5$
S_2	$w_0 w_1 w_0 w_1 w_0^2 w_1^2 w_0^2$
	$w_0 w_1 w_0^2 w_1^2 w_0 w_1 w_0^2$, L =5
S_0	$w_0 w_1 w_0^3 w_1 w_0 w_1 w_0^2$
S_1	$w_0^3 w_1^2 w_0^4$
S_2	$w_1^2 w_0 w_1^5 w_1$
	$w_0^4 w_1^3 w_0^2$, L =5
S_0	$w_0^3 w_1^3 w_0^2$
S_1	$w_0^4 w_1^3 w_1$
S_2	$w_0^2 w_1^2 w_0 w_1 w_0^2$
	$w_1^2 w_0^3 w_1^3 w_0^3 w_1^5$, L =5
S_0	$w_1^2 w_0^3 w_1^3 w_0^3 w_1^3 w_0 w_1^3$
S_1	$w_1^3 w_0^3 w_1^3 w_0^3 w_1^3 w_0 w_0$
S_2	$w_0 w_1 w_0^2 w_1^2 w_0 w_1 w_0^2 w_1^2 w_0 w_1^3 w_1$
	$w_1^5 w_0 w_1 w_0^2 w_1^2 w_0 w_1 w_0 w_1 w_1$, L =5
S_0	$w_1^2 w_0 w_1^3 w_0^4 w_1^2 w_0^6$
S_1	$w_1^3 w_0 w_1^5 w_0 w_1^7$
S_2	$w_0 w_1^3 w_0 w_1 w_0 w_1 w_0^3 w_1 w_0 w_1 w_0 w_1 w_1$
	$w_0 w_1^3 w_0^4$, L =6
S_0	$w_0 w_1^3 w_0^4 \checkmark$
S_1	$w_0^2 w_1^3 w_0^2$
S_2	$w_1^2 w_0 w_1 w_0^3$
	$w_1^2 w_0^4$, L =6
S_0	$w_1^2 w_0^4 \checkmark$
S_1	$w_1^3 w_0^2$
S_2	$w_0 w_1 w_0^3$
	$w_0 w_1 w_0 w_1 w_0^2 w_1^2 w_0^2$, L =6
S_0	$w_0 w_1 w_0 w_1 w_0^3 w_1 w_0^2$

S_1	$w_0^5 w_1^2 w_0^2$
S_2	$w_1^4 w_0 w_1^3 w_1$
	$w_0^3 w_1^2 w_0^4$, L =6
S_0	$w_0^2 w_1^2 w_0^4$
S_1	$w_0^2 w_1^3 w_0^2$ ✓
S_2	$w_0^3 w_1 w_0^3$
	$w_0^3 w_1^3 w_0^2$, L =6
S_0	$w_0^2 w_1^2 w_0 w_1 w_0^2$ ✓
S_1	$w_0^2 w_1^3 w_0^2$ ✓
S_2	$w_0^3 w_1^3 w_1$
	$w_0^2 w_1^2 w_0 w_1 w_0^2$, L =6
S_0	$w_0^3 w_1 w_0 w_1 w_0^2$ ✓
S_1	$w_0 w_1^2 w_0^4$
S_2	$w_0 w_1^5 w_1$
	$w_1^2 w_0^3 w_1^3 w_0^3 w_1^3 w_0 w_1^3$, L =6
S_0	$w_1^2 w_0^3 w_1^3 w_0^3 w_1^3 w_0^3 w_1^3$
S_1	$w_1^3 w_0^3 w_1^3 w_0^3 w_1^5 w_0 w_0$
S_2	$w_0 w_1 w_0^2 w_1^2 w_0 w_1 w_0^2 w_1^2 w_0 w_1 w_0 w_1^3 w_1$
	$w_1^3 w_0 w_1^5 w_0 w_1^7$, L =6
S_0	$w_1^2 w_0 w_1 w_0 w_1^3 w_0 w_1^5 w_0 w_1^3 w_0 w_1^3$
S_1	$w_1^3 w_0^3 w_1^3 w_0 w_1 w_0 w_1^3 w_0 w_1^3 w_0 w_0$
S_2	$w_0 w_1^5 w_0 w_1^3 w_0^3 w_1^3 w_0 w_1^3 w_1$
	$w_0^2 w_1^3 w_0^2$, L =7
S_0	$w_0^3 w_1^3 w_1$ ✓
S_1	$w_0 w_1^2 w_0 w_1 w_0^2$ ✓
S_2	$\mathbf{w_0 w_1^3 w_0^2}$
	$w_1^2 w_0 w_1 w_0^3$, L =7
S_0	$w_1^2 w_0^5$
S_1	$w_1^5 w_0^2$ ✓
S_2	$w_0 w_1 w_0 w_1 w_0^2$
	$w_1^3 w_0^2$, L =7

S_0	$w_1^2 w_0 w_1 w_0^2$
S_1	$w_1^3 w_0^2$ ✓
S_2	$w_0 w_1^3 w_1$
	$w_0^5 w_1^2 w_0^2$, L =7
S_0	$w_0^5 w_1 w_0^2$
S_1	$w_0^3 w_1^2 w_0^2$
S_2	$w_0^3 w_1^3 w_1$ ✓
	$w_0^2 w_1^2 w_0^4$, L =7
S_0	$w_0^3 w_1 w_0^3$ ✓
S_1	$w_0 w_1^2 w_0^4$ ✓
S_2	$\mathbf{w_0 w_1^3 w_0^2}$ ✓
	$w_0^3 w_1^3 w_1$, L =7
S_0	$w_0^2 w_1^2 w_0 w_1 w_1$ ✓
S_1	$w_0^2 w_1^3 w_0^2$ ✓
S_2	$\mathbf{w_0^3 w_1^3}$
	$w_0 w_1^2 w_0^4$, L =7
S_0	$\mathbf{w_0 w_1^3 w_0^2}$ ✓
S_1	$w_0^2 w_1 w_0^3$
S_2	$w_1^2 w_0^4$ ✓
	$w_1^3 w_0^3 w_1^3 w_0^3 w_1^5 w_0 w_0$, L =7
S_0	$w_1^2 w_0 w_1 w_0^2 w_1^2 w_0 w_1 w_0^2 w_1^2 w_0 w_1^3 w_0^3$
S_1	$w_1^3 w_0^3 w_1^3 w_0^3 w_1^3 w_0 w_1^5$
S_2	$w_0 w_1^3 w_0^3 w_1^3 w_0^3 w_1^3 w_0 w_1 w_0 w_0$
	$w_1^3 w_0^3 w_1^3 w_0 w_1 w_0 w_1^3 w_0 w_1^3 w_0 w_0$, L =7
S_0	$w_1^2 w_0 w_1 w_0^2 w_1^2 w_0 w_1 w_0 w_1 w_0 w_1^3 w_0^3 w_1^5$
S_1	$w_1^3 w_0^3 w_1^3 w_0^5 w_1^5 w_0 w_1 w_0 w_0$
S_2	$w_0 w_1^3 w_0^3 w_1^7 w_0 w_1 w_0 w_1^3 w_0^3$
	$w_0 w_1^5 w_0 w_1^3 w_0^3 w_1^3 w_0 w_1^3 w_1$, L =7
S_0	$w_0 w_1^3 w_0 w_1^5 w_0 w_1 w_0^2 w_1^2 w_0 w_1 w_0 w_1^3 w_0^2$
S_1	$w_0^2 w_1^3 w_0 w_1 w_0 w_1^3 w_0^3 w_1^3 w_0^3 w_1^3$

S_2	$w_1^2 w_0 w_1^3 w_0^3 w_1^3 w_0^3 w_1^5 w_0 w_1 w_1$
	$w_0 w_1^3 w_0^2$, L =8
S_0	$\mathbf{w_0 w_1^3 w_0^2}$✓
S_1	$\mathbf{w_0^2 w_1^3 w_0}$
S_2	$w_1^2 w_0 w_1 w_0^2$✓
	$w_1^2 w_0^5$, L =8
S_0	$w_1^2 w_0^4$✓
S_1	$\mathbf{w_1^3 w_0^3}$
S_2	$w_0 w_1 w_0^4$✓
	$w_0 w_1^3 w_1$, L =8
S_0	$\mathbf{w_0 w_1^3 w_0^2}$✓
S_1	$w_0^2 w_1^3$
S_2	$w_1^2 w_0 w_1 w_1$
	$w_0^3 w_1^2 w_0^2$, L =8
S_0	$w_0^2 w_1^2 w_0^2$✓
S_1	$\mathbf{w_0^2 w_1^3 w_0}$✓
S_2	$w_0^3 w_1 w_0^2$✓
	$w_0^3 w_1^3$, L =8
S_0	$w_0^2 w_1^2 w_0 w_0$✓
S_1	$\mathbf{w_0^2 w_1^3 w_0}$✓
S_2	$\mathbf{w_0^3 w_1^3}$✓
	$w_1^3 w_0^3 w_1^3 w_0^3 w_1^3 w_0 w_1^5$, L =8
S_0	$w_1^2 w_0 w_1 w_0^2 w_1^2 w_0 w_1 w_0^2 w_1^2 w_0 w_1 w_0 w_1^3 w_0 w_1^3$
S_1	$w_1^3 w_0^3 w_1^3 w_0^3 w_1^3 w_0^3 w_1^3 w_0 w_0$
S_2	$w_0 w_1^3 w_0^3 w_1^3 w_0^3 w_1^5 w_0 w_1^3 w_1$
	$w_0 w_1^3 w_0^3 w_1^7 w_0 w_1 w_0 w_1^3 w_0^3$, L =8
S_0	$w_0 w_1^3 w_0^3 w_1^3 w_0 w_1^3 w_0 w_1 w_0 w_1 w_0 w_1^3 w_0^3$
S_1	$w_0^2 w_1^3 w_0^3 w_1^3 w_0 w_1^3 w_0^5 w_1^3 w_0^2$
S_2	$w_1^2 w_0 w_1 w_0^2 w_1^2 w_0 w_1^3 w_0 w_1^7 w_0 w_1 w_0^2$
	$w_0^2 w_1^3 w_0 w_1 w_0 w_1^3 w_0^3 w_1^3 w_0^3 w_1^3$, L =8
S_0	$w_0^3 w_1^7 w_0 w_1 w_0^2 w_1^2 w_0 w_1 w_0^2 w_1^2 w_0 w_0$

S_1	$w_0 w_1^2 w_0 w_1 w_0 w_1 w_0 w_1^3 w_0^3 w_1^3 w_0^3 w_1^3 w_1$
S_2	$w_0 w_1^3 w_0^5 w_1^3 w_0^3 w_1^3 w_0^3 w_1^3$
	$w_0^2 w_1^3 w_1$, L =9
S_0	$\mathbf{w_0^3 w_1^3}$✓
S_1	$w_0 w_1^2 w_0 w_1 w_1$
S_2	$\mathbf{w_0 w_1^3 w_0^2}$✓
	$w_1^3 w_0^3$, L =9
S_0	$w_1^2 w_0 w_1 w_0^2$✓
S_1	$w_1^3 w_0^3$✓
S_2	$\mathbf{w_0 w_1^3 w_0^2}$✓
	$w_1^3 w_0^3 w_1^3 w_0^3 w_1^3 w_0^3 w_1^3 w_0 w_0$, L =9
S_0	$w_1^2 w_0 w_1 w_0^2 w_1^2 w_0 w_1 w_0^2 w_1^2 w_0 w_1 w_0^2 w_1^2 w_0 w_1 w_0 w_0$
S_1	$\mathbf{w_1^3 w_0^3 w_1^3 w_0^3 w_1^3 w_0^3 w_1^3 w_0^3}$
S_2	$w_0 w_1^3 w_0^3 w_1^3 w_0^3 w_1^3 w_0^3 w_1^5$
	$w_0^2 w_1^3 w_0^3 w_1^3 w_0 w_1^3 w_0^5 w_1^3 w_0^2$, L =9
S_0	$w_0^3 w_1^3 w_0^3 w_1^5 w_0 w_1 w_0^5 w_1^3 w_1$
S_1	$w_0 w_1^2 w_0 w_1 w_0^2 w_1^2 w_0 w_1 w_0 w_1^3 w_0^4 w_1^2 w_0 w_1 w_0^2$
S_2	$\mathbf{w_0 w_1^3 w_0^3 w_1^3 w_0^3 w_1^3 w_0^3 w_1^3 w_0^2}$
	$w_0 w_1^3 w_0^5 w_1^3 w_0^3 w_1^3 w_0^3 w_1^3$, L =9
S_0	$w_0 w_1^3 w_0^4 w_1^2 w_0 w_1 w_0^2 w_1^2 w_0 w_1 w_0^2 w_1^2 w_0 w_0$
S_1	$\mathbf{w_0^2 w_1^3 w_0^3 w_1^3 w_0^3 w_1^3 w_0^3 w_1^3 w_0}$
S_2	$w_1^2 w_0 w_1 w_0^5 w_1^3 w_0^3 w_1^3 w_0^3 w_1^3$

As is clear from the table, there are paths of several length leading from $(111000)^2$ to a periodic 6 structure of half or double length. Note, that not all possible paths were tested. In the table only paths of maximum length 9 were tested. We did not give a complete table of all possible paths of length 9, since this would result in a table of over 100 pages long. The strings marked, are strings that were already produced through another path, possibly not completely represented in the table.

As was said, more research is needed here. Other tag systems containing periodic structures such as those in **T1** should be tested. Furthermore, other periodic structures in **T1** should be tested. For now, we use the example of $(111000)^2$ to explain the idea behind the encoding.

To halve or double the length of a string in any tag system is rather trivial. However, if we want to apply the halving and doubling capacities of a tag system to simulate a universal tag system, this operation must be controlled in some or the other way. This is indeed the case when halving or doubling with periodic structures. Given a string consisting of n times a periodic structure P, one must find paths that lead from each pair of P's to four P's or only one P. As is clear from the example of **T1** this possibility indeed exists.

There are however several problems with this kind of encoding that have to be taken into account. First of all, given a sequence of pairs of PP's, the shifts necessary to transform one PP in either P or $PPPP$, do not necessarily lead to the right sequence of shifts to transform the next pair of PP in P rsp. $PPPP$. Indeed, given e.g. the sequence of shifts leading from $(111000)^2$ to 111000, i.e. 00012211. If $(111000)^2$, divisible by 3, is entered with a shift 0, the resulting string is again of a length divisible by 3. Entering this string with a shift 0 however, gives a structure $(11000)^2$ resulting in a string of length 32, not divisible by 3. From this point on, there are four more structures produced that result in strings with lengths not divisible by 3. If we would then e.g. have $PPPP$, the first two P's might lead to the desired result, but the last two might not, because from a given point on the shifts, necessary for P to be produced from PP, change.

There are two possible solutions to this problem. A first solution would result if one would find paths in a certain tag system such that the structures resulting from it lead to strings with lengths divisible by v. A second solution could be that the new sequence of shifts resulting from going from PP to P, or from PP to $PPPP$, is a path that also leads from PP to P rsp. $PPPP$, i.e. one should find paths of shifts that are in a way synchronized with each other, taking into account the lengths of the strings resulting from the structures. For example given the path 00012211, taking into account the lengths of the strings produced from the structure, the following paths should also lead to the structure

111000: 001000111 and 002222211. This is not the case, so, despite the fact that $(111000)^2$ can lead to 111000 in Post's tag system, given a certain path, it won't work for the encoding we want here. Still, we do not see any reason why one of the two solutions would not be possible in a tag system with $\mu = 2$.

Supposing this is possible, we also need means to control these paths. It are the encodings of the symbols $A_{t,s}x$ and $B_{t,s}$ that are used to serve this goal. As was said before, part of the function of these encodings is that they should be regarded as a kind of shift controlling devices. What do we mean with this? First of all, given an operation of a Turing machine in state t, scanning symbol s, $A_{t,s}x$ and $B_{t,s}$ should be thus that they are transformed in a sequence of strings, such that their respective lengths result in the right shift at the right moment. Suppose for example that $(\alpha x)^n$ should be halved and $(\beta x)^m$ doubled (simulation of a move to the left), and that a halve occurs through the path 00012211 of shifts, with $v = 3$. Then, the length of $A_{t,s}x$ and the shift $A_{t,s}x$ is entered with, determines the shift $(\alpha x)^n$ is entered with. To be more precise, the shift induced by a string $Ax_{t,s}$ is always equal to the additive complement of $(l_{A_{t,s}x} - s) \bmod v$, where s is the shift $A_{t,s}x$ is entered with. The same goes for any $B_{t,s}x$. Now given e.g. the path 00012211, the strings produced from $A_{t,s}x$, taking into account the shifts they are entered with as well as their lengths, should be thus that they give rise to this same path. Furthermore, after the simulation of an operation – in state t, scanning symbol s – is completed, $A_{t,s}x$ and $B_{t,s}x$ should be transformed into $A_{t',s'}x$ and $B_{t',s'}x$ such that each of these new strings in their turn give rise to the right paths of shifts.

Given the operations of a universal tag system using Minsky's encoding, these shift controlling strings, are then also self-reproducing in a specific way. Indeed, in simulating a universal tag system it is clear that the encoding of e.g. the symbol $A_{1,0}$ should be such that after a certain number of different transformations, the symbol $A_{0,1}$ is produced again after it has led to the encoding of a number of other symbols $A_{i,j}$. What one thus actually needs is one basic binary string for encoding $A_{1,0}$ and one for encoding $B_{1,0}$, such that each of them can be transformed in the right symbols $A_{i,j}$ and $B_{i,j}$, giving rise to the right sequences of shifts at the right time. This is most probably not a problem in itself since, in applying the method used for producing table 9.19, it was observed that there is always a wide variety of different paths of different lengths that, starting from a given string, will in the end produce the same string.

The main problem is that the several paths necessary to go from e.g. $A_{1,0}$ to $A_{1,0}$ are in their turn determined by the shifts induced by the sequences of encodings of symbols βx. The same goes for $B_{1,0}$. Indeed, in a way, the shifts induced by the encodings of $A_{s,t}$, $B_{s,t}$, αx and βx have to be perfectly synchronized at every step, where the problem of finding the right encoding for $A_{1,0}$ and $B_{1,0}$, heavily depends on the several possible shifts resulting from the productions from the sequences of αx and βx. If we would not be able to find paths going from periodic structures PP to P or $PPPP$, such that any string resulting from these structures is divisible by v, we are confronted with a very basic problem. Suppose that $v = 3$ and that at a given time the path going from $(111000)^2$ to 111000 results in the production of a string of length 32. Then, clearly, the shift $A_{t,s}x$ or $B_{t,s}x$ is entered with is not under control, it heavily depending on whether the number of strings n of length 32 is divisible by 3 or not, irrespective of whether n is odd or even. E.g. 6 times such strings will result in a shift 0, while 8 times such strings, results in a shift 2.

A possible solution to this problem is that the encoding of the $A_{t,s}x$ and $B_{s,t}$ is such that the shifts they induce remain invariant under the several possible shifts it can be entered with during the process of doubling or halving. This not only means that the paths induced by the symbols $A_{t,s}x$ and $B_{s,t}x$ leading to the doubling or halving operation should remain invariant under different shifts in simulating a certain operation of a Turing machine in state t, scanning symbol s, but also that any other symbol $A_{t',s'}x$ and $B_{t',s'}$ produced from $A_{t,s}x$ and $B_{t,s}$ should have this property. Finding such encodings might be very hard, and the problem becomes even harder when taking into account that $A_{t',s'}x$ and $B_{t',s'}$ resulting from $A_{t,s}x$ and $B_{t,s}$ depends on whether the sequence of αx (or βx) has become even or odd. Before further discussing this problem, let us first look at a possible solution to the problem for recognizing the parity of sequences of αx (or βx).

There is one clear solution to the problem of recognizing the parity of the number of αx (or βx). Suppose that it would be possible to halve any sequence of $(111000)^{2n}$ in the appropriate way. Then, a solution would follow if the path leading from $(111000)^2$ to 111000 when applied to 111000, not followed by another 111000, leads to a string S of length not divisible by v. Then, if n even, once $(111000)^{n/2}$ is produced, the string representing $Bx_{t,s}$ (or $A_{t,s}x$) would be entered with a fixed shift, leading to $Bx_{t'_0,s'_0}$ (or $A_{t'_0,s'_0}x$). If n odd, $Bx_{t,s}$ (or $A_{t,s}x$) would be entered with another shift, such that $Bx_{t'_1,s'_1}$ (or $A_{t'_1,s'_1}x$) is produced, by using S.

The information on the parity of a sequence of αx (or βx) however, must also be transferred to the next $A_{t,s}x$ (or $Bx_{t,s}$). Indeed, given the parity of αx this information can be directly transferred to $B_{t,s}x$ depending on the shift. This shift however, will also be transferred to $A_{t,s}$, in passing through the sequence of symbols βx following $B_{t,s}x$ (since the encoding of any βx is of length divisible by v). In other words, depending on the parity of the number of αx, the sequence of βx will be entered with a different shift and one must thus find a way for the βx to remain invariant under these two possible shifts. This is not necessary a problem. For example, given the string resulting from the structure 111000. If entered with a shift 1, the structure reproduces itself. When entered with a shift 0, it result in a structure 110100, which is clearly not a period 6 structure. If entered with a shift 2 though, we get 011100, which is again a period 6 structure. Given 011100, one will then need other paths to halve or double $(011100)^2$.

In general, there are three different solutions to the problem of keeping the encodings of the αx and βx invariant under certain different shifts. First of all, it might be possible that the string resulting from a given periodic structure, can e.g. be entered with two different shifts but will still give rise to that selfsame structure. If this is not the case, but the resulting strings are of the same length and are still periodic, but with different structure, there are two possible solutions. Let us return to the example of Post's tag system and the structures 111000 and 011100. Then a solution follows if the next $B_{t'_1,s'_1}$ produced is such that it gives rise to the right paths for doubling or halving $(011100)^2$ instead of $(111000)^2$. This solution might give some advantages for the encoding of the $A_{t,s}x$ and $B_{t,s}x$, since it allows for a greater variety of different paths, induced by the encodings of these symbols. Another solution would follow if it is possible to go back from 011100 to 111000 in one shift. This is not the case for this specific example, but it e.g. works for 011100 and 001110. Indeed, if the periodic string with structure 011100 is entered with a shift 0 it results in a string with structure 011100. When entered with shift 1 it results in 001110. Then, if the string with structure 001110 is entered with a shift 2 it again results in the periodic string with structure 011100.

Now, returning to the problem of finding the right encoding for $A_{t,s}$ and $B_{t,s}$. If one of the strings produced in going from structures PP to P or $PPPP$ is not divisible by v, we saw that we cannot control the shift the symbols $A_{t,s}$ or $B_{t,s}$ are entered with, and we should thus find encodings for these symbols such that the shifts they induce remain invariant whatever shift they are entered with, when this situation occurs. In taking into account our possible solution of recognizing whether a given sequence of αx or βx is odd, together with the fact that not all the strings produced in the path going from PP to P or $PPPP$ are divisible by v, the encoding of $A_{t,s}$ and $B_{t,s}$ should be thus that the shifts they induce are at certain points invariant under the shifts they are entered with, and at other times not. Any such encoding might be very difficult to find. Although its possibility is not at all excluded here, it might be more effective to search for periodic structures PP, such that any string produced in the path from going to PP to P or $PPPP$ is of length divisible by v. Then, given the encoding of a string $A_{t,s}x(\alpha x)^n B_{t,s}x(\beta x)^m$ the shift any symbol $A_{t,s}x$ rsp. $B_{t,s}x$ is entered with, is completely determined by the shift induced by the shift the strings $(\beta x)^m$ rsp. $(\alpha x)^n$ were entered with through the previous $B_{t,s}$ rsp. $A_{t,s}$.

There are two more problems left to solve, before this encoding scheme is complete: we must find a way to simulate a printing operation and we must be able to encode the halting state. There are two possible solutions for the encoding of a halting state. One can interpret a halt as the production of a very specific string that is periodic, or one can understand it as a tag system producing the empty string. In case one wants to produce a very specific periodic string, to encode the halting state, one can use the periodicity of the strings αx and βx. Since these strings are already periodic, only $A_{t,s}$ and $B_{t,s}$ must become periodic. In other words, if a Turing machine halts in state t scanning s, the symbols $A_{t,s}$ and $B_{t,s}$ should be thus that they produce periodic strings $A_{t',s'}$ and $B_{t',s'}$, such that the shifts induced by $A_{t,s}$ and $B_{t,s}$ are such that once $A_{t',s'}$ and

$B_{t',s'}$ are produced, the αx and βx are each of the structure P. Furthermore, $A_{t',s'}$ and $B_{t',s'}$ should have periodic structures of the same type as P, having length divisible by v, their lengths inducing the right shift for the αx and βx to remain periodic. Then, once the string $A_{t',s'}(\alpha x)^n B_{t',s'}(\beta x)^m$ is produced, the tag system will always produces the same sequences of strings, which can then be interpreted as the encoding of the halting symbol.

If a halting state in a tag system is interpreted as the tag system producing the empty string, one must find encodings for $A_{t,s}$ and $B_{t,s}$ that leads to a sequence of shifts such that $A_{t,s}, B_{t,s}, \alpha x, \beta x$ result in the empty string. As is clear from table 9.19, it is possible in Post's tag system to find a path going from $(111000)^2$ to 0^7, a string that finally results in the empty string. One of the problems involved here is that there exist tag systems that never halt, as is e.g. the case for **T2**. In this respect it might be more effective to identify a halting state with a specific periodic string.

We now still have to find a method for simulating a printing operation. As was said before, the symbols $A_{t,s}$ and $B_{t,s}$ should not only be used as shift controlling devices, but must furthermore be able to simulate the operation of printing a symbol. In this respect, we will split each of the strings $A_{t,s}$ and $B_{t,s}$ into two substrings $C_{A,t,s}$ and $D_{A,t,s}$, rsp. $C_{B,t,s}$ and $D_{B,t,s}$ where $C_{A,t,s}$ (or $C_{B,t,s}$) can be the empty string and is only used, when combined with $D_{A,t,s}$ (or $D_{B,t,s}$) to result in the right shifts for the doubling or halving operation. From now on, we will only consider the scheme for encoding the string $C_{A,t,s}$ and $D_{A,t,s}$, the scheme for encoding $C_{B,t,s}$ and $D_{B,t,s}$ being similar. Any binary string $C_{A_{t,s}}$ should be such that there exist different paths, through which $C_{A_{t,s}}$ can produce either another string of the form $C_{A',t',s'}$ or $C_{A'',t'',s''}P$. In Post's tag system it is indeed possible to construct such sequences of strings. For example, the string 0110110 can lead to 0110101010 or 0110110**111000**, where 0110101010 in its turn can lead to the production of 0110101010**111000**. What we need for the printing operation to work is a subclass of strings $C_{A,t,s}$ for which $C_{A,t,s}$ can produce either $C_{A,t',s'}$ or $C_{A,t',s'}P$, through different paths. In general, it seems to be possible to find a tag system for which there exists a finite set of strings $C_{A,t,s}$ such that in simulating a Turing machine, started in state 1, scanning a 0, there exist several paths starting from strings of type $C_{A,1,0}$ that lead to other strings $C_{A,t,s}$ of this type, of which some lead back to $C_{A,1,0}$ (where necessary for the simulation). Again however, we are confronted with the problem that although it might be possible to find such encodings, they have to fit in the general encoding scheme. First of all, it should be noted that independent of whether a 1 or a 0 has to be printed, the sequence of productions going from $D_{A,t,s}C_{A,t,s}$ to $D_{A',t',s'}C_{A',t',s'}$ must always lead to the necessary sequence of shifts to double.[49] Then $C_{A,t,s}$ might be encoded as a string that is transformed into $C_{A',t',s'}P$ or $C_{A',t',s'}$ in the same number of steps as the sequence $(\alpha x)^n$ is changed to $(\alpha x)^{2n}$. In this respect $D_{A,t,s}$ could be used to control the shifts $C_{A,t,s}$ should be entered with to produce either $C_{A',t',s'}P$ or $C_{A',t',s'}$, while the lengths of the strings produced from $C_{A,t,s}$ leading to $C_{A',t',s'}P$ or $C_{A',t',s'}$, taking into account the shifts induced by $D_{A,t,s}$, should be such that the sequence $(\alpha x)^n$ is changed to $(\alpha x)^{2n}$.

We now have a very theoretical encoding scheme that might be used to construct a universal tag system with $\mu = 2$ by simulating a universal tag system. More research however is needed to make this scheme more convincing and more detailed.

§3. Discussion If it would be possible to find a tag system T with $\mu = 2$, for which there exist binary strings $D_{A,s,t}, C_{A,s,t}, D_{B,s,t}, C_{B,s,t}, \alpha x, \beta x$, for each of the strings $A_{t,s}x, B_{t,s}x, \alpha_{t,s}x, \beta_{t,s}x$ of a universal tag system T_U using Minsky's encoding, such that if T_U produces $A_{t',s'}x(\alpha' x)^{n'} B_{t',s'}x(\beta' x)^{m'}$ from $A_{t,s}x(\alpha x)^n B_{t,s}x$ $(\beta x)^m$ T produces $D_{A',t',s'}C_{A',t',s'}(\alpha x)^{n'} D_{B',t',s'}C_{B',t',s'}(\beta x)^{m'}$ from $D_{A,t,s}C_{A,t,s}(\alpha x)^n$ $D_{B,t,s}C_{B,t,s}(\beta x)^m$ we would have proven that there exists a universal tag system T for which $\mu = 2$.

The encoding scheme proposed here, however, is very theoretical and as long as no universal tag system is found that is encoded by it, we cannot be sure whether it is a valid scheme. The hardest problem with this encoding is not the individual encoding of the symbols, but rather the synchronization of all the shifts induced by each of these encodings. As far as each encoding in itself is concerned there seem to be no fundamental problems. As was shown, it is perfectly possible to construct a periodic string that can halve or double, given a

[49] Remember that in Minsky's encoding, the simulation of printing a 1 or a 0 takes place in the substring $Ax(\alpha x)^n$ or $Bx(\beta x)^m$ for which the doubling operation has to be performed.

certain sequence of shifts. The problem is that we have to find encodings for the doubling and halving operations, such that the shifts induced by each pair of such periodic strings, are perfectly adapted to each other. In supposing this would be possible – and we are convinced that there exist tag systems with $\mu = 2$ for which such periodic strings exist, using one of the two solutions described for this problem – we are confronted with the problem of finding encodings for $A_{t,s}$ and $B_{t,s}$ such that the shifts necessary to perform this doubling and halving operation can be controlled. Now, it is not difficult to construct a string, that will for one simulation cycle lead to the right sequence of shifts. The problem is that the encoding of each symbol $A_{t,s}$ and $B_{t,s}$ should result in a sequence of productions resulting in the correct $A_{t,s}$ and $B_{t,s}$, where, again, all shifts of each individual, encoded symbol should be perfectly synchronized with the others. $A_{t,s}$ and $B_{t,s}$ do not only control the shift, but they should also be encoded such that they can produce the periodic strings used for the encoding of αx and βx, when a printing operation has to be simulated. While it is not a problem to construct or find such strings that are capable to do this when entered with the right shifts, it is not clear whether we can find such strings that perfectly fit into the general scheme.

In general, the hardest problem one would face in trying to effectively find a tag system through this method, is that every encoding of a symbol results in a shift that completely determines what will happen to the encoding of the symbol next to it and the question naturally poses itself what could be a good method to tackle this problem. A method might be to develop an algorithm that constructs tag systems in a very specific way, using general forms based on the method in which one (the algorithm) should then try to fit a tag system. Clearly, this algorithm should contain an automated method to determine whether a given tag system constructed is capable of producing periods of type 1. Implementing the method used to construct table 9.19, one can then further analyze the behaviour of the tag system constructed, to check whether it is suitable for the encoding proposed here and is thus universal yes or no. If one would be able to develop such program that indeed results in tag systems that are able to simulate a universal tag system, one would have a program to generate universal tag systems. But clearly, more research is needed here.

It should also be pointed out here that the abstract encoding scheme proposed here has clear similarities with Cook's proof of the universality of rule 110. It was only after we had developed this abstract encoding that we became aware of this similarity. As is the case with Cook's proof, the success or failure of this method heavily depends on the encoding of the initial condition, where the periodic structures and the shifts they give rise to must be perfectly synchronized to each other. There is however one fundamental difference between the method sketched here, and Cook's proof. If we would find a tag system with two symbols that follows the scheme, we would have a tag system that satisfies Davis's definition of universality, since we do not need the infinite repetition of certain periodic strings.

Shannon showed that there is a certain kind of interchangeability between number of states and number of symbols in Turing machines and this was further confirmed by Fig. 9.1. One could then wonder whether there also exists this kind of interchangeability between the shift number v and the number of symbols μ in tag systems, and whether e.g. method 2 would result in a tag system with $\mu = 2$ but with a very large shiftnumber v. For now there is no obvious reason why this should be the case. The encoding scheme does not exclude the possibility of tag systems using a small number of symbols and a small shift number. Furthermore, if one would take the approach that the encoding of a symbol x from a certain tag system T into a binary string – as is done for encoding n-symbolic Turing machines into 2-symbolic Turing machines – is determined by the number of symbols of T, the shift number v does not seem to play a significant role. This is affirmed by the results from experiment 6, where it was shown that there are several tag systems able to produce any binary combination of arbitrary length. The problem of course is how to control the production of such combinations, but we do not see how a larger v, and thus larger words w_0 and w_1 would make this problem less hard to solve.

Do there exist universal tag systems with two symbols? This question stood at the beginning of this research section, and has not been given a definite answer. It is clear that if any of the two methods described here could lead to a result, it will involve a combination of theoretical encoding schemes and most probably tedious analyses of the behaviour of certain tag systems, clearly involving the help of the computer.

9.4.4 Discussion on the limits of solvability and unsolvability in tag systems

In this last section we have seen that the known limits of unsolvability in tag systems are still very high, while their limits of solvability are very low. The fact that tag systems lie at the basis of many known small universal systems makes one wonder whether it is not possible to significantly lower these limits of unsolvability. Indeed, given their significance in this context it would be counterintuitive that the size of the smallest universal tag system would be gigantic as compared to the smallest known universal systems that are based on a simulation of these systems.

But why do tag systems actually lie at the basis of certain small universal systems? The only reasonable explanation I can think of is that tag systems are formally much more simple relative to e.g. Turing machines. That tag systems are formally very simple is one thing that should be clear by now, and we think Post was correct in characterizing these systems as primitive forms of mathematics. We believe that it is because of this formal simplicity that, despite the existing high limit of unsolvability in tag systems, this limit is in fact very low.

The limits of unsolvability in tag systems. Two conjectures.

Given the experimental results from the previous chapter, and the results from Sec. 9.4.1 and Sec. 9.4.3 there are indeed some very clear reasons to suppose that the limits of unsolvability in tag systems are significantly lower as compared to those in e.g. Turing machines. And although these reasons remain heuristic, they are, to our mind, very convincing.

First of all, our encoding from the $3n+1$-problem into the tag systems T_C with $\mu = 3$, $v = 2$ shows how hard it might be to prove this class of tag systems solvable. Indeed, the fact that a large number of researchers have worked on the $3n+1$-problem without having solved it, shows how hard the problems really are for this class of tag systems.

Given this reduction of the $3n+1$ problem to T_C, we would like to state the following conjecture, along the lines of the conjecture stated by Margenstern [Mar00] in this context (See Sec. 9.4.1):

Conjecture 9.4.1 *There exists at least one tag system with an unsolvable reachability problem or an unsolvable halting problem in every set of tag systems for which $\mu > 2, v = 2$*

In comparing the $3n+1$-line for Turing machines and tag systems (See Fig. 9.1 and 9.2) it is clear that T_C is considerably smaller than the size of the known Turing machines to which the $3n+1$-problem can be reduced. Furthermore, whereas the class of tag systems TS(3, 2), contains T_C, the class of Turing machines TM(3, 2) is known to be solvable. This clearly indicates that the limits of unsolvability in tag systems might indeed be lower as compared to those in Turing machines.
Since the $3n+1$-problem is one of those problems for which a large part of the research is based on computer experiments and heuristic arguments, this reduction indicates that a study that starts from an analysis of the behaviour might offer valuable new insights for this class of tag systems.
In section 9.4.3 we described two abstract methods, to construct a universal tag system with $\mu = 2$. Although we were not (yet) able to construct such a universal tag system, we argued that there are no fundamental obstacles for proving the universality for the class of tag systems with $\mu = 2$. Since we defined the size of a tag system as the product of μ with v (See Sec. 6.1.1), constructing a universal tag system with $\mu = 2$ however does not guarantee anything about the size of such tag systems and thus a lower limit of unsolvability in tag systems. There is, however, no indication that one should need a large v when the number of symbols is small. As was already mentioned, the fact that tag systems with a small shift number v seem to be capable of producing binary combinations of arbitrary length (see Experiment 6), strengthens the idea that one does not necessary need a large shift number for tag systems with only two symbols, to prove them universal. Furthermore, our encoding scheme for constructing a universal tag system with $\mu = 2$, if it could be made effective, does not exclude the idea of a universal tag system with $\mu = 2$, v relatively low. Indeed, there is no

reason to assume that one needs large shift numbers to find a tag system with the kind of periodic strings and their synchronization needed for the scheme to work.

Besides the theoretical possibility of constructing a relatively small universal tag system with $\mu = 2$, the results from the experiments from the previous chapter, add further support to the following conjecture:

Conjecture 9.4.1 *There exists at least one tag system with an unsolvable reachability problem or an unsolvable halting problem, i.e. the two forms of the problem of tag, in every set of tag systems for which* $\mu = 2, v > 2$

There is indeed heuristic support for the unsolvability of the class of tag systems $\mu = 2$, $v > 2$. Some further arguments will be given here.

First of all there is of course Post's tag system. Given my own experiences with this tag system, together with those by other researchers, one cannot but conclude that it is far from trivial to prove this tag system solvable. Although it is far from clear how to encode any computable function into this tag system, the least one can say about **T1** is that it is intractable on a practical level. One only has to consider the results on the periodic structures in this tag system to understand how fascinating it actually is. Watanabe's failure to get a grip on **T1** by starting from these periodic structures suggests that proving **T1** solvable will ask for new methods, if proving its solvability is possible at all. In general, the rich variety of the kind of periodic structures found in the different tag systems, serves as an indication of their complexity, understood intuitively. It would in fact be interesting to study the class TS(2,2) in this context. We believe, on the basis of the proof of the solvability of this class, that one will not find this same kind of variety in this class.

Contrasting the results from experiment 1 with our proof of the solvability of the class TS(2,2) indicates that there is a clear difference between this solvable class and the class $\mu = 2$, $v > 2$. While any initial condition for any tag system from the class TS(2,2) can be proven to lead to one of the three classes of behaviour *after a very small bounded number of iterations*,[50] determined by the length of the initial condition, it is clear that for any of the 52 tag systems tested,

[50] This is implied by the proofs of the several subcases.

there exist initial conditions for which it is unclear whether the tag system will ever lead to one of these classes, and if yes, when this will happen.

It should also be pointed out here that the results from experiment 1, and the problems connected to it, have a certain resemblance to the $3n+1$-problem, although this of course does not say anything about the unsolvability of this class of tag systems, but rather about their practical intractability. As we discussed, given the plots, we were led to the problem whether for each class of initial conditions of length l the plot will intersect the x-axis at a finite point, thus indicating that all of these tag systems will ultimately lead to a halt, periodicity or unbounded growth. On the assumption that the plots can be generalized, we concluded that this hypothetical intersection point moves exponentially fast to the right for increasing l, which implies that for any point x on the X-axis one can always find an initial condition of length l that will not have led to one of the three classes of behaviour after x iterations.

We tested numerous initial conditions for Post's tag system **T1**, and all these conditions led to periodicity or a halt. Given the plots of some of the other tag systems, one would most probably come to the same conclusion. In this respect **T1** and several of the other 51 tag systems studied, indeed seem to be closely connected to the $3n+1$-problem: although all conditions seem to ultimately converge to a halt or periodicity, there seems to be no general method available to decide this. The reason for this seems to lie in the behaviour that precedes the halt or periodicity: it is completely intractable, at least as far as my experience goes with these tag systems.

Although there is heuristic support for conjectures 9.4.1 and 9.4.1, more research is needed here. As long as we do not have any rigorous proof of the unsolvability of this class of tag systems, we cannot be sure about the truth of the conjectures. Still, the several "structural" and "heuristic" differences between the solvable class of tag systems TS(2, 2) with this class, shows that even if this class would be solvable, we will probably need new and more advanced techniques to prove this. We believe it could be interesting to further investigate this problem by combining several different approaches.

Some possible approaches to further study small tag systems.

There are several possible approaches for further study of small tag systems, and we will point out some of them here.

First of all, more research is needed on the methods described in Sec. 9.4.3 to encode a universal tag system in a two-symbolic tag system. A starting point would be to develop a computer program that can help to study the problem of whether the encoding schemes can work for small tag systems.

Secondly, some of the experiments should be extended to a larger sample space, and new experiments should be performed. One of the things that should be tested is whether there exists some kind of rate that determines how fast a given tag system converges to a halt or periodicity, when started with initial conditions of length l.

Thirdly, we believe that the table method described in Sec. 7.3.4 might lead to interesting theoretical results. To give two examples of what kind of results it can lead to, the reader is referred to the proof of the solvability of the class TM(2,2) and the proof that Shearer's proof cannot be applied to **T2** (Sec. 8.4). This method might be used to deduce general forms of substrings that can be produced in theory by a given tag system.[51] In analyzing these forms it is perhaps possible to gain a better understanding of certain cases in the class of tag systems with $\mu = 2, \nu > 2$ or $\mu > 2, \nu = 2$.

The periodic structures as described in Experiment 2, to our mind, beg for more research. Maybe it might be possible to define classes of tag systems by differentiating between the different types of periods that can be produced within a given tag system. One of the things that should still be done here is to develop automated methods that can be used to check for a given tag system what types of periods it can produce. Furthermore, it might be interesting to connect this property of tag systems with other domains in mathematics where periods play an important role. Possibly connected to studying the periodic structures in tag systems, is a detailed study of the connection between tag systems and number theory, as pointed out by Post. This is yet another approach on tag systems that might lead to new results.

[51] See for example Table 9.15.

Another approach could be to search for more criteria or features for differentiating between solvable and unsolvable classes of tag systems, or to further refine the ones already found. For example, further research on the (non generally valid) constraint 3 might be used to make a clear distinction between the class of tag systems with $\mu = 2$, for which $v = 2$ and $v > 2$ and to seriously simplify the solvability proof of the class TS(2,2).
A last approach we want to mention here, is the approach taken by Church, Kleene and Rosser at a given point during their research on λ-calculus. To encode as many functions over the integers one can think of in very small tag systems is a challenge we really want to face.

Concluding: These approaches, only summarily indicated here, amount to a rather large research programme on tag systems.

9.5 Conclusion.

This chapter was started from the question in what way small universal systems might help us to better understand the link between the theory underlying the unsolvability proofs for several classes of system and the actual discourse of these systems. The main starting point has been the question of the significance of such small universal systems for a study of the limits of solvability and unsolvability. A basic problem in this context is that there are several classes of Turing machines that are still not known to be solvable, nor to be universal. A fundamental question to be asked here, is whether there exist between the solvable and the universal classes, classes that are neither solvable nor universal, i.e., classes containing Turing machines of a lower degree of unsolvability than the universal machine.
As we argued throughout Sections 9.1 and 9.2, although the significance of *constructing* (small) universal systems in this context should not be underestimated, a study of the behaviour of such machines offers us no extra advantage over a study of the behaviour of the machines they simulate.
A study that starts from such analyses however, can be very important in this

context. Especially when relatively small systems are involved, a study of the behaviour leads to valuable results. This was illustrated in Sec. 9.3, through some examples from the literature. We concluded there that the best way to further investigate the problem of determining limits of solvability and unsolvability, is in combining both approaches.

In the last section we studied limits of solvability and unsolvability in tag systems. Given the several results from this section, it was shown that also in case of tag systems a combined approach seems the most promising to study these limits. We argued in this section that the limit of unsolvability in tag systems is probably lower as compared to Turing machines and conjectured that this limit is in fact very low. Also here however, one has to take into account that unsolvability does not necessarily imply universality. I.e. it might well be that the smallest classes here conjectured to be unsolvable are classes of a lower degree of unsolvability, non-universal but unsolvable.[52] But, clearly, more research is needed here.

One might well ask at this point: So, why study tag systems in the *mathematical* context of computability and unsolvability, and not the better researched Turing machines? We cannot give a definite answer to this question. There is not one fundamental theoretical reason at this point to study tag systems instead of Turing machines, and in this sense, our choice for tag systems might be regarded as a more or less subjective choice. Still, we believe that the results from this and the previous chapters are in fact reason enough to study tag systems. To our mind, it is clear from these results that given their abstract character, tag systems indeed allow for a greater freedom of method and technique and in this sense can be regarded as a complementary framework that can help to gain new results in the domain of computability and unsolvability.

[52] In fact, as far as our experience goes with **T1** we are tempted to believe that this tag system might be such an example. We do not think that it is universal. Still, we have every (heuristic) reason to believe that its halting and reachability problem are unsolvable.

Chapter 10

Conclusion. Tracing Unsolvability with a special focus on tag systems.

In the introduction to this dissertation we stated that our research did not really start from one or the other specific research question. Rather it was a plain fascination with computability and unsolvability that motivated this research and has given rise to a whole sequence of research questions.

The first thing I did when I started with this research was to read some of the papers by Church, Gödel, Post and Turing. I soon decided to focus on unsolvability rather than on undecidable propositions, because it were the unsolvable problems that attracted me the most. One of the things that fascinated me here was how the theoretical results concerning unsolvability and the theses underlying them are actually connected to the discourse of the systems these results are about. Although, from a certain point of view, it is rather trivial to see how this connection functions, it is not, if one wants to understand solvability and unsolvability on a more individual level.

After having read several papers by Post, I was convinced that the problem of "tag" must have played an important role in Post's earlier research. Having ordered the 2004 republication of Davis's [Dav65b], and read the *Account of an anticipation*, I was deeply impressed by Post's earlier work. Not only because of the results themselves, but, maybe even more, because of Post's method of generalization and abstraction and his way of describing things and noticing certain fundamental problems.

In the meantime, I also got more involved with the computer. I had already planned to learn to program before I started with my research. As was explained in Sec. 8.2.1, I never found the time to learn this in any decent way but started to program with rather specific goals in mind. During that time I experimented with several formal systems, but, under the influence of Post's work, I got more and more attracted to tag systems. There are several reasons why I chose tag systems over other systems, but I guess the main reason was and still is that they were furthest removed from anything recognizable. One of the things that very much attracted me in using a computer for studying certain problems, was the element of surprise, i.e., the fact that whatever I expected from the output, there was often something I could not have anticipated. It is exactly because their abstract character, that tag systems are capable to maximize this element of surprise.

Now that I have finished this dissertation I can only conclude for myself that although I have not succeeded in everything I wanted to do, and have clearly not managed to integrate everything into a nice coherent whole, I am still quite satisfied about what I actually did. For me the combination of a more historical and philosophical part with a more theoretical part has been basic to this research. Throughout most of my research time, both sides have been in constant interaction with each other.

In the first part of this dissertation we started from the earlier work by Church, Post and, to a lesser extent Turing (Ch. 2). One of the main conclusions from this chapter has been the fact that both Church's and Post's theoretical results clearly have their roots in the use and study of their respective formalisms. Contrary to Turing, they did not start from the idea of finding a proper identification between the intuitive notion of computability and a formalism, the formalism resulting from an analysis of the intuitive notion it is supposed to capture. Rather the formalisms were already there, and it was through a study of these formalisms that they first formulated their theses. For Post, his study of tag systems, the construction of normal form and the possibility of reducing canonical form C to normal form, lay the ground for formulating his then revolutionary results. In Church's case it was his becoming more and more aware of what λ-calculus is actually capable of that first led him to his thesis.

Chapter 3 started from the 1936 papers of Church, Post and Turing. Basic here has been our discussion of their respective theses. As was shown, the arguments they each put forward and the interpretation they attached to the theses, clearly differed from each other. In this context the "direct appeal to intuition" argument has played a prominent role. Because it could not be applied to general recursiveness or λ-definability, i.e., they do not directly result from an analysis of the notions they are supposed to capture, Church and Gödel, and with them many others, were led to the conclusion that Turing's thesis is the most convenient. However, in the light of Post's and Kleene's position in this context, emphasizing the hypothetical character of any such thesis, it is clear that although the "direct appeal to intuition" argument is very important, it is only one among many arguments supporting the theses. On the basis of this more historical study we argued that, although it is important to go through an analysis as was done by Turing, but also by Post, it is equally important to take serious the argument by confluence, both on the level of the formalisms as well as on the level of the intuitive notions underlying the several theses. To realize the true power of any of the theses it is to our mind basic to not restrict one's attention to that which is intuitively appealing. By studying several formalisms, especially those that are further removed from the intuition, one can only realize how general the notion of a computation actually is.

Only some years after the publication of the 1936 papers, the world was at war. The military agencies soon realized the advantages "computability" could offer for warfare, and in this sense the war most probably had an accelerating effect on the development of the first computers. As a physical realization of "computability", the computer itself has contributed to a further generalization of computability. Nowadays, "computability" as manifested in the computer is no longer restricted to pure calculations, but has become part of our society and our way of living. It is used to communicate, to play games, to study biological systems, to win wars,... As was argued in Sec. 4.1 the computer can be regarded as a confluence of engineering and logical work. It resulted from the experiences *and* abstract thinking of both the logicians and the engineers, or the logician-engineers (like Turing and Zuse) when developing this device. In this sense, the physical realization of computability should be taken quite literal.

The computer has also given rise to new developments in the domain of computability and unsolvability itself. We showed, in discussing two different such developments, how the computer has made explicit the limits of computability in the physical world. Besides revealing physical limits of computability however, we also argued that, from its early use on, the computer was used as a means to make available the "universe of discourse" as it was called by Lehmer, of certain objects of mathematics, to an extent that was impossible before. When these objects are instances of the formalisms considered in the several theses, the computer can be used to study the formalisms it is considered to be the physical realization of. As was shown throughout part II, the computer has indeed played exactly this role in studying such formalisms, i.e., making available certain aspects of the "universe of discourse".

Drawing from our conclusions of the first part on computers, computability and unsolvability, the purpose of the second part was to explore the universe of discourse, sometimes with, sometimes without, the help of the computer, focussing on a class of systems which is known to be very far removed from the so-called intuition, i.e., Tag Systems.

As we tried to show through the second part of this dissertation, tag systems are very abstract systems and it is exactly in this respect that they have played an important role for me personally to understand the full generality of the theses as proposed by Church, Post and Turing. They are indeed rather difficult to "program". As Minsky remarked ([Min61] p. 450):

> It would be desirable to reduce the exponentiation level in this representation but the "Tag" systems seem intractable in regard to lower level manipulations. We have been unable even to find productions which can reduce the length $n > P$ of a string to $n - 1$, for arbitrary n.

Although it is actually rather straightforward to define this operation in a very small tag system,[1] the quote illustrates how far tag systems are removed from

[1] The production rules for a tag system that subtracts 1 from a given natural number, are:

the intuitive idea of computing something specific. But it is, to our mind, exactly because they are far removed from anything more intuitive, that they are particularly well-suited to show how an intuition can be very restrictive. Is it really true that tag systems are more difficult to program than Turing machines? In a way yes. However, as the simplicity of the encoding of the $3n+1$-problem shows (Sec. 9.4.1), they are actually more suitable to encode certain functions than Turing machines.

Through the several chapters of part II we have taken several different approaches to study computability and unsolvability in the context of tag systems, with a strong emphasis on a study of these notions for more concrete instances and their behaviour. In this way we hoped to further explicate the connection between the theoretical result of the general unsolvability of a class of systems and the actual discourse of these systems.

In Chapter 6 we showed how the solvability of the halting and reachability problem for a given tag system depends on the possibility of determining for that tag system and any given initial condition that it will converge to one of the three general classes of behaviour (halting, periodicity and unbounded growth) in a finite number of steps. We illustrated through an example how such solvability proof might proceed, and were able, on the basis of the example, to prove that the decision problem for any tag system for which the lengths of the words and v are not relative prime depends on the decision problem of a certain number (the G.C.D. of these lengths and v) of other tag systems.

Chapter 7 considered the possibility of determining certain heuristic, theoretical and conjectural criteria to select tag systems the behaviour of which is

$v = 2$, $1 \rightarrow 11$, $h \rightarrow ss$, $s \rightarrow \epsilon$. Then given a number x, starting the tag system with $h(1)^x$, results in 1^{x-1} after application of the production rules. There is even a more trivial way to encode the predecessor function in a tag system, i.e. in a tag system from the class TS(2,2). Its production rules are: $v = 2$, $0 \rightarrow \epsilon$, $1 \rightarrow 11$. This tag system produces 1^{x-1} when started with $0(1)^x$. For both encodings, the result of the computation is a periodic string, and will thus be repeated ad infinitum. Of course, it is not very hard to arrange the production rules such that the tag system "halts" through the production of a halting symbol after the result of the computation is generated. In order to do so, one only has to regard s as a halting symbol in the first encoding. While encoding the predecessor function is rather straightforward, to construct a tag system that subtracts arbitrary numbers is far less trivial.

considered to be very difficult to predict. Determining decidability criteria as they were defined by Margenstern [Mar00] is to our mind a very interesting approach to get a better understanding of what kind of features on a more individual level, can mark the difference between solvability and unsolvability. Although we were not able to prove any new decidability criteria, as defined by Margenstern, we were able to provide certain heuristic means to generate "difficult" tag systems. As was argued, although constraint 3 has its merits, it needs more research and refinement in order to really use it on a theoretical level. The so-called constraint 4, and it should be noted that it is not completely correct to use the notion of constraint here, resulted in what we have called the table method. Although the idea underlying it is very simple, it is a powerful method to study tag systems. It not only played a fundamental role in the proof of the solvability of the class TS(2,2), but can also be used to establish other results. In general we believe that this table method is a good tool to study the behaviour of specific tag systems on a more theoretical level.

On the basis of these "constraints" we then developed algorithms to generate tag systems that were expected to be hard to predict. The 50 tag systems generated by the second algorithm described, as well as **T1** and **T2**, were then used in the experiments of Chapter 8. Our most theoretically appealing result from the experiments is the classification of the several tag systems according to the four different periodic types we deduced on a heuristic basis. In Sec. 9.4.3 we already considered one possible application of periods in the context of constructing small universal tag systems, and we believe that a detailed study of tag systems on the basis of these periods might lead to interesting results. Although the other five experiments did not lead us to more theoretical results, they give heuristic support for the conjecture proposed in the last chapter of this dissertation. Indeed, if anything can be concluded on the basis of these experiments, it is the fact that the class of tag systems studied through the experiments can at least be called intractable on a practical level. The fact that this experimental approach would not lead to the same results when applied to tag systems from the class TS(2,2) shows the difficulties that are involved in proving the class TS(2, 3) solvable.

The main theme of the last chapter 9 was the study of limits of solvability and

unsolvability, not only in tag systems but also in Turing machines and cellular automata. We already know that determining decidability criteria is one way to approach this problem. In this chapter, we looked at two other approaches: 1. the deliberate construction of certain systems that are universal or encode certain other functions, or, 2. a study of a given class of systems based on the behaviour of the systems included in that class. Both approaches are very useful to gain a better insight in limits of solvability and unsolvability. One of the open problems in this area is that there exist so-called in-between classes, or particular instances for which it is not known whether they are solvable or not, a problem which becomes more intricate in the light of the possibility that some of these classes might be unsolvable but not universal, and are thus of a lower degree of unsolvability than the universal Turing machine.
This was further affirmed by our study of limits of solvability and unsolvability in tag systems (Sec. 9.4). We showed that there is a very simple method for reducing the $3n+1$-problem to a very small tag system, as well as to reduce any Collatz-like function to a tag system. On the basis of this result, we provided an extra argument for the close connection between tag systems and certain aspects of number theory, a connection which needs closer inspection. We furthermore provided a proof of the solvability of the halting and reachability problem of the class of tag systems TS(2,2) and considered the possibility of constructing relatively small tag systems, with $\mu = 2$. Based on these results and those from previous chapters we proposed two conjectures stating that there are is a very low limit of unsolvability in tag systems.

Tracing unsolvability on the level of the discourse rather than on the theoretical level was and has remained our main goal. This study is to our mind not finished. We have not been able to study all the consequences of this approach. For us, there are now three challenges. First of all, we would like to develop a more coherent framework for tag systems. For now, we have the feeling that we have merely built up a kind of tool box for studying tag systems, but it has not been applied to its full extent. Secondly, we are very much attracted to Lehmer's ideas about using computers in the context of mathematics. If we will be allowed to develop a theoretical framework for tag systems, developing methods for using my computer for finding theoretical results is one challenge I want to take up. Finally, and this is very closely connected to the two previous points, I am very much fascinated by interactions between man and machine or the formalism of computability underlying it. Nowadays, one focuses very much on man, the user, as far as this interaction is concerned. If there is one thing I learned from my research, it is that in integrating the formalism itself in this interactive process, accepting that you cannot completely control it, can only lead to very fascinating results. To think about this issue on a more philosophical level is yet another challenge we want to accept.

A. Algorithm 3 for generating Tag Systems: N-ary tag systems.

The algorithm described here, generates n-symbolic tag systems, satisfying constraints 1, 2, 3 and 5. This program is thus a further generalization of algorithms 1 and 2 from chapter 7. To generate the shift number v, the same method of algorithms 1 and 2 was used. The number of symbols is determined at random, the maximal number of symbols being 6. Of course, one merely has to change a parameter in order for this algorithm to generate tag systems with $\mu > 6$.
The lengths of the respective words, are also generated at random. The largest and smallest length found in this way are rsp. identified as $l_{w_{\mathbf{max}}}$ and $l_{w_{\mathbf{min}}}$. If $l_{w_{\mathbf{min}}} \geq v$ or $l_{w_{\mathbf{max}}} \leq v$, new lengths are determined (constraint 2).
Once all the lengths and v have been fixed, the total number of times $\#a_i$ each symbol a_i is used in the words, has to be calculated in a way constraint 3 is satisfied. This is done through the following set of equations:

$$(v - l_{w_0}) \cdot \#a_0 + (v - l_{w_1}) \cdot \#a_1 + (v - l_{w_2}) \cdot \#a_2 + \ \dots \ + (v - l_{w_{\mu-1}}) \cdot \#a_{\mu-1} = 0 \tag{1}$$

$$\#a_0 + \#a_1 + \#a_2 + \ \dots \ + \#a_{\mu-1} = l_{w_0} + l_{w_1} + l_{w_2} + \ \dots \ + l_{w_{\mu-1}} \tag{2}$$

where 1 is the implementation of constraint 3. Equation 2 guarantees that the sum of all $\#a_i$ equals the sum of the lengths of all words w_i. Now, as might be clear from these equations, it is not possible to find unique solutions because there are too much variables involved. It is thus necessary to implement a kind of trial and error method in the algorithm. The number of trials is restricted in the algorithm as follows. The first equation is reordered by changing places between the term $\#a_{i_{\mathrm{Min}}} \cdot (v - l_{w_{i,\mathrm{Min}}})$ – containing the smallest positive coefficient

– and $(v - l_{w_0}) \cdot \#a_0$. After this reordering, 2 is multiplied with $(v - l_{w_{i,\mathrm{Min}}})$ and subtracted from 1. In the resulting equation, all the terms, except the first, are put to the right hand-side, resulting in:

$$\begin{aligned}\#a_1(w_{i,\mathrm{Min}} - w_1) = (l_{w_{i,\mathrm{Min}}} - v)(l_{w_0} + l_{w_1} + l_{w_2} + \ldots + l_{w_{\mu-1}}) - [(\#a_2(w_{i,\mathrm{Min}} \\ - w_2) + \ldots + \#a_0(w_{i,\mathrm{Min}} - w_0) + \ldots + \#a_{\mu-1}(w_{i,\mathrm{Min}} - w_{\mu-1})]\end{aligned} \tag{3}$$

Furthermore, all the terms, except for the first $(v - l_{w_{i,\mathrm{Min}}}) \cdot \#a_{i_{\mathrm{Min}}}$ of 1 are put to the right-hand side, resulting in:

$$\begin{aligned}(v - l_{w_{i,\mathrm{Min}}}) \cdot \#a_{i_{\mathrm{Min}}} = -((v - l_{w_2}) \cdot \#a_2 + (v - l_{w_3}) \cdot \#a_3 + \ldots + (v - l_{w_0}) \cdot \#a_0 + \\ \ldots + (v - l_{w_{\mu-1}}) \cdot \#a_{\mu-1})\end{aligned} \tag{4}$$

Now it should be clear that the maximum value of any $\#a_i$ is equal to the sum of the lengths of all words, minus $(\mu - 1)$, otherwise, it would not be possible for all $\#a_i$ to have a value greater than 0, thus resulting in a tag system with a number of symbols smaller than μ. Then, all possible combinations of positive values for $\#a_i$, bounded by $(\mu - 1)$, which are at the right-hand side of equation 3 are tried out: the algorithm tests for any such combination, whether it leads to a solution over the integers for both Eq. 1 and 2, using Eqs. 3 and 4. For any such combination for which this is the case, thus satisfying constraint 4, the algorithm creates μ words using a method similar to that used in the previous algorithm, implementing a biased random number generator based on the values $\#a_i$ found. The tag system thus created is then further tested for possible intractable behaviour through the implementation of constraint 5.

B. Plots from Experiment 1

The 52 plots showed in this appendix, resulted from experiment 1. They show how fast initial conditions lead to predictable behaviour for each of the tag systems. The plots show the number of iterations against the number of initial conditions that have not lead to predictable behaviour for a given number of iterations (10000000).

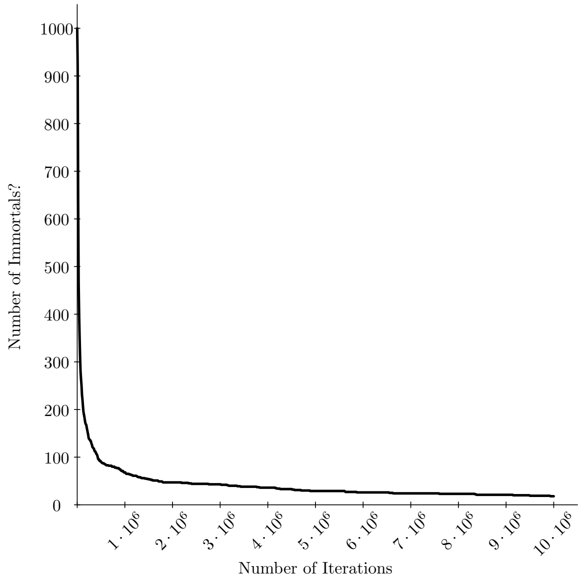


Figure 1: Plot of **T1**

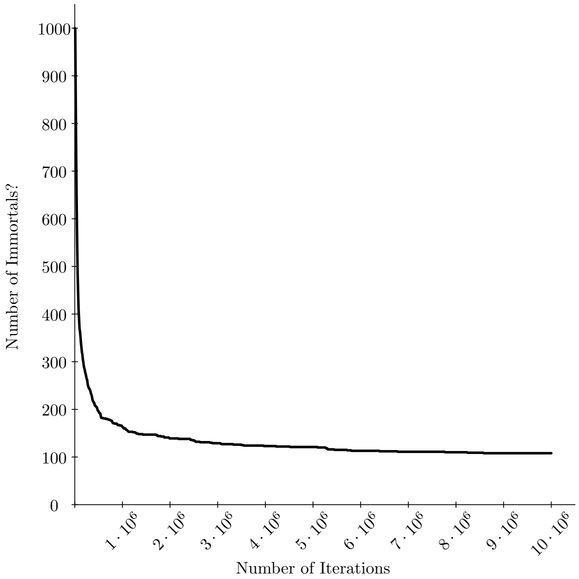


Figure 2: Plot of **T2**

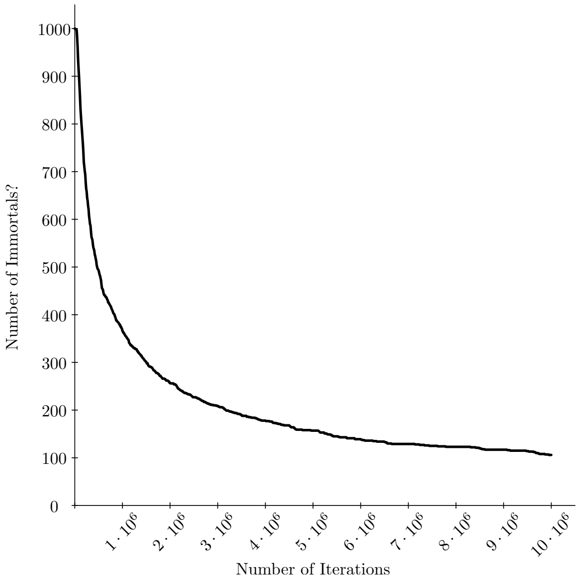


Figure 3: Plot of **T3**

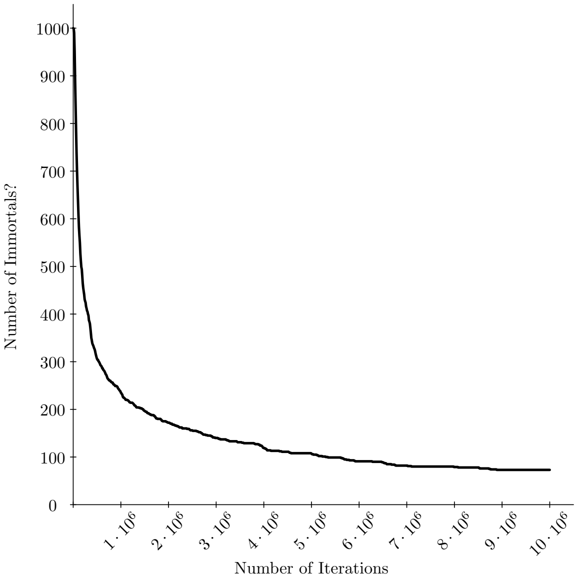


Figure 4: Plot of **T4**

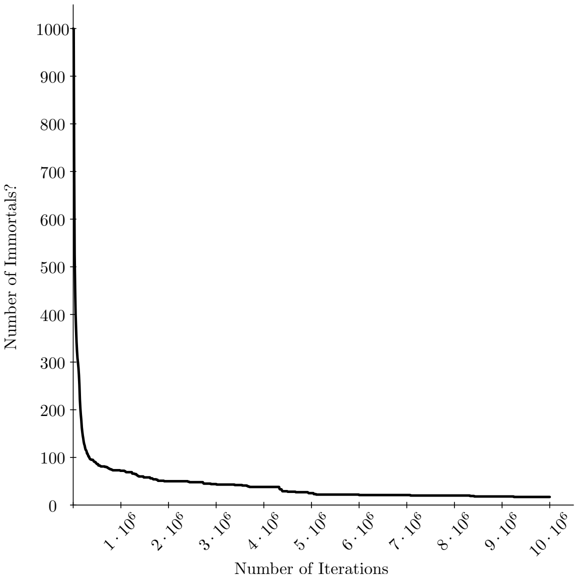


Figure 5: Plot of **T5**

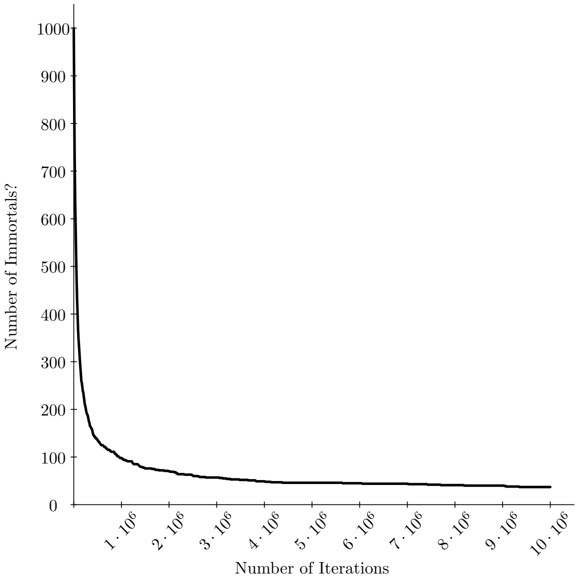


Figure 6: Plot of **T6**

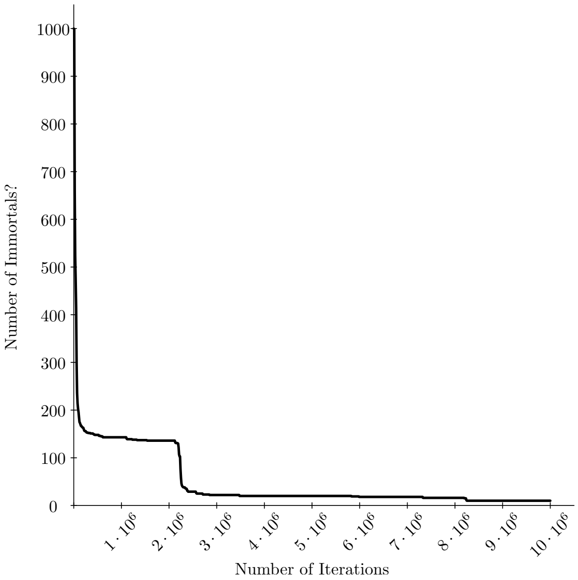


Figure 7: Plot of **T7**

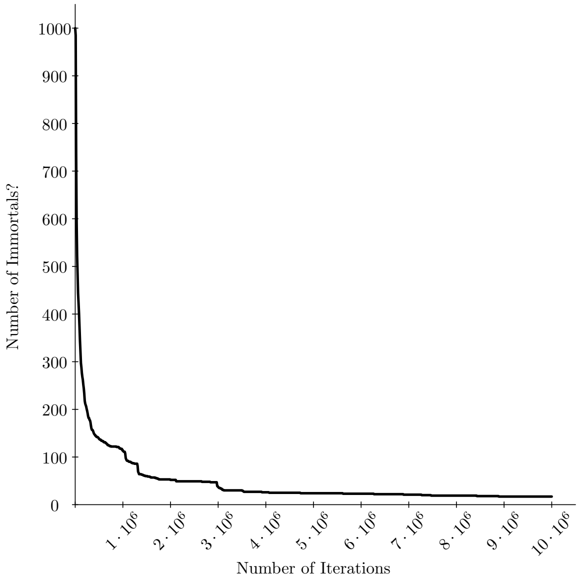


Figure 8: Plot of **T8**

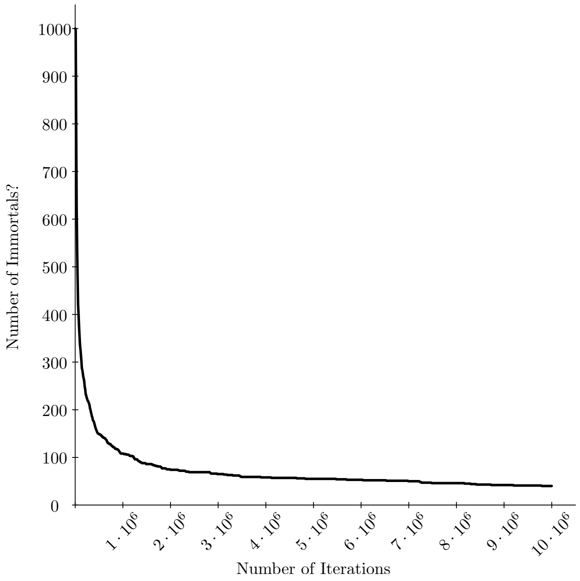


Figure 9: Plot of **T9**

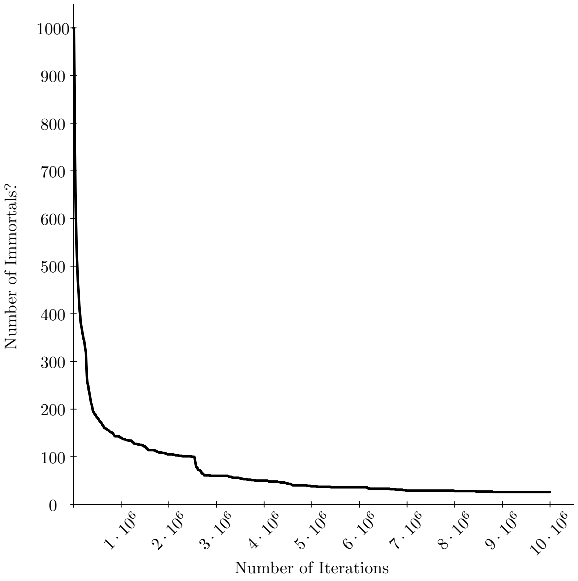


Figure 10: Plot of **T10**

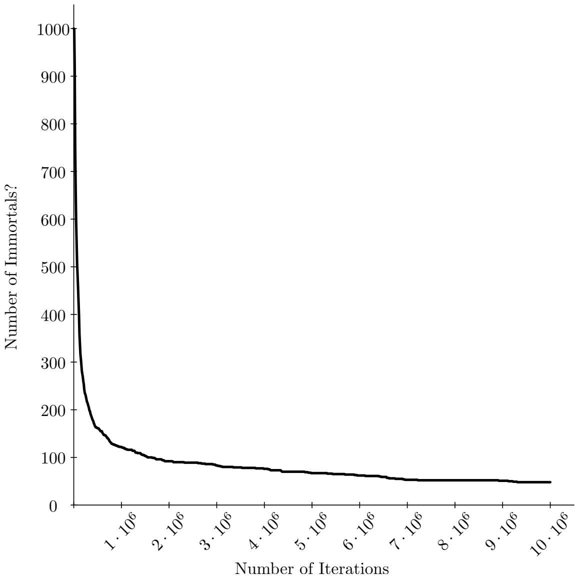


Figure 11: Plot of **T11**

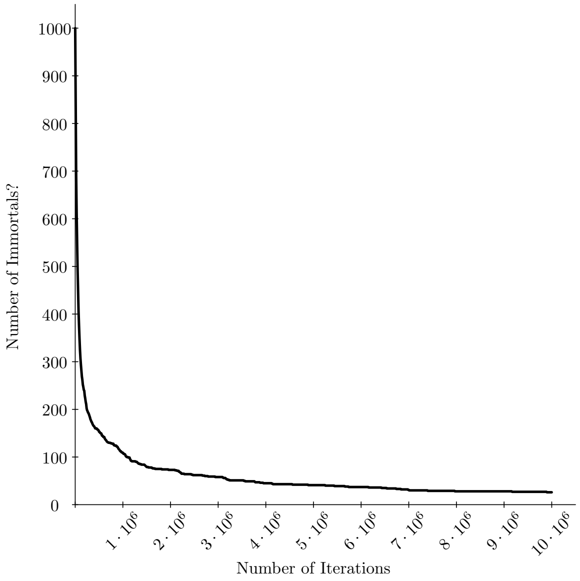


Figure 12: Plot of **T12**

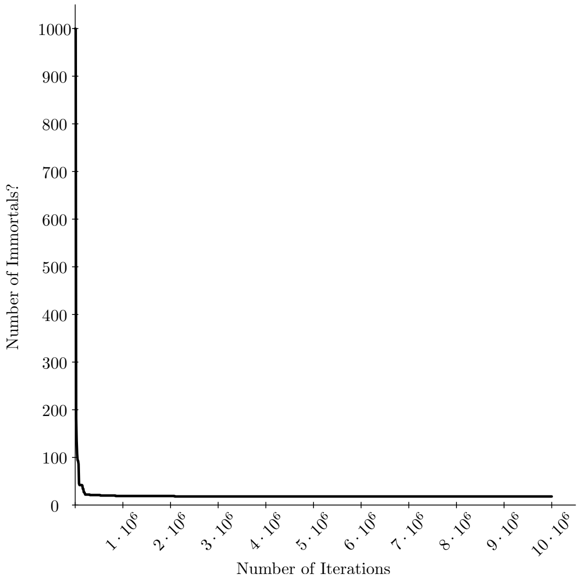


Figure 13: Plot of **T13**

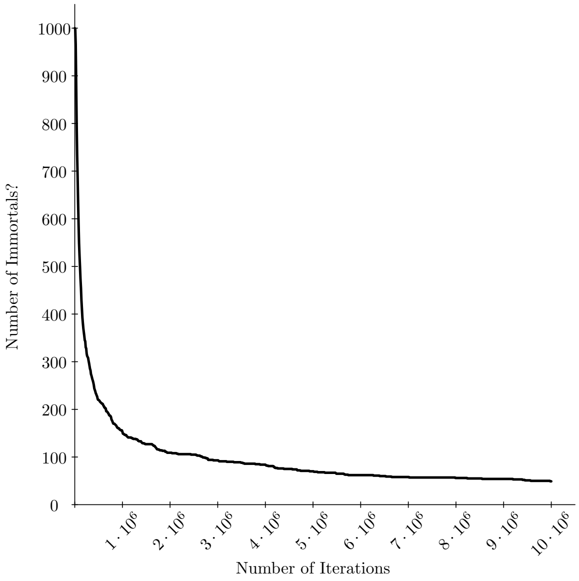


Figure 14: Plot of **T14**

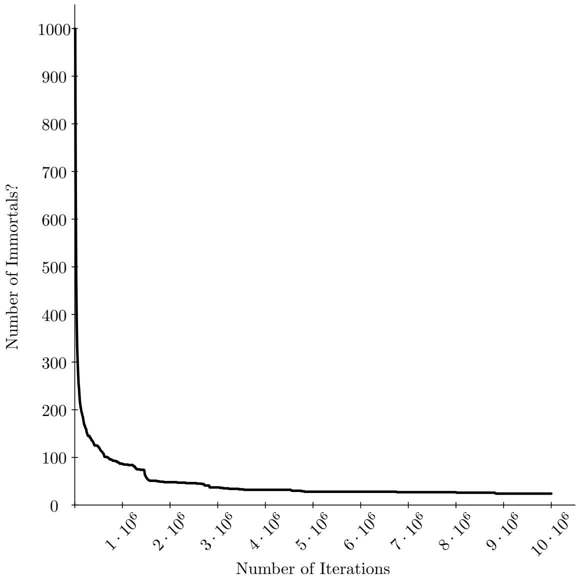


Figure 15: Plot of **T15**

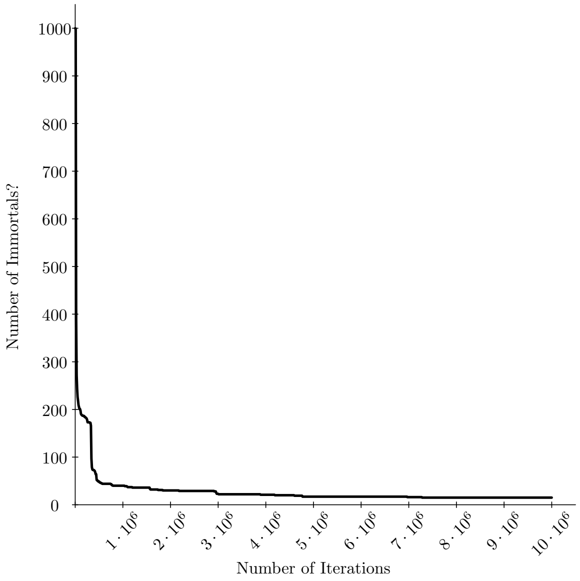


Figure 16: Plot of **T16**

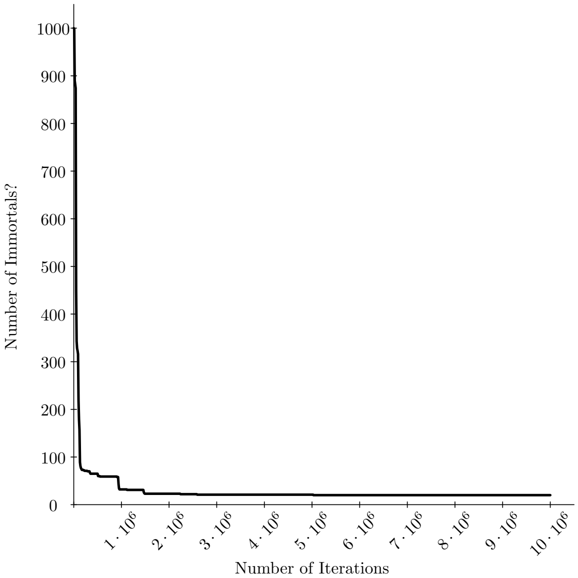


Figure 17: Plot of **T17**

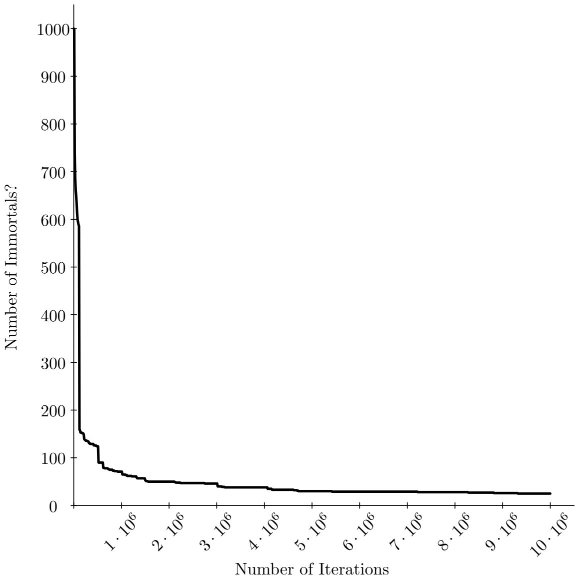


Figure 18: Plot of **T18**

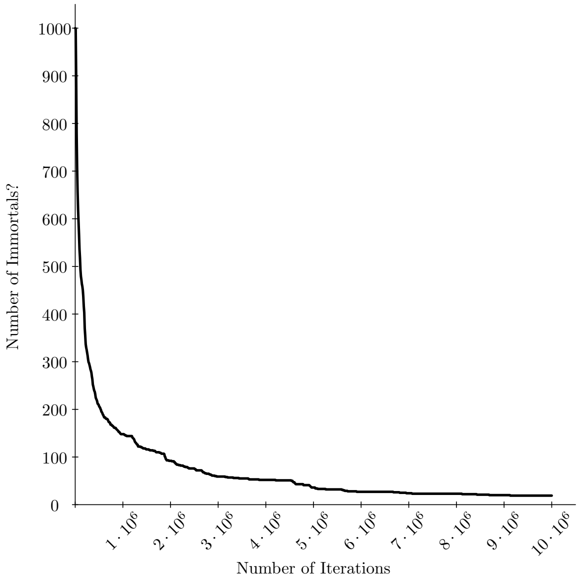


Figure 19: Plot of **T19**

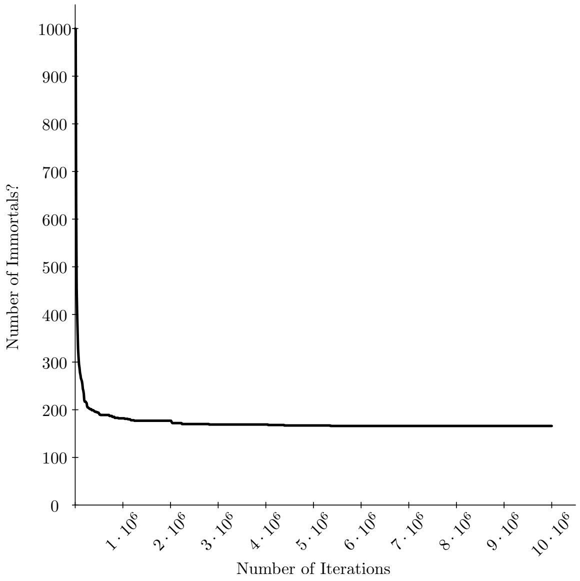


Figure 20: Plot of **T20**

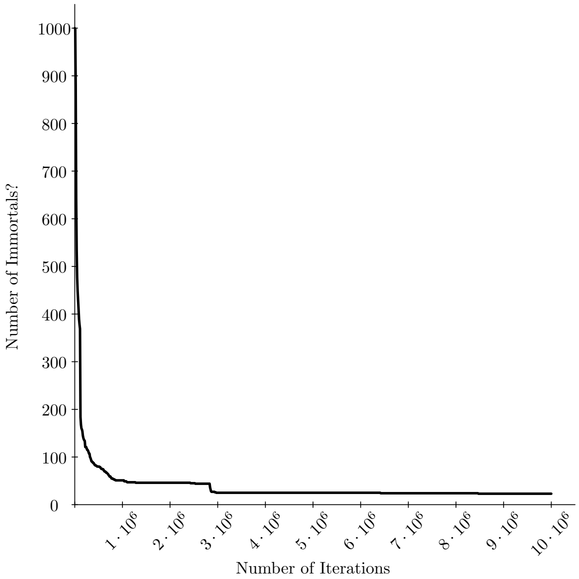


Figure 21: Plot of **T21**

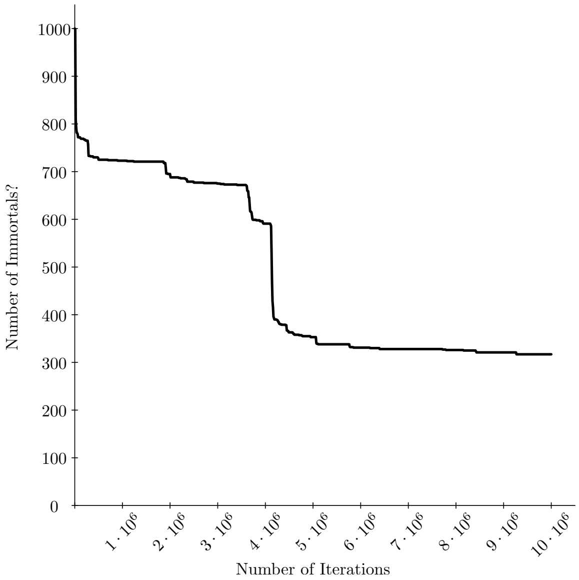


Figure 22: Plot of **T22**

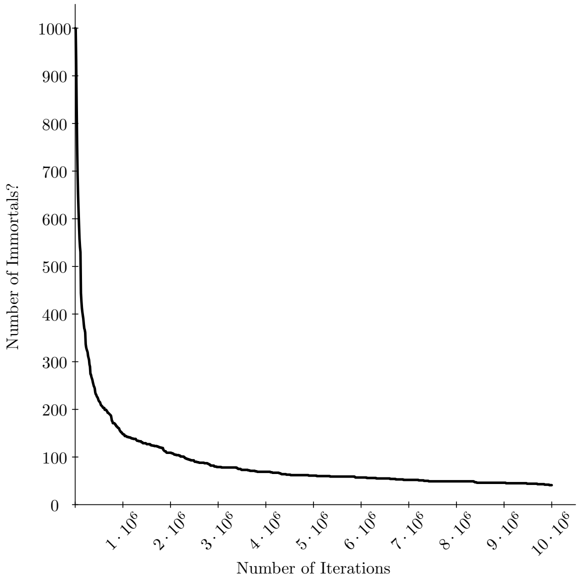


Figure 23: Plot of **T23**

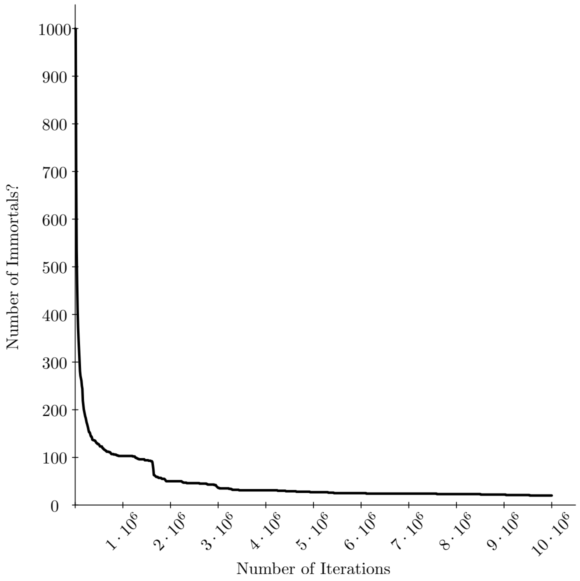


Figure 24: Plot of **T24**

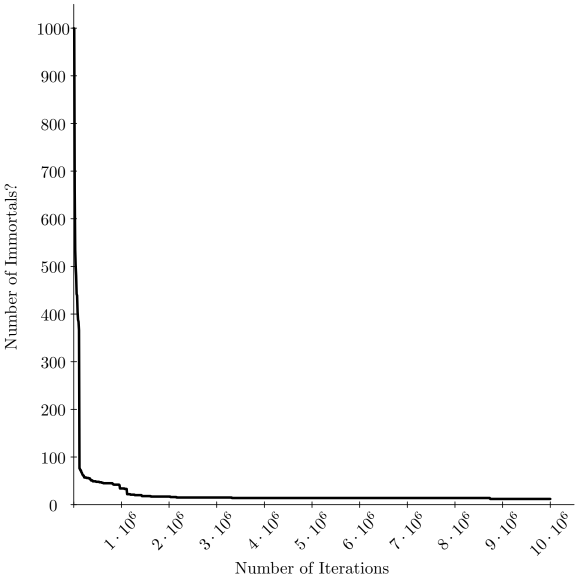


Figure 25: Plot of **T25**

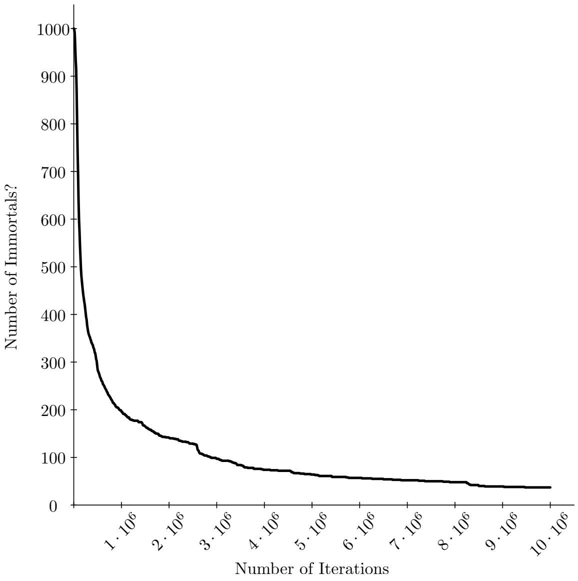


Figure 26: Plot of **T26**

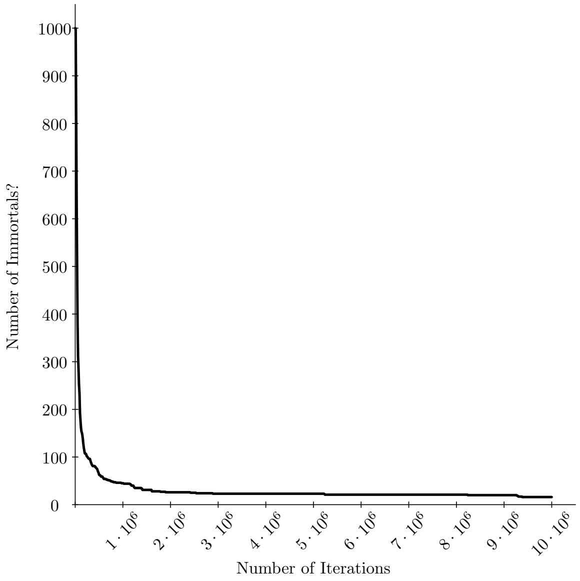


Figure 27: Plot of **T27**

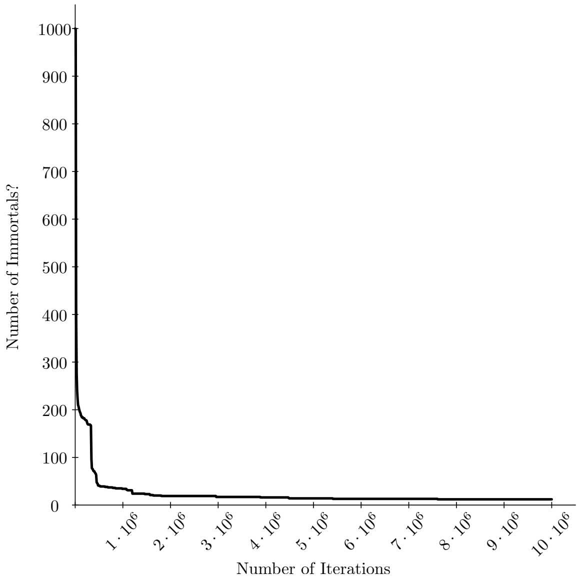


Figure 28: Plot of **T28**

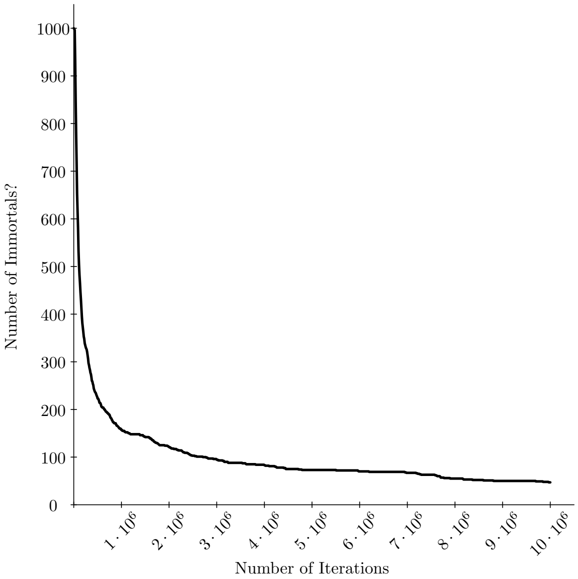


Figure 29: Plot of **T29**

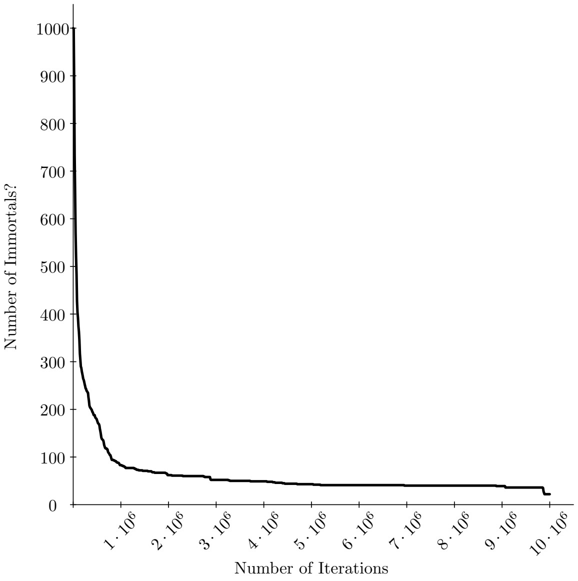


Figure 30: Plot of **T30**

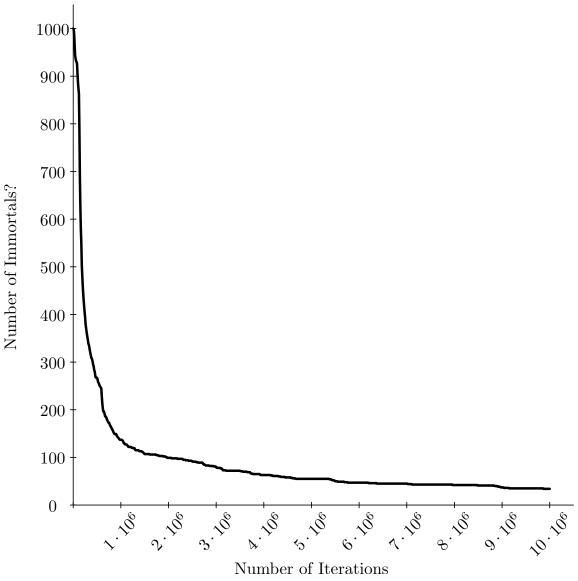


Figure 31: Plot of **T31**

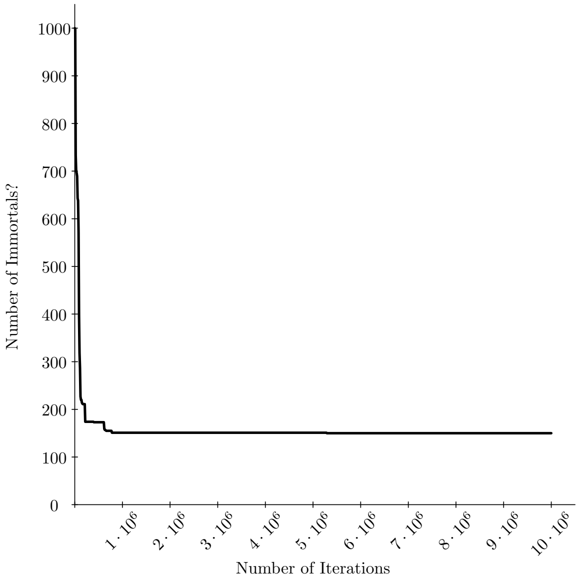


Figure 32: Plot of **T32**

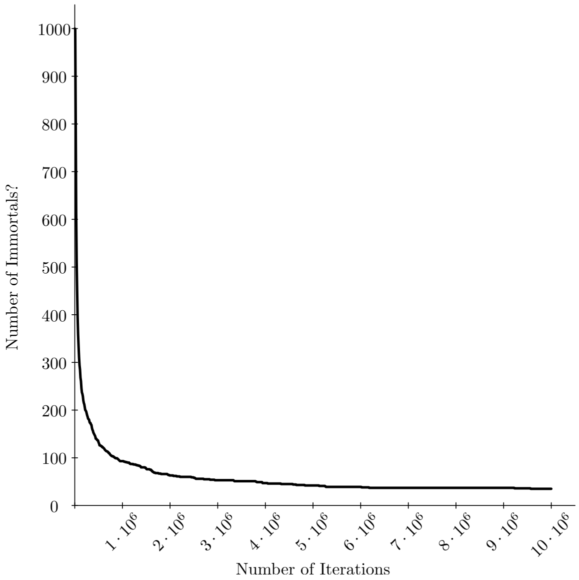


Figure 33: Plot of **T33**

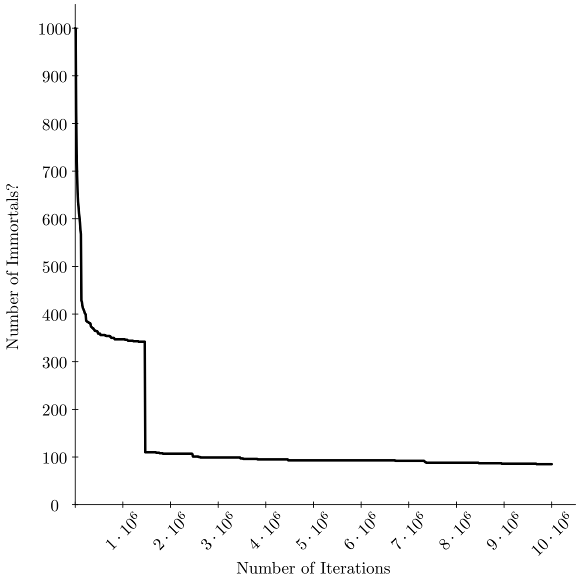


Figure 34: Plot of **T34**

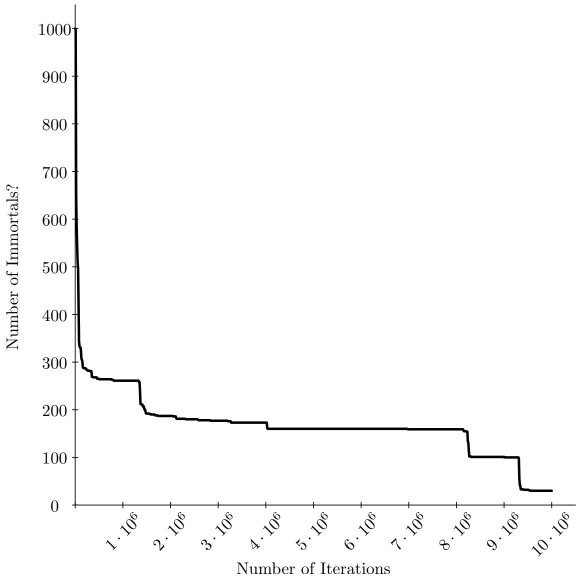


Figure 35: Plot of **T35**

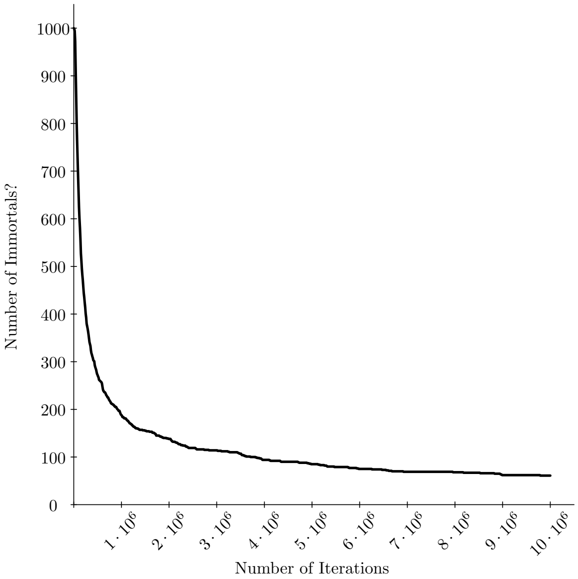


Figure 36: Plot of **T36**

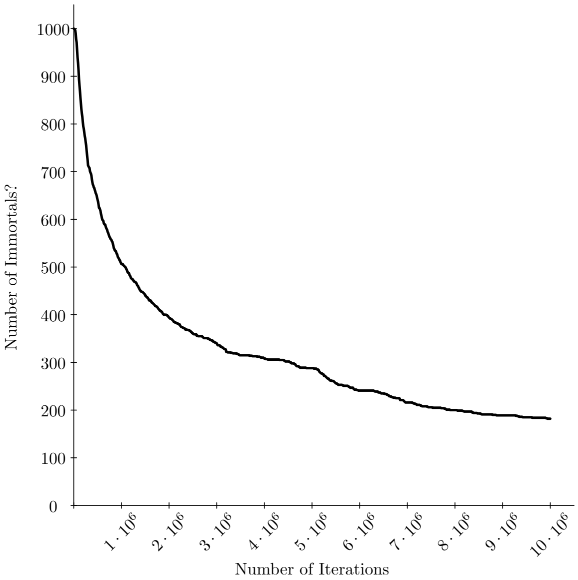


Figure 37: Plot of **T37**

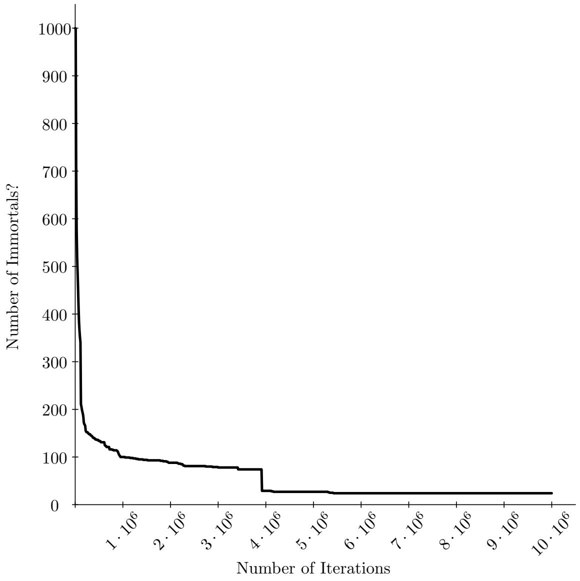


Figure 38: Plot of **T38**

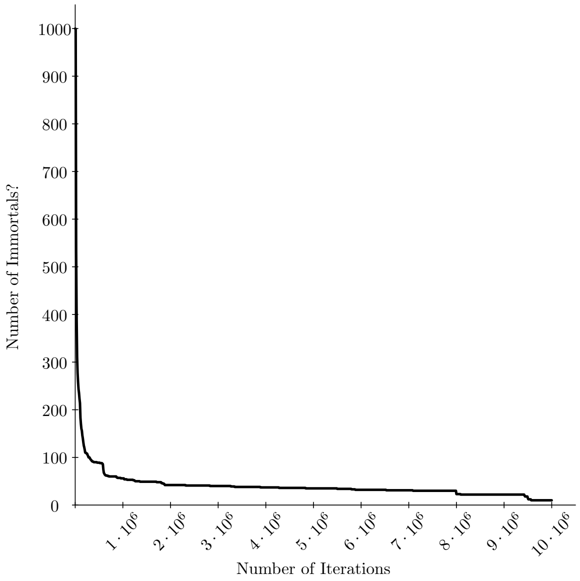


Figure 39: Plot of **T39**

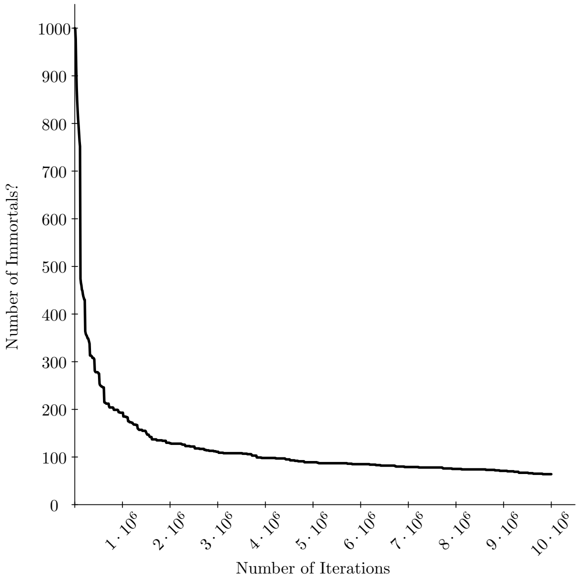


Figure 40: Plot of **T40**

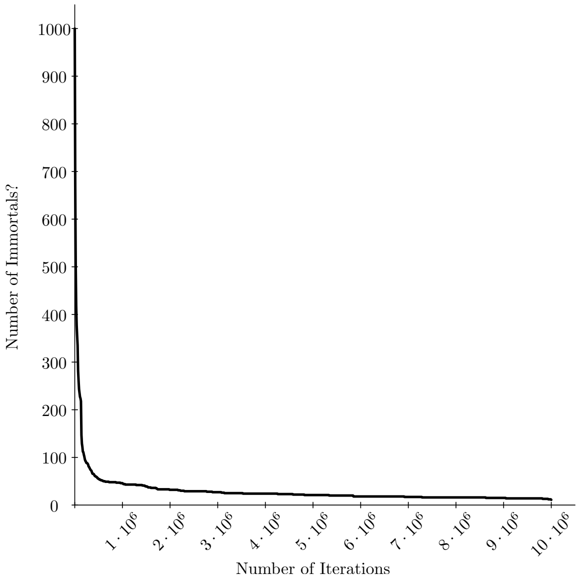


Figure 41: Plot of **T41**

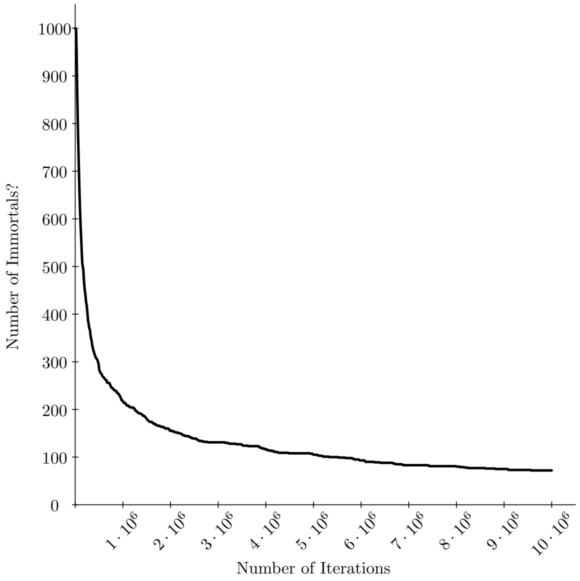


Figure 42: Plot of **T42**

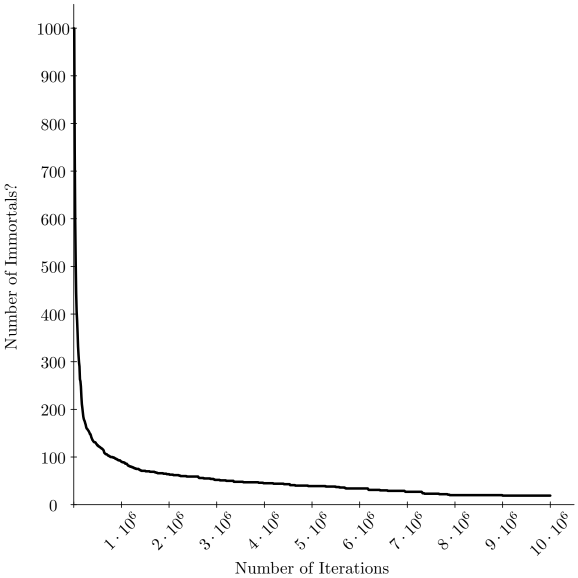


Figure 43: Plot of **T43**

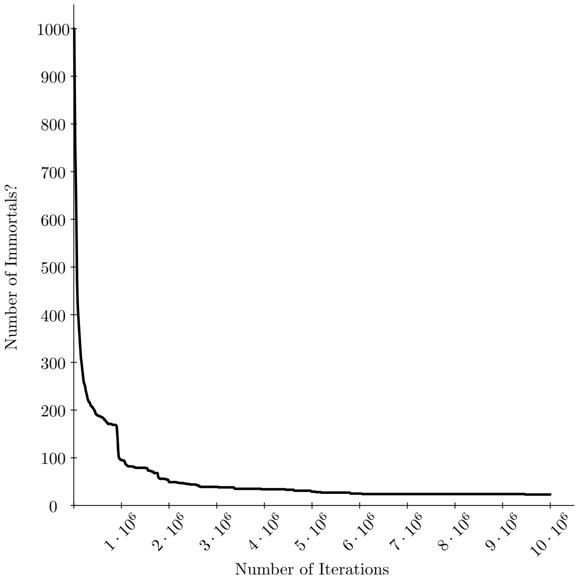


Figure 44: Plot of **T44**

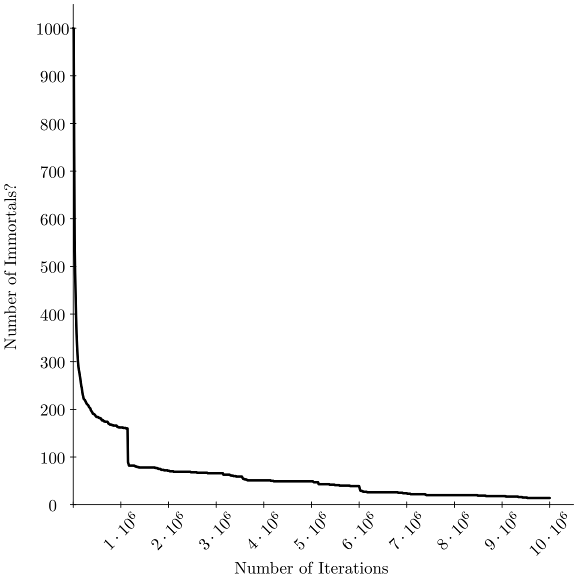


Figure 45: Plot of **T45**

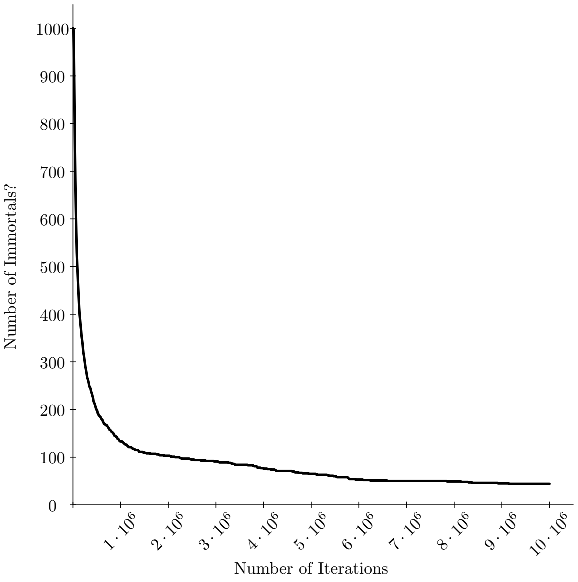


Figure 46: Plot of **T46**

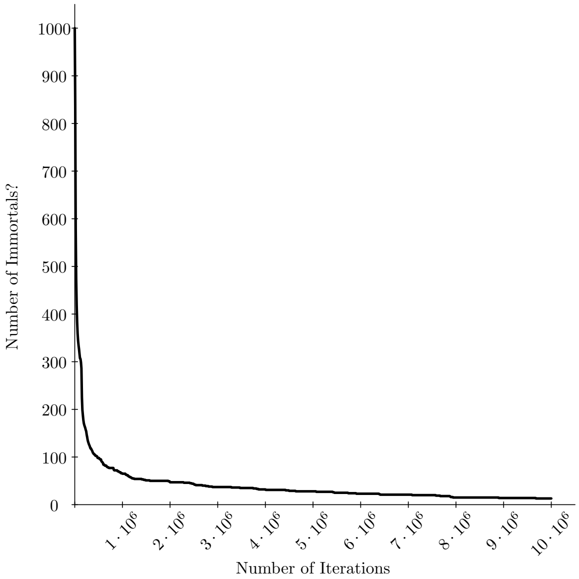


Figure 47: Plot of **T47**

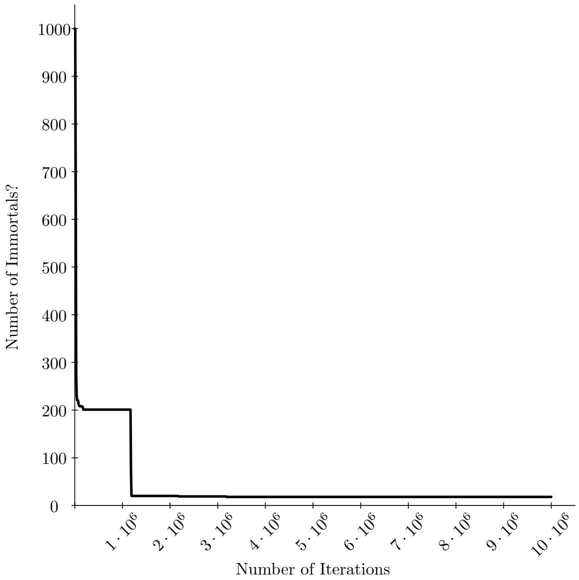


Figure 48: Plot of **T48**

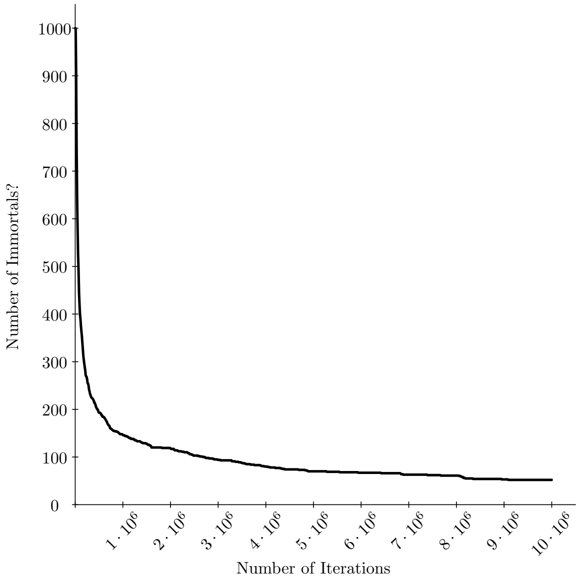


Figure 49: Plot of **T49**

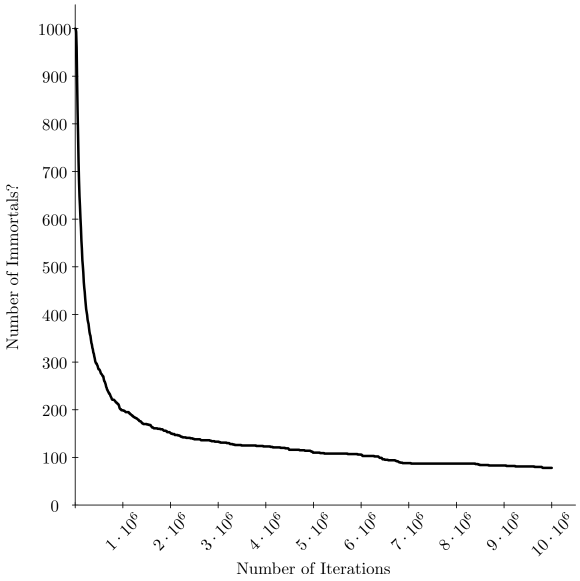


Figure 50: Plot of **T50**

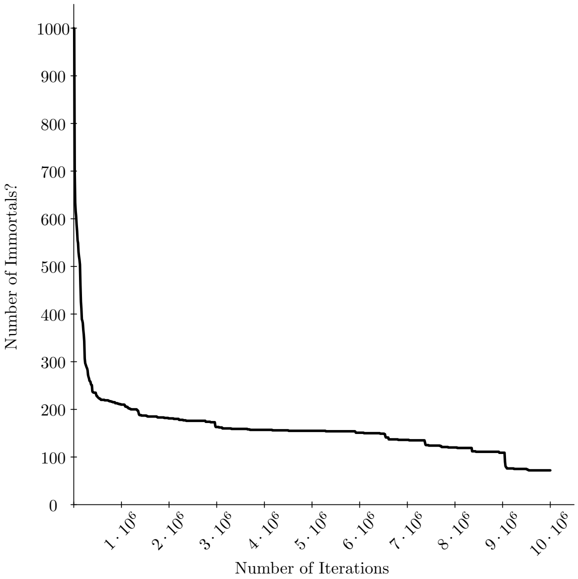


Figure 51: Plot of **T51**

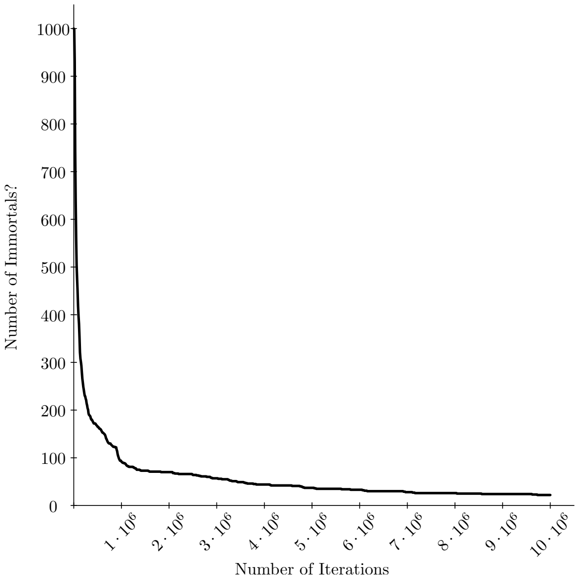


Figure 52: Plot of **T52**

C. Detailed Description of Four experiments on Tag Systems

This appendix gives a detailed description of the four experiments mentioned in Ch. 8.

Experiment 3: Sensitive dependence on Initial Conditions

In chapter 7 we discussed several features of tag systems that can mark the difference, when combined, between tag solvability and unsolvability. The purpose of this experiment is to explore how important certain of these features are for the actual behaviour of a given tag system. The features to be tested are: the shift number v, the position and number of the letters in the words and the length of the words. Although we already know that these parameters play a determining role in the predictability or non-predictability of a tag system, we do not know how significant they are in the actual execution of a tag system. I.e. the basic question to be asked here is how sensitive a tag system constructed by using the constraints from Chapter 7 is to one tiny change in one of these parameters during execution. How reasonable is it to suppose that changing, e.g., one letter completely changes the behaviour of a given tag system? There already exists a measure for finding out how sensitive a given system is to small changes. It is a measure from chaos theory and is called the Lyapunov exponent.

Tag systems and Chaos Theory In chaos theory one investigates so-called non-linear dynamical systems. While I am of course not a specialist in the subject, I did some reading on chaos theory, and the closely connected subject of fractal geometry, during the beginning of my research. One of the best introductory books I read – and which is not a mere popularization of the subject – is called *Chaos and Fractals* [PJS92], where clear definitions are given of e.g. Lyapunov exponents and several kinds of fractal dimensions. It was this book, that helped me to implement my first computer experiments in Basic and its influence on my thinking during the months following my reading of the book should not be underestimated. At a given time, given my knowledge of chaotical systems and my knowledge of universal systems and unsolvable decision problems I began to wonder as to whether one could find a relation between universal (or at least intractable) systems and chaotical systems, and I thus performed some rather basic experiments to test this for tag systems.[2] Now in [PJS92], a system is considered chaotic if:[3]

1. it has a dense set of periodic points
2. it shows sensitive dependence on initial conditions, measured through the Lyapunov exponent.
3. it shows mixing behaviour (is topologically transitive).

where the first condition is in fact implied by the last two. A periodic point is an initial condition which leads to periodicity. A dense set of periodic points means that, if the system is defined over a certain interval of initial conditions, it must be the case that for any arbitrary subinterval there is at least one periodic point. The mixing property, expressed informally, means that we can get everywhere from anywhere. This means that for any two open (arbitrary small, but with non-zero length) subintervals J and I, it is always possible to find initial

[2] Some months later I stumbled into a very interesting paper by Blondel, Delvenne et. al [BDK05] in which this connection is closer inspected. In the paper the authors approach the possible connection between chaotical behaviour and universal computational systems from a more mathematically rigorous perspective. It should be mentioned that I am indebted to Delvenne for having motivated me at a certain time during my research. I would also like to thank him here for his advice on some of the issues related to the question concerning the connection between chaotical behaviour and tag systems.

[3] This definition is actually that given by Devaney in his [Dev89]

conditions in I which, when iterated under the mapping, will eventually lead to points from interval J (and vice versa). Now, if one wants to know whether the intractability in tag systems is in some or the other way connected to chaotical behaviour one is facing a rather hard problem. First of all, proving that something is chaotical is very difficult. Besides the fact that there is still no consensus with regards to the definition of chaotical systems[4], it was shown in [dCD90] that the problem to determine for a given system whether it is chaotic or not is generally unsolvable. As a consequence one often merely has strong experimental support to conjecture a certain system chaotic. Besides this problem, trying to connect chaotic behaviour with tag systems is furthermore problematic because of the finiteness involved in tag systems. First of all, we cannot work with infinite initial conditions, while research on chaos, even for discrete systems such as CA, always involves initial conditions which are infinite. We cannot do this with tag systems: tagging a word at the end of an infinite string can at least be called a problematical. Secondly, in order for the mixing property to be valid, we seem to be in need of initial conditions that do not become periodic, halt or grow after a finite number of iterations. Still, there are some indications to connect chaotical behaviour to some of the tag systems considered here. First of all, as we already know, it can be shown for several tag systems (including **T1**, that they have periodic points, and it seems probable that there is at least one periodic point for every class of initial conditions of length l. Furthermore, as will be shown here, we can experimentally that the 52 tag systems considered here have positive Lyapunov exponents. As far as mixing is concerned, it should be noted that if one would be able to prove (or show experimentally) that certain tag systems are ergodic, one has also shown that they have the mixing property: ergodicity means that there are infinitely many points in every interval from which one can reach any other point, i.e. when can get from any point to any point (and thus also interval). Now a way to prove for discrete systems that they are ergodic, is by using the definition Shannon gives of ergodicity in his influential [Sha48], pp. 8–9. Here ergodicity is defined for discrete Markov processes. If one considers the possible transitions between the finite number of states in a graph, a discrete Markov process is ergodic if:

1. The graph does not consist of two isolated parts A and B such that it is impossible to go from junction points in A to junctions points in B.
2. The greatest common divisor of the lengths of all circuits in the graph must be one.

[4] The one given by Devaney is the most commonly used.

> Now, it seems rather straightforward to apply this definition to tag systems. In Sec. 10 an experiment will be discussed which could be used to investigate this property experimentally in tag systems, since it uses the approach of Markov processes in tag systems. We did not test the ergodicity property however, due to a general lack of time.

The Lyapunov exponent used in chaos theory measures how sensitive a given system is to its initial conditions. Informally speaking, if a system shows sensitive dependence, this means that arbitrary small differences in the initial conditions become exponentially large with the evolution of the system. To be more specific, given two initial conditions, which are arbitrarily close to each other. Now apply a certain function to both points and measure the difference between the two new values generated. Then apply the same mapping to these two new values and again measure the difference between the two newly generated values,... Now if this difference becomes larger and larger in an exponential way, one says that the map shows sensitive dependence on initial conditions. Indeed, one tiny difference becomes gigantic after some steps. It is this sensitive dependence which can be a cause of discrete transitions. Think for example about a die: Taking into account all the (physical) factors which lead to a certain result (e.g. 6), one small deviation can lead from a throw 6 to a throw 1. It is the Lyapunov exponent which can tell as how sensitive a system is, indicating how fast the differences grow. There are several ways to calculate the Lyapunov exponent but the one I used is based on the method given in [PJS92]:

$$\frac{1}{n}\sum_{k=1}^{n}\ln\left|\frac{\tilde{\mathrm{E}}_k}{\varepsilon}\right| \tag{5}$$

where n is the number of iterations performed, ε is a kind of fixed error, and $\tilde{E}_k$ is the difference between the result of iterating the map on the k-th value produced and the result of iterating the map on the k-the value + ε. Now how will we measure the Lyapunov exponent in tag systems?

Set-up of experiment 3.

In this experiment we will measure how the position of the letters in the words, the shift number v and the lengths of the words are determining factors in the behaviour of a tag system. Six different Lyapunov exponents were measured, each one describing another kind of error development. To calculate these exponents, the first ten initial conditions classified as Immortals? in experiment 1, for each of the tag systems **T1-T52**, were used. Then, for each initial condition, 10000 strings are produced using the operation $\overset{\circ}{\rightarrow}$. We used this operation, since using basic tag iterations cannot be used to measure the significance of e.g. changing the shift. Indeed, the effect of a changed shift, can only be measured after all letters from a string have been processed.

Lyapunov exponents 1 and 2

The first two exponents, measure the effect of changing one letter. For each string S_k produced by the tag system, the program creates a new string $\tilde{S}_k$ by changing a letter of S_k, at a beforehand determined position. This fixed position $p_{\mathbf{fix}}$ is calculated for each tag system individually, and is based on the value of v. More specifically:

$$p_{\mathbf{fix}} = \left(\left\lfloor \frac{|S_0|}{5 \cdot v} \right\rfloor \cdot v \right) + 1$$

In this way, one makes sure that the changed letter will be scanned by the system. Through $p_{\mathbf{fix}}$ the size of the error induced at every iteration step is restricted, in order to guarantee that it is possible for every string produced the letter at position x will be changed, i.e. all strings produced should never become shorter than x. It seemed reasonable that no string produced by the system would become shorter than $p_{\mathbf{fix}}$.[5]

On the basis of this changed letter, the fixed error ε from the formula for calculating the Lyapunov exponent, could be calculated as follows:

$$\varepsilon = 2^{-1 \cdot (\lfloor \frac{p_{\mathbf{fix}}}{v} \rfloor + 1)}$$

[5] In order to be completely sure, a subroutine was added in the algorithm to check whether it is indeed the case that no string becomes shorter than $p_{\mathbf{fix}}$. For each of the tag systems tested, it was indeed the case that the string used never became shorter than $p_{\mathbf{fix}}$.

The error induced at every step thus measures the difference between the relevant letters of S_k and $\tilde{S}_k$.

$\tilde{\mathrm{E}}_k$ in its turn is calculated by measuring the difference between the relevant letters in the strings S'_k and $\tilde{S}'_k$ produced by applying $\overset{\circ}{\rightarrow}$ to S and $\tilde{S}_k$. Instead of first converting these strings from binary to decimal and then measuring the difference between these decimal values, I choose to use the following difference measure – completely in line with the calculation of ε:

$$\tilde{E}_k = \sum_{p=0}^{\lceil \frac{\mathrm{Max}}{v} \rceil} \mathrm{abs}\Big(a_{v\cdot p+1} \cdot 2^{-(v\cdot p+1)} - \tilde{a}_{v\cdot p+1} \cdot 2^{-(v\cdot p+1)}\Big)$$

where Max is the length of the longest of the two strings, $a_{v\cdot p+1}$ and $\tilde{a}_{v\cdot p+1}$ are the values of the letters in position $vp+1$ of S'_k respectively $\tilde{S}'$, i.e. the values of the relevant letters in these two strings. If it is the case that the lengths of the strings differ, it is assumed that the letters of the shorter string in a position beyond the length of this shorter string, are all zero. Knowing $\tilde{E}_k$ and the fixed error ε it is then possible to calculate a Lyapunov exponent using the above given formula.

There is one problem involved with this kind of measurement. Suppose strings S'_k and $\tilde{S}'_k$ are produced from S_k and $\tilde{S}_k$, and, as is the case for all tag systems having a shift number $v = 3$, further suppose that $\varepsilon \approx 0.0000095367$, the letter at position 58 being changed to produce $\tilde{S}_k$ from S_k. Now, if the letter at position 58 in $\tilde{S}'_k$ is the same as that in S'_k, $\tilde{E}_k$ can never become larger than $\varepsilon \approx 0.0000095367$. Putting $\tilde{E}_k$ and ε into the formula for calculating the Lyapunov exponent, the error produced at iteration step k will thus not result in an addition but in a substraction of a given number in our calculation of the exponent. As a consequence, if there are more strings S'_k and $\tilde{S}'_k$ produced for which the letters at position 58 are the same, even if there are many differences between S'_k and $\tilde{S}'_k$, the Lyapunov exponent will be a negative number, thus leading to the conclusion that there is no sensitive dependence on initial conditions. In order to check this, a second Lyapunov exponent was measured. It is calculated following the same method as that of the first, however, S'_k and $\tilde{S}'_k$ are produced from S_k and $\tilde{S}_k$, not after one application of $\overset{\circ}{\rightarrow}$ but two. If for a given tag system, the experiment again results in a negative exponent for both measurements, the idea of the significance of the letters for the determination of the behaviour of a tag system should at least be reconsidered.

Lyapunov exponents 3 and 4

The purpose of Lyapunov exponents 3 and 4 is to measure the effect of the shift itself, i.e. how large is the impact on a tag process, if one disturbs the shift with e.g. 1? Instead of looking at the differences between the relevant letters in a string, we will here look at the difference in the lengths of the strings. One of the reasons to look at this, is that the length of a string determines the shift after one application of $\overset{\circ}{\rightarrow}$. In doing so we actually take into account two of the above mentioned features of tag systems, i.e. the shift number itself as well as the lengths of the words. Indeed, slightly changing the length of a string, can be interpreted as one change of the shift number at a given iteration step, or a change in the length of one word.
As we already know, the position at which the shift enters a string is determining for what letters will or will not be scanned, these letters in their turn determining the shift at a later time. In order to measure its effect, and thus to calculate the Lyapunov exponent, the program creates for each string S_k produced by the tag system, two new strings $\tilde{S}_{k,0}$ and $\tilde{S}_{k,1}$, the shift being changed at the beginning as well as at the end of S_k. The shift was changed at the beginning of S_k by erasing its first letter, thus resulting in $\tilde{S}_{k,0}$. The determination of the change of the shift at the end of the string depends on the length of S_k. If $S_k \equiv 0 \bmod v$ then the v-th letter from the right of S_k is erased else the $v-1$-th letter from the right is erased. This was done in order to maintain the same effect of the shift in both cases. If $S_k \equiv 0 \bmod v$, erasing the last $v-1$ letter from the right cannot have any effect, since this letter will not be scanned. If, however, the v-th letter from the right is erased, it is guaranteed that the last letter scanned in S_k and $\tilde{S}_{k,1}$ can be different in content.
The calculation of ε is very straightforward. Since we want to measure the difference in the lengths of the strings S'_k, $\tilde{S}'_{k,0}$, and $\tilde{S}'_{k,1}$ produced from rsp. S_k, $\tilde{S}_{k,0}$, and $\tilde{S}_{k,1}$ through $\overset{\circ}{\rightarrow}$, the initial error is the difference in length between S_k and rsp. $\tilde{S}_{k,0}$ and $\tilde{S}_{k,1}$ and thus always equal to 1. $\tilde{E}_{k,0}$ and $\tilde{E}_{k,1}$ are thus equal

to:

$$\tilde{E}_{k,0} = \text{Abs}(|\tilde{S}'_{k,0}| - \Big|S'_k|)$$

$$\tilde{E}_{k,1} = \text{Abs}(|\tilde{S}'_{k,1}| - |S'_k|)$$

Knowing ε, and rsp. $\tilde{E}_{k,0}$ and $\tilde{E}_{k,1}$, it is thus possible to measure the effect of a shift in the determination of the behaviour of a tag system. There is one question which should be posed here: why measure the differences in the lengths and not in the actual shifts?

"Lyapunov exponents" 5 and 6

It would indeed have been a more logical choice to measure the differences in the shifts induced by $\tilde{S}_{k,0}$ and $\tilde{S}_{k,1}$ as compared to S_k. There is however a problem involved here, related to the definition of a Lyapunov exponent. As it has been defined here, the shift with which a string is entered, determining which letters will and will not be scanned, always varies between 0 and $v-1$. Take e.g. a string of length 14 and a shift number $v = 3$. If all the relevant letters except for the last of this string have been processed, a new string will have been produced. Then if finally the last letter from the original string has been processed, the first letter of our new string will have been erased. However, the maximal number of first letters being erased in a string resulting in this way can never be larger than $v-1$. In this respect, the error induced by the shift can never become arbitrarily small or large. If one identifies the error induced as the shift it is thus impossible to measure a Lyapunov exponent given its definition, since both the error induced as well as the error resulting from it are bounded. In replacing the measurement of the shifts by the measurement of the lengths, it does become possible to at least make the error arbitrarily large, the smallest possible error being 1. While not being completely unproblematic, since we cannot make our error arbitrarily small, this still seems to be the best method to measure the effect of the shift by calculating the Lyapunov exponent. In the end, one should not forget that we are working with discrete systems, and one thus has to find alternative methods to measure sensitive dependence on initial conditions.

Notwithstanding these problems, we also included a measurement using formula 5 for calculating the exponents numbered 5 and 6. The same method was used as that for calculating exponents 3 and 4. However, instead of measuring the difference in the lengths, the differences in the shifts induced by $\tilde{S}_{k,0}$ and $\tilde{S}_{k,1}$ as compared to S_k are calculated. $\tilde{E}_{k,0}$ and $\tilde{E}_{k,0}$ are calculated as follows, using the additive complements of the remainder of the lengths after division by v:

$$\tilde{E}_{k,0} = \text{Abs}(\overline{|\tilde{S}'_{k,0}| \bmod v} - \overline{\Big|S'_k| \bmod v})$$

$$\tilde{E}_{k,1} = \text{Abs}(\overline{|\tilde{S}'_{k,1}| \bmod v} - \overline{|S'_k| \bmod v})$$

The fixed error ε is set to 1. Using ε, $\tilde{E}_{k,0}$ and $\tilde{E}_{k,1}$ it is then possible to calculate a fifth and a sixth exponent. It remains a fact that, although the term “Lyapunov exponent” is used with respect to exponents 3 to 6, the reader should be aware of the fact that using this terminology here is a bit tricky. Still, using (5) for measuring such exponents, a positive result gives us a clear indication of the fact that the shift indeed plays a determining role in the behaviour of tag systems, one small shift in the initial condition giving rise to large deviations in the future behaviour of the system. In this respect, while being tricky, the use of the term Lyapunov exponent is reasonable here, since it exactly measures that which the Lyapunov exponent is used for: sensitive dependence on initial conditions.

Discussion of the results

In the following table an overview is given of the several different Lyapunov exponents calculated for each of the 52 tag systems.

Table 1: Overview of the values for the 6 different Lyapunov exponents calculated.

T.S.	L1	L2	L3	L4	L5	L6
T1	0,20986	12,6473	3,13671	0,21505	0,23024	12,6473
T2	-0,51606	5,81639	2,53615	0,37532	0,79774	5,81639

Continued on next page

Table 1 – continued from previous page

T.S.	**L1**	**L2**	**L3**	**L4**	**L5**	**L6**
T3	-0,38830	9,09703	1,93795	0,23201	0,45044	9,09703
T4	-0,23778	5,89286	1,78356	0,24394	0,79310	5,89286
T5	0,21470	4,85625	2,21598	0,31457	0,94651	4,85625
T6	0,24355	13,0258	3,48448	0,59671	0,23158	13,0258
T7	-0,28043	4,43291	2,91542	0,35915	0,94250	4,43291
T8	-0,14530	7,51333	2,60118	0,31845	0,63511	7,51333
T9	0,34157	7,36924	3,49389	0,57887	0,63472	7,36924
T10	-0,34809	9,20602	2,93124	0,27714	0,44619	9,20602
T11	-0,12525	9,68609	2,77962	0,32325	0,44555	9,68609
T12	0,24654	13,0261	3,64890	0,59495	0,23210	13,0261
T13	0,01863	3,21484	3,77603	0,65589	1,17652	3,21484
T14	-0,00993	6,48944	2,70436	0,44110	0,63465	6,48944
T15	-0,26616	6,55209	3,47492	0,32866	0,63334	6,55209
T16	-0,14565	3,78280	3,72553	1,27682	1,18847	3,78280
T17	-0,01439	3,39951	3,32477	0,41777	1,19019	3,39951
T18	0,00763	3,69115	3,73933	1,05045	0,98165	3,69115
T19	0,38320	7,86141	2,68984	0,28661	0,63305	7,86141
T20	0,00101	4,36420	2,13883	0,28047	1,05747	4,36420
T21	-0,00779	3,10876	1,61515	0,22656	1,17244	3,10876
T22	-0,10948	2,63181	3,20627	0,77904	1,37792	2,63181
T23	-0,00836	9,76965	3,05023	0,60139	0,45057	9,76965
T24	-0,34111	6,33215	3,54355	0,53880	0,63407	6,33215
T25	-0,32030	3,36655	1,78095	0,20042	1,17318	3,36655
T26	-0,33586	6,42591	2,03395	0,20469	0,79752	6,42591
T27	0,24441	6,04969	2,27059	0,23823	0,80015	6,04969
T28	-0,14936	3,78191	3,65095	1,00675	1,18414	3,78191
T29	-0,36760	9,52192	2,55202	0,37272	0,44827	9,52192
T30	0,27572	4,88829	1,75367	0,25418	0,94291	4,88829
T31	-0,00584	13,1268	3,56512	0,56356	0,23223	13,1268
T32	-0,21426	1,25821	1,90897	0,34805	1,53561	1,25821

Continued on next page

Table 1 – continued from previous page

T.S.	L1	L2	L3	L4	L5	L6
T33	-0,15379	6,22818	2,41893	0,33040	0,79669	6,22818
T34	-0,64277	9,31465	3,51719	0,72575	0,45076	9,31465
T35	-0,21476	3,30227	3,56580	1,38173	1,27459	3,30227
T36	-0,17917	6,93520	2,27206	0,40513	0,63462	6,93520
T37	-0,45828	6,63360	1,81005	0,18229	0,63713	6,63360
T38	0,00212	4,72056	2,52954	0,54072	0,94197	4,72056
T39	-0,36713	4,48617	3,71934	1,01565	0,94284	4,48617
T40	0,00830	9,21331	2,76923	0,45992	0,44893	9,21331
T41	0,20195	4,35241	4,06979	0,97182	0,94938	4,35241
T42	0,00108	5,69042	1,81780	0,19552	0,80445	5,69042
T43	0,15567	10,2722	3,25836	0,34293	0,44612	10,2722
T44	-0,26523	6,02164	0,00107	0,00395	0,69426	6,02164
T45	-0,69882	7,61557	3,79718	0,96948	0,63165	7,61557
T46	0,12586	7,63512	2,20026	0,29539	0,63789	7,63512
T47	-0,00204	9,89289	3,94462	1,19387	0,44535	9,89289
T48	-0,35373	2,61741	1,81118	0,22834	1,45931	2,61741
T49	-0,27384	8,90288	2,91498	0,62219	0,44738	8,90288
T50	-0,20170	9,97428	2,37917	0,27435	0,44720	9,97428
T51	0,21810	3,75900	3,05455	0,43194	1,17205	3,75900
T52	-0,43902	6,18305	2,35307	0,28940	0,80082	6,18305

As is clear from these results, tag systems do have a positive Lyapunov exponent. Again, as was also the case for the previous experiments there are clear variations between the different tag systems. As was expected, the first Lyapunov exponent has negative values for the majority of the tag systems. However, as is clear from the second Lyapunov exponents, the exponent becomes positive in doing the measurement after two applications of $\overset{\circ}{\rightarrow}$.
The fact that the class of tag systems studied here have positive exponents is not only a further affirmation of the fact that shift and position of the letters

are determining factors in the behaviour of tag systems, but it also gives us an indication of how sensitive a system is to these parameters. Given the fact that the tag systems used in this experiment all satisfy the same set of constraints, used to limit the possibility of solvability and thus letting open the possibility of intractability or even unsolvability, one is led to the conclusion that tag systems for which all results point into the direction of what can at least be called intractable behaviour, all show sensitive dependence on initial conditions. So what about tag systems that do not satisfy these constraints? We have not investigated this question to its full extent, but as far as the solvable class of tag systems with $\mu = 2, \nu = 2$ is concerned, we can only say that they are far less sensitive to initial conditions. In Sec. 9.4.2 we will give a proof of the solvability of this class of tag systems. From the results of this proof it indeed follows that this class of tag systems is not sensitive to initial conditions. Indeed, the proof basically differentiates between classes of initial conditions. If a certain initial condition belongs to a a specific class, changing the shift or a letter will normally have no effect for the final behaviour of the system.

The fact that the results from the experiment show that the tag systems studied here are sensitive to small changes, serves as a further indication that these tag systems might be very hard to prove solvable. Indeed, since positive Lyapunov exponents are a sign of chaotical behaviour and since systems showing chaotical behaviour are known to be complex and lead to unpredictability – think about the famous butterfly effect – the results given here only affirm that these tag systems are, from an experimental point of view, very hard to predict.

Experiment 4: Measuring the distribution of the letters

One of the constraints (constraint 4) implemented in the tag generating algorithm, takes into account the proportions between the total number of times each letter appears in all the words. Although this constraint is not necessary valid for certain tag systems that show intractable behaviour, since we have to exclude certain classes of initial conditions for certain tag systems, it was argued that this constraint can indeed function as a decidability criterion for certain classes of tag systems. One of the problems with this constraint is that if it is true that every letter has *effectively* an equal chance to be scanned, i.e., while actually executing the system, the tag process must in the end become periodic or halt, as was argued in 6.1.2 following Hayes and Minsky. This implies that we are facing a hard theoretical problem here.

If it is indeed true that every letter has an equal chance to be scanned, taking into account the means of the total number of 0's and 1's to be scanned in the initial condition, it must be predictable through some kind of estimate (taking into account the average length of the strings produced) the number of steps it takes a tag system to become periodic or halt.

Let us denote in what follows the mean $\#0/(l_0 + l_1)$ rsp. $\#1/(l_0 + l_1)$ of the total number of 0's and 1's in the words calculated through constraint 4 as μ_0 and μ_1. Then, given an initial condition and a tag system **T** that satisfies constraint 4. After the initial condition has been processed, a certain number of 0's and 1's will have been scanned, leading to a string, that is either short, longer or of the same length as the initial condition. If the mean of the number of 0's in the initial condition is larger than μ_0 the system will halt or produce a string for which the mean of the number of 0's and 1's is equal to μ_0 rsp. μ_1 after a certain number of iterations.[6] If relatively more 1's than 0's are scanned, the system will keep on growing until again a string is produced for which the mean of the number of 0's and 1's is equal to μ_0 rsp. μ_1. The question then of course is: when will a string be produced for which these means are equal to μ_0 and μ_1? As will become clear from the experimental results to be discussed here, it does not take very long for the tag system to produce a string for this to be true. Without further tackling this last question, it is clear that once a string is produced for which the mean of the total number of 0's and 1's is equal to μ_0 and μ_1 (thus leading to the production of a string through $\overset{\circ}{\rightarrow}$ that is neither shorter nor longer than the string it resulted from) we are faced with the problem of whether or not, *on the average*, the distribution of the number of 0's and 1's scanned, remains equal to μ_0 and μ_1. If this be true, it should be possible to

[6] The mean would then be equal to the total number of 0's and 1's in a string of length l, each divided by l.

find an estimate to predict when exactly the system will either halt or become periodic.
In order to further study this problem, experiment 4 was set up, effectively measuring the average of the number of times a 0 or a 1 is scanned by each of the tag systems **T1–T52**. Before starting with the description of the set-up of the experiment it is important to point out that if *on the average* the mean for the number of 0's and 1's scanned is equal to μ_0 rsp. μ_1, this feature does not make it impossible for a tag system to produce over 10000000 strings when started with an initial condition of length 300, before leading to periodicity or a halt, and thus would not contradict the results from experiment 1. Indeed, given length l of a string produced, there are about $2^{l/v}$ possible combinations of length l the system can range over. Since the tag systems will not merely produce strings of length 300, μ_0 resp. μ_1 only being valid on the average, the tag systems should have enough "space" to produce, for some initial conditions over 10000000 strings before becoming periodic or halting.

Set-up of experiment 4

We will use 10 of the initial conditions classified as Immortals? in experiment 1. For each of these conditions, the tag system was run for 10000000 iterations. Thus, the size of the sample space N is thus equal to $N = 10^8$.

Each event $x_{i,j}$ is equal to either 1 (true) or 0 (false) depending on whether the symbol i (i.e. 0, zero, or 1, one) occurs or occurs not on the jth iteration. After 10 initial conditions have been processed in this way, the experiment determines the mean (expected value) $\mu_{0,N}$ and $\mu_{1,N}$ for rsp. the number of times a 0 or a 1 was scanned, using the variables $x_{i,j}$ where:

$$\mu_{i,N} = \textstyle\sum_{j=1}^{N} \frac{x_{i,j}}{N} \quad i \in \{0,1\}$$

Using $\mu_{i,N}$ the standard deviations $\sigma_{0,N}$ and $\sigma_{1,N}$ are calculated as follows:

$$\sigma_{i,N} = \sqrt{\sum_{j=1}^{N} (x_{i,j} - \mu_{i,N})^2}$$

This was done for each of the tag systems **T1–T52**. In order to check whether the means $\mu_{0,N}$ and $\mu_{1,N}$ converge or do not converge to specific values, the mean and standard deviations were calculated after 5000000 respectively 10000000 iterations were performed. The experiment was repeated for another set of 10 initial conditions classified as Immortals? by experiment 1, namely the first 10 found in the second run of experiment 1.
Besides measuring the mean and standard deviations of the number of 0's and 1's in the strings produced, a subroutine was included that measured how many iterations were minimally and maximally needed before a string was produced for which the ratio of 0's resp. 1's scanned, to the total number of symbols scanned is equal to $\mu_0 = \#0/(l_0 + l_1)$ and $\mu_1 = \#1/(l_0 + l_1)$. This was done to tackle the problem to know how long it takes before a string is produced for which the mean of the number of 0's and 1's is equal to μ_0 rsp. μ_1. In this way the possible influence of an initial imbalance between the total number of 0's and 1's on the actual values of $\mu_{0,N}$ and $\mu_{1,N}$ can be excluded, as will become clear through the results.

Discussion of the results of experiment 4

In table the results are given of how many iterations were minimally and maximally needed, for each of the initial conditions for each of the tag systems, before a string was produced for which the ratio of 0's resp. 1's scanned, to the total number of symbols scanned is equal to $\mu_0 = \#0/(l_0 + l_1)$ and $\mu_1 = \#1/(l_0 + l_1)$.

Table 2: Minimum and maximum number of iterations for a string to be produced with μ_0 and μ_1

T.S.	**Min**	**Max**
T1	12	1296
T2	0	196
T3	1	125
T4	1	277
T5	7	157

T.S.	**Min**	**Max**
T6	2	402
T7	4	141
T8	42	204
T9	0	58
T10	18	278
T11	3	323
T12	4	543
T13	1	73
T14	1	381
T15	1	79
T16	0	19
T17	1	34
T18	6	102
T19	48	192
T20	0	182
T21	1	85
T22	1	105
T23	45	551
T24	6	84
T25	0	128
T26	0	204
T27	0	196
T28	1	40
T29	1	159
T30	10	144
T31	1	110
T32	0	94
T33	0	141
T34	63	648
T35	19	625
T36	0	133

T.S.	**Min**	**Max**
T37	1	222
T38	21	355
T39	8	402
T40	60	354
T41	7	33
T42	0	232
T43	9	320
T44	0	110
T45	1	173
T46	3	290
T47	4	669
T48	13	202
T49	60	270
T50	16	191
T51	0	102
T52	4	63

These results show that the minimum and maximum number of iterations before a string is produced for which the average number of relevant 0's and 1's is equal to μ_0 and μ_1 is very low. We can thus conclude that whatever the results for $\mu_{0,N}$ and $\mu_{1,N}$ might be, the influence on these results of a possible imbalance between the total number of 0's and 1's relative to μ_0 and μ_1 can be neglected.

In table 3 the results from experiment 4 are shown. For each tag system there are four rows with data. The first two rows show means and standard deviations for the number of 0's scanned by the tag system, for two runs of the experiment, the last two rows the results for the number of 1's scanned.

Table 3: Means and Standard deviations for the number of significant letters 0 and 1 for two runs of the same experiment with different initial conditions.

T.S.	$\mu_{5.10^6}$	$\mu_{1.10^7}$	$\sigma_{5.10^6}$	$\sigma_{1.10^7}$
T1	0,49938814	0,49955107	0,12998921	0,12996718
	0,49925946	0,49961642	0,13011946	0,13002905
	0,50061186	0,50044893	0,12998920	0,12996718
	0,50074054	0,50038358	0,13011946	0,13002904
T2	0,49947212	0,49970484	0,14838064	0,14848491
	0,49959620	0,49971658	0,14841034	0,14852215
	0,50052788	0,50029516	0,14838064	0,14848491
	0,50040380	0,50028342	0,1484103	0,14852215
T3	0,49983170	0,49992785	0,13075027	0,13069996
	0,49990816	0,49993954	0,13075214	0,13084408
	0,50016830	0,50007215	0,13075027	0,13069996
	0,50009184	0,50006046	0,13075214	0,13084408
T4	0,49973474	0,49988311	0,13221058	0,13222642
	0,49982328	0,49987309	0,13233494	0,13232892
	0,50026526	0,50011689	0,13221058	0,32226417
	0,50017672	0,50012691	0,13233494	0,13232892
T5	0,49976184	0,49977215	0,12015511	0,12004122
	0,49955336	0,4997634	0,12000163	0,11999297
	0,50023816	0,50022785	0,12015511	0,12004122
	0,50044664	0,50023654	0,12000163	0,11999297
T6	0,49972618	0,49978999	0,09555880	0,09559299
	0,49971438	0,49982279	0,09559107	0,09566045
	0,50027382	0,50021001	0,09555880	0,09559299
	0,50028562	0,50017721	0,09559107	0,09566045
T7	0,49951472	0,49974622	0,18011528	0,18008183

Continued on next page

Table 3 – continued from previous page

T.S.	$\mu_{5.10^6}$	$\mu_{1.10^7}$	$\sigma_{5.10^6}$	$\sigma_{1.10^7}$
	0,49955906	0,49979458	0,17997552	0,18007529
	0,50048528	0,50025378	0,18011528	0,18008183
	0,50044094	0,50020542	0,17997552	0,18007529
T8	0,33305362	0,33317766	0,09178837	0,09174928
	0,33293400	0,33320407	0,09175058	0,09174997
	0,66694638	0,66682234	0,09178837	0,09174928
	0,66706600	0,66679593	0,09175048	0,09174997
T9	0,49966454	0,49985959	0,13156549	0,13162844
	0,49959824	0,49985637	0,13151971	0,13159662
	0,50033546	0,50014041	0,13156549	0,13162844
	0,50040176	0,50014362	0,13151971	0,13159662
T10	0,49938774	0,49962633	0,15431907	0,15429176
	0,49917504	0,49966490	0,15424648	0,15429739
	0,50061226	0,50037367	0,15431907	0,15429176
	0,50082496	0,50033501	0,15424648	0,15429739
T11	0,49950282	0,49972887	0,12215274	0,12223241
	0,49946464	0,49973360	0,12214831	0,12224636
	0,50049718	0,50027113	0,12215274	0,12223241
	0,50053536	0,50026640	0,12214831	0,12224636
T12	0,49972586	0,49982412	0,09560242	0,09562781
	0,49944148	0,49984048	0,09550610	0,09562127
	0,50027414	0,50017588	0,09560242	0,09562781
	0,50055852	0,50015952	0,09550610	0,09562127
T13	0,49949974	0,49968790	0,15002645	0,14995474
	0,49927004	0,49970047	0,14968294	0,14983029
	0,50050026	0,50031210	0,15002645	0,14995474
	0,50072996	0,50029953	0,14968294	0,14983030
T14	0,49963248	0,49977276	0,13008482	0,13015534

Continued on next page

Table 3 – continued from previous page

T.S.	$\mu_{5.10^6}$	$\mu_{1.10^7}$	$\sigma_{5.10^6}$	$\sigma_{1.10^7}$
	0,49953060	0,49981012	0,13005642	0,13020544
	0,50036752	0,50022724	0,13008482	0,13015534
	0,50046940	0,50018988	0,13005642	0,13020544
T15	0,49972884	0,49986530	0,16651339	0,16655761
	0,49954180	0,49981591	0,16647819	0,16650336
	0,50027116	0,50013470	0,16651339	0,16655761
	0,50045820	0,50018409	0,16647819	0,16650336
T16	0,49979408	0,49990788	0,12480711	0,12485622
	0,49984036	0,49991851	0,12483387	0,12486723
	0,50020592	0,50009212	0,12480711	0,12485622
	0,50015964	0,50008149	0,12483387	0,12486723
T17	0,49978560	0,49973714	0,15985869	0,15980832
	0,49982562	0,49976360	0,15980718	0,15978938
	0,50021440	0,50026286	0,15985869	0,15980832
	0,50017438	0,50023640	0,15980718	0,15978938
T18	0,66638342	0,66652311	0,09462977	0,09473888
	0,66641522	0,66651897	0,09466949	0,09,47239
	0,33361658	0,33347689	0,09462977	0,09473888
	0,33358478	0,33348103	0,09466949	0,09472386
T19	0,33308414	0,33320892	0,09175205	0,09173822
	0,33311054	0,33316499	0,09173671	0,09174116
	0,66691586	0,66679108	0,09175205	0,09173822
	0,66688946	0,66683501	0,09173671	0,09174116
T20	0,49974366	0,49973723	0,16357288	0,16352701
	0,49986852	0,49978575	0,16361993	0,16354407
	0,50025634	0,50026277	0,16357288	0,16352701
	0,50013148	0,50021425	0,16361993	0,16354407
T21	0,49981100	0,49987312	0,14490903	0,14491107

Continued on next page

Table 3 – continued from previous page

T.S.	$\mu_{5.10^6}$	$\mu_{1.10^7}$	$\sigma_{5.10^6}$	$\sigma_{1.10^7}$
	0,49977838	0,49986938	0,14488041	0,14489612
	0,50018900	0,50012688	0,14490903	0,14491107
	0,50022162	0,50013062	0,14488041	0,14489612
T22	0,49959102	0,49990204	0,16306324	0,16305483
	0,49964138	0,49991681	0,16314762	0,16306462
	0,50040898	0,50009796	0,16306324	0,16305483
	0,50035862	0,50008319	0,16314762	0,16306462
T23	0,66640374	0,66653087	0,10210538	0,10210400
	0,66641296	0,66649893	0,10208961	0,10208835
	0,33359626	0,33346913	0,10210538	0,10210400
	0,33358704	0,33350107	0,10208961	0,10208835
T24	0,49979174	0,49986252	0,11619561	0,11624656
	0,49981954	0,49989120	0,11631582	0,11631089
	0,50020826	0,50013748	0,11619561	0,11624656
	0,50018046	0,50010878	0,11631588	0,11631089
T25	0,49945316	0,49974298	0,16185312	0,16176421
	0,49945262	0,49976761	0,16176812	0,16172444
	0,50054684	0,50025702	0,16185312	0,16176421
	0,50054738	0,50023239	0,16176812	0,16172444
T26	0,49969278	0,49981653	0,12353910	0,12353606
	0,49975510	0,49980258	0,12354148	0,12353893
	0,50030722	0,50018347	0,12353910	0,12353606
	0,50024490	0,50019742	0,12354148	0,12353893
T27	0,49969210	0,49981332	0,15659398	0,15657887
	0,49963414	0,49982193	0,15659637	0,15661223
	0,50030790	0,50018668	0,15659398	0,15657887
	0,50036586	0,50017807	0,15659637	0,15661223
T28	0,49984624	0,49991003	0,12478911	0,12483193

Continued on next page

Table 3 – continued from previous page

T.S.	$\mu_{5.10^6}$	$\mu_{1.10^7}$	$\sigma_{5.10^6}$	$\sigma_{1.10^7}$
	0,49979740	0,49990626	0,12480439	0,12483158
	0,50015376	0,50008997	0,12478911	0,12483193
	0,50020260	0,50009374	0,12480439	0,12483158
T29	0,49978982	0,49981942	0,14841228	0,14844056
	0,49974866	0,49986636	0,14855689	0,14854926
	0,50021018	0,50018058	0,14841228	0,14844056
	0,50025134	0,50013364	0,14855689	0,14854926
T30	0,49970360	0,49981825	0,12880062	0,12874801
	0,49973298	0,49979978	0,12872202	0,12873290
	0,50029640	0,50018175	0,12880062	0,12874801
	0,50026702	0,50020022	0,12872202	0,12873290
T31	0,49980272	0,49988075	0,10384891	0,10392084
	0,49978128	0,49989686	0,10389587	0,10391434
	0,50019728	0,50011925	0,10384891	0,10392084
	0,50021872	0,50010314	0,10389587	0,10391434
T32	0,49982960	0,49986072	0,15392181	0,15399016
	0,49979662	0,49981192	0,15383678	0,15393101
	0,50017040	0,50013928	0,15392181	0,15399016
	0,50020338	0,50018808	0,15383678	0,15393101
T33	0,49945962	0,49962027	0,14452561	0,14457672
	0,49963512	0,49980241	0,14461800	0,14458836
	0,50054038	0,50037973	0,14452561	0,14457672
	0,50036488	0,50019759	0,14461710	0,14458836
T34	0,66620500	0,66637831	0,13500016	0,13500566
	0,66604222	0,66633337	0,13495825	0,13496250
	0,33379500	0,33362169	0,13500016	0,13500566
	0,33395778	0,33366663	0,13495825	0,13496250
T35	0,57117182	0,57132786	0,09933985	0,09941825

Continued on next page

Table 3 – continued from previous page

T.S.	$\mu_{5.10^6}$	$\mu_{1.10^7}$	$\sigma_{5.10^6}$	$\sigma_{1.10^7}$
	0,57115694	0,57130930	0,09926328	0,09936833
	0,42882818	0,42867214	0,09933985	0,09941825
	0,42884306	0,42869070	0,09926328	0,09936833
T36	0,49976804	0,49985907	0,11562680	0,11557620
	0,49984234	0,49988718	0,11558913	0,11555868
	0,50023196	0,50014093	0,11562680	0,11557620
	0,50015766	0,50011282	0,11558913	0,11555868
T37	0,49988144	0,49993118	0,13993679	0,13995524
	0,49991570	0,49994972	0,13979562	0,13983142
	0,50011856	0,50006882	0,13993679	0,13995524
	0,50008430	0,50005028	0,13979562	0,13983142
T38	0,66630288	0,66645279	0,11988668	0,11994058
	0,66621214	0,66639295	0,11984015	0,11989572
	0,33369712	0,33354721	0,11988668	0,11994058
	0,33378786	0,33360705	0,11984015	0,11989572
T39	0,59969032	0,59986344	0,13080099	0,13075185
	0,59975322	0,59979177	0,13080073	0,13078377
	0,40030968	0,40013656	0,13080099	0,13075185
	0,40024678	0,40020823	0,13080073	0,13078377
T40	0,66641428	0,66650411	0,09463214	0,09469003
	0,66643667	0,66651491	0,09465528	0,94689283
	0,33358572	0,33349589	0,09463214	0,09469003
	0,33356330	0,33348509	0,09465528	0,09468928
T41	0,49984712	0,49991314	0,11798523	0,11800486
	0,49983632	0,49991098	0,11792113	0,11801989
	0,50015288	0,50008686	0,11798523	0,11800486
	0,50016368	0,50008902	0,11792113	0,11801989
T42	0,49977064	0,49982939	0,13232168	0,13227383

Continued on next page

Table 3 – continued from previous page

T.S.	$\mu_{5.10^6}$	$\mu_{1.10^7}$	$\sigma_{5.10^6}$	$\sigma_{1.10^7}$
	0,49984594	0,49987804	0,13225646	0,13226538
	0,50022936	0,50017061	0,13232168	0,13227383
	0,50015406	0,50012196	0,13225646	0,13226538
T43	0,49953504	0,49966654	0,17514851	0,17516905
	0,49938176	0,49966047	0,17521036	0,17519734
	0,50046496	0,50033346	0,17514851	0,17516905
	0,50061824	0,50033953	0,17521036	0,17519734
T44	0,49988816	0,49996036	0,12527685	0,12524682
	0,49987316	0,4999750	0,12523898	0,12522297
	0,50011184	0,50003964	0,12527685	0,12524682
	0,50012684	0,50002510	0,12523898	0,12522297
T45	0,49967972	0,49982178	0,14709139	0,14714867
	0,49970650	0,49980823	0,14701959	0,14705493
	0,50032028	0,50017822	0,14709139	0,14714867
	0,50029350	0,50019177	0,14701959	0,14705493
T46	0,49980734	0,49987601	0,11822903	0,11820656
	0,49983636	0,49990080	0,11818817	0,11815686
	0,50019266	0,50012399	0,11822903	0,11820656
	0,50016364	0,50009920	0,11818817	0,11815686
T47	0,49977184	0,49987176	0,09156444	0,09157788
	0,49975230	0,49986460	0,09161004	0,09160956
	0,50022816	0,50012824	0,09156444	0,09157788
	0,50024770	0,50013540	0,09161004	0,09160956
T48	0,49956328	0,49963353	0,15639615	0,15635436
	0,49956376	0,49961638	0,15644630	0,15636404
	0,50043672	0,50036647	0,15639615	0,15635436
	0,50043624	0,50038362	0,15644630	0,15636404
T49	0,66638068	0,66649068	0,11605808	0,11606001

Continued on next page

Table 3 – continued from previous page

T.S.	$\mu_{5.10^6}$	$\mu_{1.10^7}$	$\sigma_{5.10^6}$	$\sigma_{1.10^7}$
	0,66633558	0,66643564	0,11599395	0,11600792
	0,33361932	0,33350932	0,11605808	0,11606001
	0,33366442	0,33356436	0,11599395	0,11600792
T50	0,49967284	0,49978430	0,12749461	0,12752230
	0,49971274	0,49988220	0,12756446	0,12754785
	0,50032716	0,50021570	0,12749461	0,12752230
	0,50028726	0,50011780	0,12756446	0,12754785
T51	0,49966630	0,49977212	0,15440620	0,15444849
	0,49968416	0,49976963	0,15439637	0,15443534
	0,50033370	0,50022788	0,15440620	0,15444849
	0,50031584	0,50023037	0,15439637	0,15443534
T52	0,49958554	0,49979053	0,15330951	0,15333547
	0,49961964	0,49978397	0,15340843	0,15339744
	0,50041446	0,50020947	0,15330951	0,15333547
	0,50038036	0,50021603	0,15340843	0,15339744

Looking at the results from the table, it is clear that for the tag systems used in the experiment, the means measuring the number of times a 0 is scanned and the number of times a 1 is scanned in actually executing each of these systems, *almost* converge with μ_0 and μ_1, the means of the total number of 0's and 1's in the words of each of the tag systems. Take for example **T1**, for which the total number of 0's in the respective words is equal to the total number of 1's with means $\mu_{0,5\cdot10^6}$ = 0,49938814 and $\mu_{1,5\cdot10^6}$ = 0,50061186, values which both approximate $\mu_0 = \mu_1 = 0.5$ for **T1**, but differ from it only slightly. As an isolated result, this is far from surprising. Given the experimental character of these results one cannot expect a perfect match of both means with 0.5. In the second run of the experiment, we see again that $\mu_{0,5\cdot10^6}$ is a tiny bit smaller than 0.5 and $\mu_{1,10\cdot10^6}$ a tiny bit larger than 0.5. Still as isolated results, we cannot draw any conclusions here. However, this kind of observation is not only found for **T1**, but for all the tag systems tested. For example, **T8** for which $\mu_0 = 0.33333...$, $\mu_1 = 0.66666...$ the actual means after 5000000 iterations are $\mu_{0,5\cdot 10^6}$ rsp. 0,33305362 and $\mu_{0,5\cdot 10^6} = 0{,}66694638$. Given the generality of the observations that each $\mu_{0,5\cdot 10^6}$ and $\mu_{0,10\cdot 10^6}$ is always a tiny bit smaller than the expected mean μ_0 based on the number of 0's in the words of the tag systems while each $\mu_{1,5\cdot 10^6}$ and $\mu_{1,10\cdot 10^6}$ is always a bit larger than μ_1, we are led to the conclusion that this observation might be generalizable, at least when initial conditions are involved that lead to a relatively large number of iterations without leading to one of the three classes of predictable behaviour.

This result illustrates how it is possible for a given tag system to behave in a very erratic way, showing what we have earlier called “fluctuating” behaviour. Indeed, in general the tag systems will scan a tiny bit more 1's than 0's, thus leaving the system just enough space to produce new strings. As was explained before, if this would not be the case, the system must become periodic or halt after a certain number of iterations, since the total number of strings that can be produced is finite, because the lengths are limited. Although it is still possible for the tag systems to become periodic or halt despite these small deviations, this will not necessarily happen. It is also important to point out that since it is always the case that a bit more 1's are scanned on the average this does not imply that the system must lead to unbounded growth. Indeed, although the system grows on the average, it is not necessarily the case that for arbitrary n, the system will, from a certain point on, never produce a string of length shorter than n. In a way, given the values of $\mu_{0,N}$ being very close to μ_0 the system can always return to strings of shorter length. One could maybe say that both values $\mu_{0,N}$ and $\mu_{1,N}$ keep each other in balance: $\mu_{1,N}$ allows for the possibility that the system can keep producing new strings without necessary becoming periodic or halting, while $\mu_{0,N}$ assures that the system will not show unbounded growth.

A further important observation based on the results from table 3 is that, except for **T17** and **T20** the values for $\mu_{0,N}$ and $\mu_{1,N}$ do converge to the values μ_0 and μ_1, the larger N becomes. Indeed, except for the two tag systems mentioned the value for $\mu_{0,5\cdot 10^6} < \mu_{0,10\cdot 10^6}$ and $\mu_{1,5\cdot 10^6} > \mu_{1,10\cdot 10^6}$. If this result can be generalized, this implies that in the end the tag systems will indeed become periodic or halt. The fundamental question to be asked then is how many iterations this "in the end" will take. Based on the results in the table we cannot draw any conclusions of how slow or fast such convergence will take place.

Experiment 5: Tag systems and randomness.

> I hope you will inform me of results, good or bad, of new kinds of generators you have tested, particularly deterministic generators, but also the output of physical devices.(I have found none of the latter that get past DIEHARD, and would like to learn of any that do.) Since, in my opinion, there is no true randomness, collective experience in finding sequences that depart from the theoretical ideal in a significant way can perhaps lead to better ways for finding those that do not.
>
> George Marsaglia, 1986.[7]

Given the results from the previous experiment, it seemed interesting to test whether the distribution of the significant letters in the strings produced through the tag systems can in any way be considered random or not. Now, given the fact that $\mu_{0,N}$ is always a bit smaller than μ_0 and $\mu_{1,N}$ always a bit larger than μ_1, at least for the initial conditions tested during the experiment with $N \leq 10000000$, one expects that the sequence of significant letters produced by these tag systems cannot be random. If, despite these expectations, the distribution of the relevant letters in the tag systems considered can from a certain point of view be considered random, we would have further evidence of the intractability of this class of tag systems.

The notion of randomness in the context of deterministic systems is not completely unproblematic. If we would apply the notion of randomness as defined in algorithmic information theory, we even wouldn't have to perform any tests, since there is a very short algorithm behind the sequence of significant letters produced by a tag system, and we can thus not regard the sequences of letters generated as being random. However, if one wants to draw any conclusion about the distribution of the significant letters for the tag systems considered,

[7] [Mar97], p. 1

we are forced to take up a more pragmatic point of view. Indeed, without drawing any conclusions about the “true” random character of tag systems, the purpose of this experiment is to check in how far the distribution of the relevant letters produced by a given tag systems can be called “random”, random in the sense of: having passed for several statistical tests of randomness.

Set-up of experiment 5.

In order to perform experiment 5, Marsaglia’s *Diehard. A battery of random tests* [Mar96] has been an indispensable instrument. This battery contains 15 different classes of tests for randomness.[8] I will not give a detailed description of how the tests work. Our main reasons for using the battery is simply to know whether one can connect the seeming intractable behaviour of the tag systems considered here with statistical randomness. We used Marsaglia’s Diehard because it is nowadays one of the standard tests to check whether a certain sample is statistically random. It is e.g. often used by programmers who want to check whether their random number generator is good enough to be used in a program.[9].

The data Diehard had to process for each of the tag systems, are generated by the first 15 initial conditions classified as Immortals? by experiment 1. For each tag system these 15 initial conditions are each run for 10.000.000 iterations. In order for Diehard to check the distribution of the significant letters in these strings, they are considered in the order in which they are generated by the tag system. Our data first have to be properly converted to the correct format Diehard needs in order to work i.e. it only accepts a special kind of binary files. To be more specific Diehard needs files in binary mode, with 32-bit integers, using extended ASCII-code. Each 32-integer is thus represented by 4

[8] The tests are: Birthday Spacings test, Overlapping Permutations test, Ranks of 31x31 and 32x32 matrices, Ranks of 6x8 matrices, Monkey tests on 20-bit words, Monkey tests OPSO, OQSO, DNA, Count the 1’s in a stream of bytes, count the 1’s in specific bytes, Parking Lot test, Random Spheres test, The squeeze test, Overlapping sums test, Runs test, The Craps test

[9] More information on the tests can be found in [Mar84, MZ93]

8-bit integers, using the 256 symbols from the extended ASCII-code. We thus have to convert our sequences of bits generated by the program. To achieve this goal, the output files are, of course, all opened in binary mode. Now, instead of directly outputting each bit generated, each 8 relevant letters generated were stored in a string, then converted to its decimal value. After this, the string was set to the empty string, so that the above sketched process could be repeated. Then for each of the decimal values x produced in this way, the following operation is applied: x mod 256. The result of this operation is a Long data type, a 32-bit integer, which is then stored in the binary file through its ASCII code.

It is generally known that producing the correct format for Diehard asks for rather involved procedures. In order to check whether the conversion we used works, we applied Diehard to the random number generator used in VB, using a randomize timer function. In this respect, the results from the experiment should be considered relative the VB's random number generator.

Discussion of the results from experiment 5.

In the following table the results are shown of applying Diehard to each of the binary files generated for each of the 52 tag systems. Furthermore, the first line of data in the table shows the same results for VB's pseudo-random number generator. Now, if Diehard is applied to a given file it results in a text-file containing the results for each of the tests. To know whether the data in your file pass a given test you have to look at what is called the p-value, which is in the interval [0, 1). As is pointed out in the output files from Diehard:

> Most of the tests in DIEHARD return a p-value, which should be uniform on [0,1) if the input file contains truly independent random bits. Those p-values are obtained by p=F(X), where F is the assumed distribution of the sample random variable X—often normal. But that assumed F is just an asymptotic approximation, for which the fit will be worst in the tails. Thus you should not be surprised with occasional p-values near 0 or 1, such as .0012 or .9983. When a bit stream really FAILS BIG, you will get p's of 0 or 1 to six or more places. By all means, do not, as a Statistician might, think that a p < .025 or p> .975 means that the RNG has "failed

> the test at the .05 level". Such p's happen among the hundreds that DIEHARD produces, even with good RNG's. So keep in mind that " p happens".

Given this explanation, a tag system does not pass a test when it produces p-values equal to 0 or 1 to six or more places, such as e.g. 1.000000.
Now, if the output from a given tag system passes for a given test, a 1 is put in the correct column, else a 0 is added. If a test is actually a set of tests, we use a string of 0's and 1's to indicate for which of the tests the systems failed or not. For certain tests not one but several p-values are outputted. It is then possible that some of the p-values are OK while others or not. If this happens, a "?" is used instead of a 0 or a 1.

Table 4: Overview of the results from Diehard

T.S.	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
VB	1	0	11	1	1	000	0	1	1	1	1	0	1	1	0
T1	0	0	00	0	0	000	0	0	0	0	0	0	0	0	0
T2	0	1	11	0	0	000	0	0	0	0	0	0	1	1	?
T3	0	1	11	0	0	000	0	0	0	0	1	1	1	1	?
T4	0	1	11	0	0	001	0	0	0	0	1	0	1	1	?
T5	0	1	11	0	0	000	0	0	0	0	0	0	0	1	0
T6	0	1	11	0	0	000	0	0	0	0	0	1	1	1	1
T7	0	1	00	0	0	000	0	0	0	0	0	0	0	1	?
T8	0	0	11	0	0	000	0	0	0	0	0	0	1	0	0
T9	0	1	11	0	0	000	0	0	0	0	1	1	1	1	1
T10	0	1	11	0	0	000	0	0	0	0	0	0	1	1	?
T11	0	1	11	0	0	001	0	0	0	0	0	0	1	1	0
T12	0	1	11	0	0	000	0	0	0	0	0	1	1	1	1
T13	0	0	00	0	0	000	0	0	0	0	0	0	1	1	?
T14	0	1	11	0	0	000	0	?	0	0	0	0	0	1	0
T15	0	0	00	0	0	000	0	0	0	0	0	0	1	1	?
T16	0	1	11	0	0	000	0	0	0	0	0	0	0	1	0
T17	0	1	00	0	0	001	0	0	0	0	0	0	1	1	?

Continued on next page

Table 4 – continued from previous page

T.S.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
T18	0	0	11	0	0	000	0	0	0	0	0	0	0	0	0
T19	0	0	11	0	0	000	0	0	0	0	0	0	1	0	0
T20	0	1	11	0	0	001	0	0	0	0	0	0	1	1	?
T21	0	1	11	0	0	000	0	0	0	0	1	0	1	1	?
T22	0	0	00	0	0	000	0	0	0	0	0	0	1	1	?
T23	0	0	11	0	0	000	0	0	0	0	0	0	1	0	0
T24	0	1	11	0	0	000	0	0	0	0	0	0	0	1	0
T25	0	1	11	0	0	000	0	0	0	0	0	0	0	1	?
T26	0	1	11	0	0	001	0	1	0	0	1	1	1	1	1
T27	0	1	00	0	0	000	0	0	0	0	0	0	1	1	?
T28	0	1	11	0	0	000	0	0	0	0	0	0	0	1	0
T29	0	1	11	0	0	000	0	0	0	0	0	0	1	1	?
T30	0	1	11	0	0	000	0	0	0	0	0	0	0	1	0
T31	0	0	11	0	0	000	0	0	0	0	0	1	1	1	1
T32	0	0	11	0	0	000	0	0	0	0	0	0	0	1	?
T33	0	1	11	0	0	000	0	0	0	0	0	0	1	1	0
T34	0	0	00	0	0	000	0	0	0	0	0	0	0	0	0
T35	0	1	11	0	0	000	0	0	0	0	0	0	1	0	?
T36	0	1	11	0	0	000	0	?	0	0	0	0	0	1	0
T37	0	1	11	0	0	000	0	0	0	0	0	0	0	1	0
T38	0	0	00	0	0	000	0	0	0	0	0	0	1	0	0
T39	0	0	11	0	0	000	0	0	0	0	0	0	0	0	?
T40	0	0	11	0	0	000	0	0	0	0	0	0	0	0	0
T41	1	1	11	0	0	001	0	1	0	0	1	1	1	1	1
T42	0	1	11	0	0	001	0	0	0	0	1	0	1	1	0
T43	0	1	11	0	0	000	0	0	0	0	0	0	0	1	0
T44	0	1	11	0	0	000	0	0	0	0	0	0	0	1	0
T45	0	1	11	0	0	000	0	0	0	0	0	0	1	1	?
T46	0	1	11	0	0	000	0	0	0	0	0	0	0	1	0
T47	0	1	11	0	0	000	0	?	0	0	0	1	1	1	0

Continued on next page

Table 4 – continued from previous page

T.S.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
T48	0	1	11	0	0	000	0	0	0	0	0	0	0	1	?
T49	0	0	11	0	0	000	0	0	0	0	0	0	1	0	0
T50	0	1	11	0	0	001	0	?	0	0	0	0	1	1	?
T51	0	0	11	0	0	000	0	0	0	0	0	0	1	1	0
T52	0	1	11	0	0	000	0	0	0	0	0	0	0	1	?

The majority of the tag systems considered pass for at least some of the tests. Let us first look at the result for the pseudo-random number generator from VB. Clearly it passes for only 10 tests, but this might be due to the method of conversion used. Still, the results for the generator from VB are significantly better than those for the tag systems.

Looking at the results for the tag systems, there are only two that did not pass any of the tests, i.e. **T1** and **T34**. In fact, Diehard did not even want to process the output files for both tag systems. At first we thought this was due to an error in the program used, but then we applied a very quick test using a visualization to compare the output of these two tag systems with that of the other tag systems. We will not go into a detailed explanation of this test, but the results showed that, as compared to the other tag systems **T1** and **T34** are indeed far less random.[10] **T26** and **T41** performed best, **T26** passing for 8 and **T41** passing for 9 tests. **T41** even passes for the first test, called the Birthday spacing test known to be a very hard test to pass. For example, one common class of pseudo-random generators, linear congruential generators, fail for this test.

[10] The test is in fact a visual test, using a technique from fractal geometry called the chaos game. This method is used to generate a fractal very quickly. The point is that for the fractal to be visualized in a good and quick way one needs a good random number generator. I applied this method to all the tag systems and VB's pseudo-random number generator for 100000 bits generated. The generator for VB resulted in the most correct visualization, without any bias in the result. The images for **T1** and **T34** were far from perfect, while those for the remaining tag systems were rather good, but always involving very small biases, visualizing certain parts of the fractal used more precise than certain other parts of the fractal.

Test 2, 3 and 14 are most frequently passed by each of the tag systems. No tag systems passes tests 4, 5, the first two of the class of tests 15, 7, 9 and 10. On the average each tag system passes about 3 tests.

Given our rather basic knowledge about tests for randomness and the fact that our conversion method might not be perfect it is impossible to draw any more conclusions based on table 4. Still, it remains a fact that the majority of tag systems passes for at least some of the tests, a fact that surprised us given the results from the previous experiment. Indeed, given the fact that $\mu_{0,N}$ is always a bit smaller than μ_0 makes it very improbable that the tag systems would pass for all the tests. That the majority of tag systems do pass for some tests illustrates again how intractable these systems actually are, at least from an experimental point of view. Furthermore, the fact that **T1** does not pass any of the tests, while it is already known as a very hard nut to crack only further supports this.

We can only conclude here that more research is needed with respect to the possible connection between statistical randomness and tag systems.

Experiment 6: Markov analysis

The present experiment was implemented for two reasons. First of all, it was used to get a better idea about the distribution of the relevant letters 0 and 1 in the tag systems involved, checking what 2-symbol-combinations of given length are possible. Secondly, the experiment measures their information theoretical entropy.

The main technique used in the experiment is that of a Markov process. For the reader who is not familiar with this concept, the following intermezzo gives a short explanation of the general idea behind Markov processes.

Markov processes as a way to measure the intractability of tag system In 1948 Claude Shannon wrote his seminal paper on information theory, called *A mathematical theory of communication* [Sha48]. Without going into the details of this paper it is important to note that the techniques used here are based on the first

> part of this paper on discrete noiseless systems, where a discrete information source is interpreted as a discrete Markov process. Markov processes are used to measure the influence of past events on contemporary events, the general method having many applications in science.[11] Now to explain the general idea a bit, I will use the example Shannon gives i.e. the English language, as a discrete Markov source. A Markov process is defined by a certain number of states and a number of transition probabilities. In understanding the English language as such a source, the states could be the letters of the alphabet plus a space. The transition probabilities give us an answer to the question of how probable it is that state x will be followed by state y. Applied to our example, one can for example ask how probable it is that the letter a will be followed by the letter z in English. Given a large sample of English texts we can then empirically approximate all the transition probabilities between all the letters of the English language. A more complicated Markov process is obtained if one looks at the probabilities of each letter when preceded by 2 letters, e.g. the probability that z will be preceded by ak. As far as the example of the English language is concerned it might be interesting to note that if one defines transition probabilities not on the level of letters but on the level of words, Markov processes can be used to generate small texts. However, one is quickly confronted with a rather difficult problem, noted by Chomsky: the longer the (syntactically sound) text one wants to generate the larger the table of the transition probabilities between words has to be, looking at larger and larger n, i.e. the probability of word x given n words.

In his [Sha48] Shannon uses Markov processes to define the information theoretical entropy of a Markov process. It is measured as follows:[12]

$$H = -K \sum_{i=1}^{N} p_i \, log_2 p_i$$

where N is the total number of states and p_i the probability of state i. The constant K merely amounts to a choice of a unit measure. Now what exactly is

[11] More information on this subject can be found in [LS06].

[12] log_2 is logarithm to base 2.

measured by the entropy H of a system? For Shannon, his measure of entropy was inspired by the following questions ([Sha48], p. 10):

> Can we define a quantity which will measure, in some sense, how much information is "produced" by such a process, or better, at what rate information is produced? (...) Can we find a measure of how much "choice" is involved in the selection of the events or of how uncertain we are of the outcome?

Indeed, H in a certain way is a measure of how unpredictable a certain information source is. As was said, in using Markov processes one measures how probable a certain state is if preceded by a combination of length n of a certain number of states. Now if we have a system with two states 0 and 1, for which the probabilities that the system is in state 0 or 1 are both equal to 0.5 whatever the length n of the combinations preceding the state one is taking into consideration, this implies that both states are uncorrelated. This means that whatever information one already has about the past behaviour of the system, this will not give us any information about what will happen next. Applying the formula to calculate H to this example, the entropy will indeed be maximal and equal to 1. If on the other hand the probability that the system is in state 1 is 0.9 and the probability that the system will go to state 0 is 0.1, H will be significantly lower, $H \approx 0.4688$. In general, given a system with N states, the maximal entropy, each state being equally probable, is always equal to N - 1.

Since H can be understood as a measure for how unpredictable a certain system is, given what one already knows about the system, it is interesting to measure H for each of the 52 tag systems. If H is close to its maximum value, this serves as a further indication of the intractability of these tag systems.

Besides measuring the entropy, Markov processes can also help us to add further strength to the results from previous experiments. In the experiment we will look at the probabilities for all possible combinations of n consecutive letters, n ranging from 2 to 10. How is this related to Markov processes? If $n = 2$, measuring the probability of all 2^2 combinations gives us the transition probabilities of rsp. 1 followed by 0, 1 followed by 1, 0 followed by 1 and 0 followed by 0. The same goes for any n. If e.g. $n = 4$, measuring the probabilities of all 2^4 combinations of 1 and 0, gives us the probability that 000 will be followed by 1, 001 will be followed by 1,...Measuring these probabilities does give us more information about the actual distribution about the relevant letters in the tag systems as compared to experiment 4, because here we do not look at the probability that a relevant letter is equal to 1, but rather at the probability that a relevant letter is equal to 1 when e.g. preceded by 10011. This indeed gives us more information about the way the relevant letters are actually distributed. Knowing that e.g. for **T1** both $\mu_{0,N}$ and $\mu_{1,N}$ are almost equal to 0.5 does not say anything about the way these 0's and 1's are actually ordered in the strings produced. Take for example the strings 0101010101010101010....and 01000101111011001001. Although for both strings the probabilities for both 0 and 1 are equal to 0.5, the way these 0's and 1's are actually distributed in these strings is very different. By using Markov processes one can get a more exact idea about these actual distribution. In other words, in looking at tag systems from the perspective of Markov processes we get a more correct view of how the letters 0 and 1 are actually ordered by looking at the transition probabilities of 0 and 1 when preceded by all possible combinations of a certain length n.

Set-up of experiment 6.

Contrary to the previous experiment, this one does not start from the initial conditions classified as Immortals? in the first experiment. Instead it works with the significant letters of every 10000th string produced by the first ten of these initial conditions, until the 10000000th. This was done to enhance speed, the sample space still remaining rather large. Furthermore I chose to work with every 10000th string produced instead of e.g. the first 10000 in order to get a general idea of the combinations for each length n allowed by a given tag system.

For each length n, n going from 2 to 10, each of these 1000000 strings is analyzed in order to find out what combinations occur and which don't. This is done as follows. Given a string S and n, the algorithm scans through S starting from the first letter in S until the letter at position $l_S - n + 1$. Starting from every of these letters, going from the first to the $l_S - n + 1$, we can determine all possible combinations of length n in string S. Given such a combination, the program checks whether the combination has already been found. For this purpose we used a very efficient search-and-sort algorithm called red-black binary search trees. I will not go into the details of how this algorithm exactly works,[13] but is interesting to point out that the average and worst-case insert, delete, and search time is equal to $O(lnn)$, where n is the number of combinations already found.

Then, if a combination has already been found the number of times this string has already been found is incremented with one. If not, this new string is added as a new combination of length n, and its counter is set to 1. In this way on finds how many different combinations there are of length n, as well as the probabilities of each of these combinations, necessary to calculate the entropy H. Given a combination C_i of length n that has been found x_i times in the strings processed, where $X = x_1 + x_2 + ... + x_N$ is the total number of combinations processed, then:

$$p_i = \frac{x_i}{X}$$

Using these p_i we can then calculate the entropy H. Once H is calculated it is normalized to 1, 1 thus becoming the maximum value for each of the entropies measured. Indeed, since we are not directly measuring the probabilities of 0 or 1 when preceded by a given combination of length $n-1$ we are actually considering the tag system as a Markov source with the number of states equal to 2^n measuring the probability of each of these states. As was said before, the maximum value of H for a Markov source with n states is equal to n. In this respect, one can normalize the value of H for such Markov source to 1 by dividing H through n.

Discussion of the results from experiment 6.

In table 5 the results from this experiment are shown. The column headed $\mathbf{C_n}$ shows the total number of different combinations found of length n for each of

[13] The interested reader can find lots of information on the internet, including red-black trees animations, descriptions of how they work and several programs in different languages implementing them. The technique was first invented by Rudolf Bayer in 1972 [Bay72].

the tag system, the column headed **H** shows the different entropies H found for each of the tag systems for each n.

Table 5: Overview of the values for H and C_n

T.S.	$\mathbf{C_n}$	**H**
T1	4	0,987928353554315
T1	8	0,981430189675427
T1	16	0,958087445055827
T1	30	0,930016548127735
T1	56	0,899877170636265
T1	98	0,867697928688029
T1	174	0,838488499523347
T1	296	0,812424402112214
T1	508	0,789706696472871
T2	4	0,997974833163837
T2	8	0,973515675359684
T2	16	0,958781095838667
T2	32	0,949096145275092
T2	64	0,941740206053233
T2	128	0,935875614335382
T2	256	0,930822072320171
T2	511	0,926118503040237
T2	1019	0,921825423766702
T3	4	0,991453762492224
T3	8	0,987029093263688
T3	16	0,982502677410114
T3	32	0,973024164388446
T3	64	0,963443869192076
T3	127	0,95393676799249
T3	249	0,944638612654986
T3	483	0,935731855285116

Continued on next page

Table 5 – continued from previous page

T.S.	C_n	H
T3	926	0,927656234792737
T4	4	0,999999772038294
T4	8	0,992900939634788
T4	16	0,988800949970723
T4	32	0,984506167904919
T4	64	0,978542816303551
T4	126	0,972541357085492
T4	247	0,966652672598187
T4	482	0,960720227905257
T4	935	0,955181862768562
T5	4	0,999999385403305
T5	8	0,998313860118975
T5	16	0,995094844208773
T5	32	0,989590208847438
T5	64	0,983514087037008
T5	127	0,976835941423444
T5	251	0,969963747716849
T5	491	0,963268409853753
T5	956	0,956957927514029
T6	4	0,993591347641056
T6	8	0,981557760003899
T6	16	0,963487715408458
T6	32	0,942667672752498
T6	64	0,923586518695249
T6	128	0,907118382868144
T6	256	0,891683455198207
T6	512	0,878126587734554
T6	1019	0,8655892532853
T7	4	0,999602636903081

Continued on next page

Table 5 – continued from previous page

T.S.	C_n	H
T7	8	0,963648956003519
T7	16	0,933191069675924
T7	30	0,901586673110578
T7	56	0,875172917444588
T7	98	0,851027814819229
T7	173	0,831013380386521
T7	294	0,812745913244989
T7	496	0,797007853764857
T8	4	0,918292702284119
T8	8	0,905915036178863
T8	15	0,897421205528586
T8	29	0,889570302999553
T8	55	0,881854188969423
T8	103	0,872518213086307
T8	195	0,862535478321108
T8	365	0,852900274238313
T8	674	0,843112050717451
T9	4	0,996242923196066
T9	8	0,99246142517034
T9	16	0,988150903075221
T9	32	0,983171381838785
T9	64	0,97819478309671
T9	128	0,973318093316027
T9	256	0,968440630456758
T9	511	0,963659902303373
T9	1020	0,958986428247648
T10	4	0,992897133035027
T10	8	0,960488646988515
T10	16	0,940043450319765
T10	32	0,923469042425438

Continued on next page

Table 5 – continued from previous page

T.S.	**C_n**	**H**
T10	64	0,910683172946748
T10	127	0,89963939302559
T10	252	0,889085283298753
T10	494	0,879376379980914
T10	967	0,870584895885724
T11	4	0,999999834551035
T11	8	0,994057793374206
T11	16	0,984844923511867
T11	32	0,969882969472102
T11	62	0,954575669621191
T11	120	0,939074986206634
T11	228	0,924873750666105
T11	434	0,911855101427949
T11	818	0,899969884935122
T12	4	0,993622835384448
T12	8	0,981594952215596
T12	16	0,963512674246496
T12	32	0,942702732773767
T12	64	0,923638106196113
T12	128	0,907172673332963
T12	256	0,891725284725174
T12	511	0,878151431135497
T12	1017	0,865600994583697
T13	4	0,998690033455919
T13	8	0,996487903644725
T13	16	0,982119440350869
T13	32	0,967326304926026
T13	63	0,948060015072296
T13	124	0,930950655704051
T13	241	0,914135044118279

Continued on next page

Table 5 – continued from previous page

T.S.	**C_n**	**H**
T13	468	0,899656297627606
T13	898	0,886076514155045
T14	4	0,999999853383587
T14	8	0,999999674104788
T14	16	0,999333739391107
T14	32	0,997501667734414
T14	64	0,994485923240983
T14	128	0,990549300343404
T14	256	0,986478188470991
T14	510	0,98192537540557
T14	1016	0,9774106104825
T15	4	0,995586826921479
T15	8	0,971840898795852
T15	16	0,957003229924972
T15	32	0,939545527436431
T15	64	0,92457865834439
T15	126	0,905019388101923
T15	244	0,887795694923577
T15	467	0,872638403205517
T15	887	0,859201894222116
T16	4	0,95913455556819
T16	8	0,940038873541977
T16	16	0,921319645277734
T16	30	0,904321242942122
T16	57	0,887417702000667
T16	105	0,872236156389779
T16	192	0,859207435158048
T16	350	0,847590513700209
T16	635	0,837388511667327

Continued on next page

Table 5 – continued from previous page

T.S.	C_n	H
T17	4	0,999997459424533
T17	8	0,987312537962706
T17	16	0,975103823316272
T17	32	0,963991963210149
T17	64	0,953131940322969
T17	128	0,942250966297885
T17	255	0,932054767209251
T17	506	0,922562601235123
T17	1002	0,913611655655647
T18	4	0,87662465240909
T18	7	0,859790206840416
T18	13	0,843778922710186
T18	23	0,829518144020049
T18	40	0,815122262864398
T18	68	0,801879378750493
T18	118	0,79075625140911
T18	199	0,780867570170298
T18	332	0,771706056168381
T19	4	0,917835616933164
T19	8	0,905481701855163
T19	15	0,897008744371083
T19	29	0,889208529966218
T19	55	0,881529211062088
T19	103	0,872229863710776
T19	195	0,862282730625318
T19	365	0,852685036855781
T19	674	0,842911184539232
T20	4	0,999999585320101
T20	8	0,964309967129064
T20	15	0,934455098865268

Continued on next page

Table 5 – continued from previous page

T.S.	**C_n**	**H**
T20	28	0,909920749075861
T20	50	0,888450166684052
T20	91	0,870769528095471
T20	161	0,855478307463583
T20	285	0,841926286777364
T20	496	0,829755209416617
T21	4	0,992609504659764
T21	8	0,988613534111605
T21	16	0,98508976653372
T21	32	0,982020568016604
T21	64	0,978896113337104
T21	128	0,976309091596172
T21	256	0,973153696130722
T21	511	0,969873169558847
T21	1017	0,966366626689781
T22	4	0,995028043069428
T22	8	0,967511445381153
T22	16	0,947169507238609
T22	32	0,930235603300027
T22	64	0,916981960478203
T22	127	0,905718704196486
T22	252	0,895612570714664
T22	495	0,886316752405463
T22	974	0,877714545491835
T23	4	0,912558775479252
T23	8	0,909695475588475
T23	15	0,901891785279827
T23	28	0,886677848940112
T23	50	0,874536684375989
T23	89	0,862852686044491

Continued on next page

Table 5 – continued from previous page

T.S.	C_n	H
T23	156	0,852168945881725
T23	270	0,841226710776098
T23	458	0,830897552447702
T24	4	0,999861569470289
T24	8	0,982077509351244
T24	16	0,970533409940886
T24	32	0,955752984899994
T24	63	0,942940923459171
T24	124	0,930821249855123
T24	240	0,917818135249777
T24	459	0,904749265663955
T24	864	0,891663402865024
T25	4	0,95905018826618
T25	8	0,930470725011647
T25	15	0,90712981910863
T25	28	0,888127107786232
T25	51	0,8725423880677
T25	92	0,860147789761333
T25	165	0,849783509253144
T25	296	0,840547449660665
T25	523	0,831536494637734
T26	4	0,999999679044943
T26	8	0,998731082382393
T26	16	0,996885688362023
T26	32	0,991741382305176
T26	62	0,982469770974469
T26	120	0,973816259585423
T26	228	0,964533431755394
T26	434	0,956372584604651

Continued on next page

Table 5 – continued from previous page

T.S.	**C_n**	**H**
T26	818	0,948510335458434
T27	4	0,998334843239256
T27	8	0,969038273054934
T27	16	0,950401408576968
T27	32	0,935406712643919
T27	63	0,921552549859523
T27	122	0,909275556961472
T27	234	0,897900138832837
T27	447	0,887432817891648
T27	849	0,877783576261407
T28	4	0,959173195787488
T28	8	0,940173319926858
T28	16	0,921433106208743
T28	30	0,904381586562028
T28	57	0,887489146960177
T28	105	0,872307562528013
T28	192	0,859279704647051
T28	349	0,847667441087415
T28	633	0,837467806910853
T29	4	0,993610810092871
T29	8	0,958864583192788
T29	16	0,937674973940885
T29	32	0,919831111538996
T29	64	0,9050637285618
T29	127	0,891998124636528
T29	252	0,880844496357431
T29	497	0,871075948354865
T29	977	0,862145319650549
T30	4	0,959130607099188

Continued on next page

Table 5 – continued from previous page

T.S.	**C_n**	**H**
T30	8	0,945589286060826
T30	16	0,931422077180189
T30	32	0,920228301605105
T30	63	0,910802825309898
T30	122	0,901652850864017
T30	231	0,891678135121636
T30	430	0,881668665906777
T30	780	0,871151705853427
T31	4	0,992614578853106
T31	8	0,981079829092837
T31	16	0,951666321773956
T31	29	0,918557455852599
T31	51	0,886594180809282
T31	89	0,858049881566155
T31	154	0,832475129013359
T31	265	0,810523777821311
T31	455	0,791383481448436
T32	4	0,995520467773172
T32	8	0,980975608826621
T32	16	0,973431971835682
T32	32	0,968299138543747
T32	64	0,963985263704641
T32	128	0,960626280580779
T32	256	0,957759857098876
T32	512	0,955169764208116
T32	1024	0,952732733178914
T33	4	0,99263806639173
T33	8	0,984097534242795
T33	16	0,968595296727312
T33	31	0,950235886827746

Continued on next page

Table 5 – continued from previous page

T.S.	C_n	H
T33	59	0,935036381578578
T33	110	0,920853780654117
T33	199	0,907142753570386
T33	357	0,894741817259847
T33	629	0,883656807454477
T34	4	0,903202581216562
T34	7	0,8841715339316
T34	13	0,832329589187641
T34	23	0,794487463566085
T34	37	0,766091476235182
T34	65	0,740097379488945
T34	106	0,718348234590988
T34	162	0,700734799347346
T34	270	0,685469994437638
T35	4	0,984327625396006
T35	8	0,981536734922642
T35	16	0,977393826718627
T35	32	0,972851287308702
T35	63	0,967700109040374
T35	123	0,962537187686786
T35	239	0,950044910910541
T35	457	0,938253337512037
T35	870	0,927008358615163
T36	4	0,998402947232386
T36	8	0,996653362225427
T36	16	0,992103576587484
T36	32	0,986852328217664
T36	64	0,980726970202156
T36	127	0,974319556707172
T36	252	0,967971708645817

Continued on next page

Table 5 – continued from previous page

T.S.	C_n	H
T36	497	0,961484163225245
T36	980	0,955225200689269
T37	4	0,990863707071144
T37	8	0,986965536205173
T37	16	0,984105916706379
T37	32	0,979458653026122
T37	64	0,974463048353324
T37	128	0,96889536827068
T37	256	0,963435643869767
T37	508	0,95746956327611
T37	1007	0,951775696333637
T38	4	0,912597137509447
T38	8	0,909103142429943
T38	15	0,898869627595231
T38	29	0,89004207511522
T38	55	0,881843494318672
T38	101	0,873152463849856
T38	189	0,865408760673346
T38	344	0,857897121453458
T38	618	0,850142887390297
T39	4	0,967971522785754
T39	8	0,964441845402299
T39	16	0,961996304861096
T39	31	0,955765182569274
T39	57	0,931879003721963
T39	98	0,90628838437281
T39	165	0,883902472134738
T39	275	0,864745248086901
T39	439	0,84796010848853

Continued on next page

Table 5 – continued from previous page

T.S.	**C_n**	**H**
T40	4	0,876290626690919
T40	7	0,859439098814319
T40	13	0,843504844480214
T40	23	0,829313858043346
T40	40	0,8149678608291
T40	68	0,801755246646372
T40	118	0,790667297984482
T40	199	0,780811307140945
T40	332	0,771677877429403
T41	4	0,999999695679684
T41	8	0,998500476196571
T41	16	0,996689283864917
T41	32	0,994176927729044
T41	64	0,990523985142974
T41	128	0,986176656727466
T41	256	0,981097064914038
T41	512	0,975773198987587
T41	1024	0,970383519632123
T42	4	0,999999801139809
T42	8	0,992762730050645
T42	16	0,988600785787819
T42	32	0,984290261418228
T42	64	0,978388494705557
T42	126	0,972410600996219
T42	247	0,966530798636443
T42	482	0,960612845242362
T42	935	0,955097361169937
T43	4	0,959058791197194
T43	7	0,898413651380682
T43	12	0,843204978417221

Continued on next page

Table 5 – continued from previous page

T.S.	C_n	H
T43	20	0,804374161629095
T43	33	0,777214467688794
T43	54	0,755473070985423
T43	88	0,737476039805785
T43	143	0,722083181098091
T43	231	0,709162334664677
T44	4	0,992681495685027
T44	8	0,986837450765276
T44	16	0,967086108017654
T44	30	0,939587123224251
T44	56	0,906188114960693
T44	98	0,873544628753868
T44	171	0,84487284571956
T44	285	0,819867591045862
T44	471	0,798543702934792
T45	4	0,99171242962545
T45	8	0,955755576357214
T45	16	0,928982217848665
T45	32	0,906979634446539
T45	63	0,890236387742149
T45	123	0,875241279187015
T45	236	0,861755686306942
T45	453	0,849749743906566
T45	855	0,838907953786991
T46	4	0,999999903219858
T46	8	0,997274986686665
T46	16	0,99159786331985
T46	32	0,982623960888116
T46	64	0,973306729215822
T46	128	0,964871109154055

Continued on next page

Table 5 – continued from previous page

T.S.	**C_n**	**H**
T46	255	0,9570642959172
T46	505	0,949188327649406
T46	995	0,941646643167942
T47	4	0,981115366723891
T47	8	0,959836325867988
T47	16	0,932954787210515
T47	32	0,912776992920237
T47	64	0,897744591925159
T47	128	0,883128665030557
T47	256	0,869571191653916
T47	512	0,856403117458891
T47	1022	0,844647896273301
T48	4	0,99697019354212
T48	8	0,967783324347551
T48	16	0,950006515987496
T48	32	0,935915037764061
T48	64	0,923981001943398
T48	126	0,913878449120504
T48	246	0,905122881739716
T48	478	0,897626934563903
T48	916	0,89090558980129
T49	4	0,916370224559969
T49	7	0,896226749168512
T49	13	0,876371914561514
T49	23	0,858827034986847
T49	37	0,841971344391714
T49	65	0,826338596583575
T49	108	0,811624060911139
T49	166	0,797664004074768

Continued on next page

Table 5 – continued from previous page

T.S.	**C_n**	**H**
T49	279	0,785042052074679
T50	4	0,999999556823011
T50	8	0,986425325206591
T50	15	0,959781159048677
T50	27	0,93459622647331
T50	49	0,911502916162561
T50	88	0,891882147753888
T50	158	0,876110763828353
T50	279	0,862472185401844
T50	486	0,850478313348463
T51	4	0,991112115683411
T51	8	0,985235781133565
T51	16	0,968134969865202
T51	32	0,951103007449003
T51	62	0,936746074006103
T51	118	0,92272469488699
T51	221	0,909911400093967
T51	410	0,898141530448617
T51	754	0,887447334350475
T52	4	0,992483544297284
T52	8	0,978213930211043
T52	16	0,964564176893529
T52	31	0,95020365145023
T52	59	0,935661341441236
T52	110	0,920421368196743
T52	201	0,906573698936217
T52	363	0,89373470890364
T52	654	0,882151716595262

There are several tag systems for which for each length n, the total number of different combinations is close or even equal to 2^n, resulting in an entropy very close to the maximum value 1. This adds further strength to the results from experiment 5. Indeed, one of the typical features of binary strings that are statistically random is that every combination of a given length n has equal probability $1/n$. As is clear from the results this is not the case, given the fact that none of the entropies is equal to 1. Since most of the entropies found however are very close to 1, it is not surprising that many of the tag systems passed for some of the tests from Diehard. The fact that most of the entropies of the tag systems considered are close to 1 again indicates how intractable these tag systems actually are.

There are some tag systems for which H decreases considerably with increasing n. For most of the tag systems there is a very good explanation for this to happen: for some of the tag systems $\mu_0 \neq \mu_1$, thus lowering the number of possible different combinations of length n. For **T1** this is not the case, while H significantly decreases with n, $H = 0,789706696472871$. This result only affirms that **T1** cannot pass for any statistical test for randomness. In general however it is clear that for most of the tag systems considered **H** is close to its maximum value.

The fact that for a given n the results indicate that for many of the tag systems most of the combinations of length n occur at a given time during the tag process, while the number of times each of these combinations occurs must be more or less the same for each of the combinations, given the value of **H**, further serves as an explanation for the fact that these tag systems can run for a very long time when started with certain initial conditions. Indeed, this result implies that the possible combinations of given length n grow exponentially fast. As was said before, one of the preconditions for a tag system not to become periodic is that when it produces a string of a given length, all possible combinations of significant letters in strings of that length should not have been exhausted yet. If all the combinations of a given length, allowed by a tag system, would have been produced already, the tag system will become periodic the next time it produces a combination of that length. Now if the growth of combinations as a function of n would be very slow, or even constant, the system in a way has less space to produce strings of a certain length. Indeed, the fewer the number of possible combinations of a given length, the higher the chance that the system can become periodic. Of course, this is a purely theoretical argument, and slower growth does not necessary imply more periodicity. As we already know, there are other factors that determine the behaviour of a tag system. We thus have to wait for the results. If however, a clear difference in the growth factors between several tag systems agrees with the differences found in the number of Immortals?, this growth could serve as another kind of measure of the intractability of tag systems.

Bibliography

[AA93] IRIS LEE ANSHEL and MICHAEL ANSHEL (1993), *From the Post-Markov Theorem through Decision Problems to Public-Key Cryptography*, The American Mathematical Monthly, **9**, 835–844.

[AB71] STAL AANDERAA and DAG BELSNES (1971), *Decision problems for tag systems*, The Journal of Symbolic Logic, **36** (2), 229–239.

[AB06] BENEDIKT LÖWE JOHN V. TUCKER ARNOLD BECKMANN, ULRICH BERGER (ed.) (2006), *Logical Approaches to Computational Barriers. Second Conference on Computability in Europe, CIE2006, Swansea, UK, Lecture Notes in Computer Science*, vol. 3988, Springer, Berlin.

[AH28] WILHELM ACKERMAN and DAVID HILBERT (1928), *Grundzüge der theoretischen Logik*, Springer, Berlin.

[Alt72] FRANZ L. ALT (1972), *Archaeology of Computers – Reminiscences, 1945–1947*, Communications of the ACM, **15** (7), 693–694.

[AS96] HAROLD ABELSON and GERALD J. SUSSMAN (1996), *Structure and Interpretation of Computer Programs*, MIT Press, Cambridge, Massachusetts, can be downloaded from: http://mitpress.mit.edu/sicp/.

[Asp84a] WILLIAM ASPRAY (1984), *Transcript of an interview with Alonzo Church by William Aspray on 17 May 1984 at the University of California, Los Angeles. (The Princeton Mathematics Community, transcript number 5. Available at:*

http://libweb.princeton.edu/libraries/firestone/rbsc/finding aids/mathoral/ pmc05.htm).

[Asp84b] ——— (1984), *Transcript of an interview with Stephen C. Kleene and J. Barkley Rosser by William Aspray on 26 April 1984 in Madison, Wisconsin. (The Princeton Mathematics Community, transcript number 23. Available at: http://libweb.princeton.edu/libraries/firestone/rbsc/finding aids/mathoral/ pmc23.htm).*

[Bac80] JOHN W. BACKUS (1980), *Programming in America in the 1950s – Some personal Impressions*, in: [HMR80], 125–135.

[Bac81] ——— (1981), *The history of Fortran I, II and III. Transcript of discussion*, in: [Wex81], 25–73.

[Bar97] HENK BARENDREGT (1997), *The Impact of the Lambda-Caclulus in Logic and Computer Science*, The Bulletin of Symbolic Logic, **3** (2), 181–215.

[Bau80] FRIEDRICH L. BAUER (1980), *Between Zuse and Rutishauer*, in: [HMR80], 505–524.

[Bay72] RUDOLF BAYER (1972), *Binary B-Trees: Data Structure and Maintenance Algorithms*, Acta Informatica, **1**, 290–306.

[BC01] DAVID H. BAILEY and RICHARD E. CRANDALL (2001), *On the random character of Fundamental Constant Expansions*, Experimental Mathematics, **10** (2), 175–190.

[BC05] MARK BRAVERMAN and STEPHEN COOK (2005), *Computing over the Reals: Foundations for Scientific Computing*, Technical report available at: http://arxiv.org/abs/cs.CC/0509042.

[BDK05] VINCENT D. BLONDEL, JEAN-CHARLES DELVENNE and PETR KURKA (2005), *Computational Universality in Symbolic Dynamical Systems*, in: MAURICE MARGENSTERN (ed.), *Machines, Computations, and*

Universality: 4th International Conference, MCU 2004, Saint Petersburg, Russia, September 21-24, 2004, Revised Selected Papers, Lecture Notes in Computer SCience, vol. 3354, Springer Verlag, Berlin, 233–244.

[Beh22] HEINRICH BEHMANN (1922), *Beiträge zur Algebra der Logik, insbesondere zum Entscheidungsproblem*, Mathematische Annalen, **86**, 163–229.

[BF97] CESARE BURALI-FORTI (1897), *Una Questione sui numeri transfiniti*, Rendiconti del Circolo matematico di Palermo, **11**, 154–164, translated in English in [vH67], 104–111.

[Boo66] WILIAM W. BOONE (1966), *Word problems and recursively enumerable degrees of unsolvability. A first paper on Thue systems*, Annals of Mathematics, **83** (3), 520–571.

[Bra83] ALLEN H. BRADY (1983), *The determination of the value of Rado's noncomputable function* Σ *for four-state Turing machines*, Mathematics of Computation, **40** (162), 647–665.

[Bra88] ——— (1988), *The Busy Beaver Problem and the Meaning of Life*, in: [Her88], 259–277.

[BS00] JOHN T. BALDWIN and SAHARON SHELAH (2000), *On the classificability of Cellular Automata*, Theoretical Computer Science, **230** (1), 117–129.

[BSS89] LEONORE BLUM, MICHAEL SHUB and STEVE SMALE (1989), *On a Theory of Computation and Complexity over the Real Numbers; NP Completeness, Recursive Functions and Universal Machines*, Bulletin of the American Mathematical Society (New Series), **21**, 1–46.

[Bul05] M. BULLYNCK (2005), *Decimal Periods and their Tables: A Research Topic (1765-1801)*, (forthcoming).

[Bul07] MAARTEN BULLYNCK (2007), *Eniacisms. Computations on the ENIAC: Some set-ups with various notes,* available at: http://www.kuttaka.org/.

[Bur66] ARTHUR W. BURKS (1966), *Preface,* in [vN66], 1–28.

[Bur86] ——— (1986), *Introduction to von Neumann's work on Natural and Artificial Automata,* in: [vN86], 363–390.

[BW72] FRIEDRICH L. BAUER and HANS WÖSSNER (1972), *The "Plankalkül" of Konrad Zuse: A forerunner of Today's programming languages,* Communications of the ACM, **15** (7), 678–685.

[Can74] GEORG CANTOR (1874), *Über eine Eigenschaft des Inbegriffes aller reellen algebraischen Zahlen,* Crelles Journal für Mathematik, **77**, 258–262, also published in [Can32], 115–118.

[Can91] ——— (1891), *Über eine elementare Frage der Mannigfaltigkeitslehre,* Jahresbericht der Deutschen Mathematiker Vereinigung, **1**, 75–78, also published in [Can32], 278–281.

[Can32] ——— (1932), *Gesammelte Abhandlungen mathematischen und philosophischen Inhalts,* Springer, Berlin.

[CD77] BRIAN E. CARPENTER and ROBERT W. DORAN (1977), *The other Turing machine,* The Computer Journal, **20** (3), 269–279.

[CD86] BRIAN E. CARPENTER and ROBERT W. DORAN (eds.) (1986), *A.M. Turing's ACE Report of 1946 and Other papers,* MIT Press, Cambridge, Massachusettes.

[Cha87] GREGORY CHAITIN (1987), *Algorithmic Information Theory,* Cambridge University Press, Cambridge.

[Chu24] ALONZO CHURCH (1924), *Uniqueness of the Lorentz transformation,* American Mathematical Monthly, **31**, 376–382.

[Chu25] ——— (1925), *On irredundant sets of postulates*, Transactions of the American Mathematical Society, **27**, 318–328.

[Chu27] ——— (1927), *Alternatives to Zermelo's assumption*, Transactions of the American Mathematical Society, **29**, 178–208.

[Chu28] ——— (1928), *On the law of the excluded middle*, Bulletin of the American Mathematical Society, **34**, 178–208.

[Chu32] ——— (1932), *A set of postulates for the foundation of logic*, Annals of mathematics, **33** (2), 346–366, 2nd series.

[Chu33] ——— (1933), *A set of postulates for the foundation of logic (second paper)*, Annals of mathematics, **34** (4), 839–864, 2nd series.

[Chu35] ——— (1935), *An unsolvable problem of elementary number theory*, Bulletin of the American Mathematical Society, **41**, 332–333, abstract of a talk given on 19 April 1935, to the American Mathematical Society.

[Chu36a] ——— (1936), *A Bibliography of Symbolic Logic*, The Journal of Symbolic Logic, **1** (4), 121–219.

[Chu36b] ——— (1936), *Editorial letter to the review section*, The Journal of symbolic logic, **1** (1), 42.

[Chu36c] ——— (1936), *An unsolvable problem of elementary number theory*, American Journal of Mathematics, (58), 345–363, also published in [Dav65b], 108–109.

[Chu36d] ——— (1936), *A note on the Entscheidungsproblem*, The journal of symbolic logic, **1** (1), 40–41.

[Chu36e] ——— (1936), *Correction to A note on the Entscheidungsproblem*, The journal of symbolic logic, **1** (3), 101–102.

[Chu37a] ——— (1937), *Review of [Pos36]*, Journal of Symbolic Logic, **2** (1), 43.

[Chu37b] ——— (1937), *Review of [Tur37],* Journal of Symbolic Logic, **2** (1), 42–43.

[Chu38] ——— (1938), *The constructive second number class,* Bulletin of the American Mathematical Society, **44**, 224–232.

[Chu41] ——— (1941), *The calculi of lambda-conversion,* no. 6 in Annals of Mathematics Studies, Princeton university Press.

[Cli48] RICHARD F. CLIPPINGER (1948), *A Logical Coding System Applied to the ENIAC (Electronic Numerical Integrator and Computer),* ballistic Research Laboratories Report Nr. 673, Aberdeen Proving Ground, 29 Sept. 1948. Available at: http://ftp.arl.army.mil/ mike/comphist/48eniac-coding/.

[CM58] NOAM CHOMSKY and GEORGE A. MILLER (1958), *Finite state languages,* Information and control, **1** (2), 91–112.

[CM63] JOHN COCKE and MARVIN MINSKY (1963), *Universality of Tag systems with P = 2,* artificial Intelligence Project – RLE and MIT Computation Center, memo 52.

[Con72] JOHN H. CONWAY (1972), *Unpredictable Iterations,* in: *Proceedings of the 1972 Number Theory conference,* University of Colorado, Boulder, 49–52.

[Con87] ——— (1987), *FRACTRAN- A Simple Universal Computing Language for Arithmetic,* in: T.M. COPER and B. GOPINATH (eds.), *Open Problems in Communication and Computation,* Springer Verlag, New York, 3–27.

[Coo] STEPHEN COOK, *The **P** versus **NP** problem,* http://www.claymath.org/millennium, clay Mathematics Institute. Millennium prize problems.

[Coo71] ——— (1971), *The complexity of theorem-proving procedures*, Proceedings of the third annual ACM symposium on Theory of computing, 151–158.

[Coo04] MATTHEW COOK (2004), *Universality in Elementary Cellular Automata*, Complex Systems, **15** (1), 1–40.

[Cop98] JACK B. COPELAND (1998), *Turing's O-machines, Penrose, Searle, and the Brain*, Analysis, **58**, 128–138.

[Cot03] PAOLO COTOGNO (2003), *Hypercomputation and the Physical Church-Turing Thesis*, British Journal for the Philosophy of Science, **54**, 181–223.

[CP99] JACK B. COPELAND and DIANE PROUDFOOT (1999), *Alan M. Turing's forgotten ideas in Computer Science*, Scientific American, **253** (4), 98–103.

[Cra78] RICHARD E. CRANDALL (1978), *On the "3x + 1" problem*, Mathematics of computation, **32** (144), 1281–1292.

[Cur30] HASKEL B. CURRY (1930), *Grundlagen der kombinatorischen Logik (I and II)*, American Journal of Mathematics, **52** (3–4), 509–536, 789–834.

[Cur58] HASKELL B. CURRY (1958), *Calculuses and formal systems*, Dialectica, **12**, 249–273.

[Dav56] MARTIN DAVIS (1956), *A note on universal Turing machines*, in: [MS56a], 167–177.

[Dav57] ——— (1957), *The definition of Universal Turing machine*, Proceedings of the American Mathematical Society, **8** (6), 1125–1126.

[Dav58] ——— (1958), *Computability and Unsolvability*, McGraw-Hill, New York.

[Dav65a] ——— (1965), *Introduction to Post's [Pos65]*, in: [Dav65b], 338–339.

[Dav65b] ——— (1965), *The Undecidable. Basic papers on undecidable propositions, unsolvable problems and computable functions,* Raven Press, New York, corrected republication (2004), Dover publications, New York.

[Dav82] ——— (1982), *Why Gödel didn't have Church's thesis,* Information and Control, **54**, 3–24.

[Dav87] ——— (1987), *Mathematical Logic and the Origin of Modern Computers,* reprinted in [Her88], 149–174.

[Dav88] ——— (1988), *Influences of Mathematical Logic on Computer Science,* in: [Her88], 315 -326.

[Dav89] ——— (1989), *Emil post's contributions to computer science,* in: *Proceedings of the Fourth Annual Symposium on Logic in computer science, Pacific Grove, California.*, 134–137.

[Dav90] ——— (1990), *Is mathematical insight algorithmic?,* Behavioral and Brain Sciences, **13** (4), 659–660.

[Dav93] ——— (1993), *How subtle is Gödel's theorem? More on Roger Penrose,* Behavioral and Brain Sciences, **16**, 611–612.

[Dav94] ——— (1994), *Emil L. Post. His life and work,* in: [Pos94], xi–xviii.

[Dav95] ——— (1995), *Logic in the twenties,* The Bulletin of Symbolic Logic, **1** (3), 273–278.

[Dav01a] ——— (2001), *The early history of automated deduction,* in: ALAN ROBINSON and ANDREI VORONKOV (eds.), *Handbook of Automated Reasoning,* Elsevier, Amsterdam, 3–16.

[Dav01b] ——— (2001), *Engines of Logic: Mathematicians and the Origin of the Computer,* W.W. Norton and Company, New York, published in hardcover as "The Universal Computer. The Road from Leibniz to Turing".

[Dav04] ——— (2004), *The myth of hypercomputation*, in: [Teu04], 195–211.

[Dav05] ——— (2005), *What did Gödel believe and when did he believe it?*, The Bulletin of Symbolic Logic, **11** (2), 194–206.

[Dav06a] ——— (2006), *The Church-Turing Thesis: Consensus and Opposition*, in: [AB06], 125–132.

[Dav06b] ——— (2006), *Why there is no such discipline as hypercomputation*, Applied Mathematics and Computation, **178**, 4–7.

[Daw97] JOHN DAWSON (1997), *Logical Dilemmas: The Life and Work of Kurt Gödel*, AK Peters, Wellesley.

[dCD90] NEWTON C.A. DA COSTA and FRANCISCO A. DORIA (1990), *Undecidability and Incompleteness in Classical Mechanics*, International Journal of Theoretical Physics, **30** (8), 1041–1073.

[Ded32] RICHARD DEDEKIND (1930–32), *Gesammelte Mathematische Werke, vol. I-III*, Vieweg, Braunschweig.

[Ded72] ——— (1872), *Stetigkeiten und Irrationale Zahlen*, Vieweg, Braunschweig, also published as [Ded32], 315–334.

[Ded88] ——— (1888), *Was sind und was sollen die Zahlen*, Vieweg, Braunschweig, also published as [Ded32], 335–391.

[Deh11] MAX DEHN (1911), *Über unendliche diskontinuierliche Gruppen*, Mathematische Annalen, **71**, 116–144.

[Des47] RENÉ DESCARTES (1647), *Méditations Métaphysiques*, translated from Latin text (1641) by duc de Luynes (J.M. Beyssade (ed.), Flammarion, Paris, 1993).

[Dev89] ROBERT DEVANEY (1989), *An Introduction to Chaotic Dynamical Systems*, Addison-Wesley, New York.

[Eck80] JOHN PRESPER ECKERT (1980), *The Eniac*, in: [HMR80], 525–540.

[Eck87] ROGER ECKHARDT (1987), *Stan Ulam, John von Neumann and the Monte Carlo Method*, Los Alamos Science (Special Issue, Stanislaw Ulam 1909-1984), **15**, 131–137.

[EGW04] EUGENE EBERBACH, DINA GOLDIN and PETER WEGNER (2004), *Turing's ideas and models of computation*, in: [Teu04], 159–194.

[ELdlL92] DAVID EPSTEIN, SILVIO LEVY and RAFAEL DE LA LAVE (1992), *Letter from the editor*, Experimental Mathematics, **1** (1), 1–3, statement of the Philosophy of the journal "Experimental Mathematics", available at: http://www.math.ethz.ch/EMIS/journals/EM/expmath/philosophy.html.

[End98] HERBERT B. ENDERTON (1998), *Alonzo Church and the reviews*, Bulletin of Symbolic Logic, **4** (2), 172–180.

[End05] ——— (2005), *Alonzo Church: Life and work*, in: [End05], to appear.

[Eul61] LEONHARD EULER (1761), *Specimen de usu observationum in mathesi pura*, Novi Commentarii academiae scientiarum Petropolitanae, **6**, 185–230, opera Omnia, Series 1, vol. 2, 459–492.

[Fef88] SOLOMON FEFERMAN (1988), *Turing in the land of O(z)*, in: [Her88], 113–145.

[Fef95] ——— (1995), *Penrose's Gödelian Argment*, Psyche, **2** (7), 21–32, http://psyche.cs.monash.au/.

[FH03] LANCE FORTNOW and STEVE HOMER (2003), *A short History of Computational Complexity*, in: JOHN DAWSON DIRK VAN DALEN and AKI KANAMORI (eds.), *The History of Mathematical Logic*, North-Holland, Amsterdam.

[Fox63] JEREMY FOX (ed.) (1963), *Mathematical Theory of Automata, Microwave Research Institute Symposia Series*, vol. XII, Polytechnic Press, Brooklyn, NY.

[Fre79] GOTTLOB FREGE (1879), *Begriffschrift, eine der arithmetischen nachgebildete Formelsprache des reinen Denkens*, L.Nebert, Halle, translated in English in [vH67], 5–82.

[Fri57] RICHARD M. FRIEDBERG (1957), *Two recursively enumerable sets of incomparable degrees of unsolvability*, Proceedings of the National Academy of Sciences, (43), 236–238.

[Gan80] ROBIN GANDY (1980), *Church's Thesis and Principles for Mechanism*, in: J. BARWISE, H.J. KEISLER and K. KUNEN (eds.), *The Kleene Symposium*, North-Holland, Amsterdam, 123–148.

[Gan88] ——— (1988), *The confluence of ideas in 1936*, 55–111, published in [Her88].

[Gan06] PAUL GANNON (2006), *Colossus. Bletchley Park's greatest secret*, Atlantic Books, London.

[Gau01] CARL FRIEDRICH GAUSS (1801), *Disquisitiones Arithmeticae*, Fleischer, Leipzig.

[GBvN46] HERMAN H. GOLDSTINE, ARTHUR W. BURKS and JOHN VON NEUMANN (1946), *Preliminary Discussion of the Logical Design of an Electronic Computing Instrument*, report prepared for U. S. Army Ord. Dept. under Contract W-36-034-ORD-7481. Also published in [vN63] and [vN86].

[GG00] IVOR GRATTAN-GUINNESS (2000), *The search for mathematical roots, 1870-1940: Logics, set theories, and the foundations of mathematics from Cantor through Russell to Gödel*, Princeton University Press, Princeton.

[Göd30] KURT GÖDEL (1930), *Die Vollständigkeit der Axiome des logischen Funktionenkalküls*, Monatshefte für Mathematik and Physik, **37**, 349–360, translated in English in [vH67], 582–591.

[Göd31] ——— (1931), *Über formal unentscheidbare Sätze der Principia Mathematica und verwandtrer Systeme I*, Monatshefte für Mathematik und Physik, (38), 173–198, republished in English in [Dav65b], 5–38, and in [vH67], 596–616.

[Göd34] ——— (1934), *On undecidable propositions of formal mathematical systems.*, in: [Dav65b], 41–71, Based on mimeographed notes on lectures given by Gödel at the Institute for Advanced Study during the spring of 1934.

[Göd36] ——— (1936), *Uber die Länge der Beweise*, Ergebnisse eines mathematischen Kolloquium, **7**, 23–24, translated in [Dav65b], 82–83.

[Göd46] ——— (1946), *Remarks before the Princeton Bicentennial Conference on Problems in Mathematics*, in: [Dav65b], 84–88.

[Göd51] ——— (1951), *Some basic theorems on the foundations of mathematics and their implications. Gibbs lecture*, in: [Göd95], 304–323.

[Göd56] ——— (1956), *Letter to von Neumann, 20 March 1956*, in: [Göd03b], 373–377.

[Göd65] ——— (1965), *Postscriptum to [Göd34]*, in: [Dav65b], 71–73.

[Göd95] ——— (1995), *Collected Works III. Unpublished Essays and Lectures*, Oxford University Press, Oxford.

[Göd03a] ——— (2003), *Collected Works V: Correspondence A–G*, Oxford University Press, Oxford.

[Göd03b] ——— (2003), *Collected Works V: Correspondence H–Z*, Oxford University Press, Oxford.

[Gol72] HERMAN H. GOLDSTINE (1972), *The Computer from Pascal to von Neumann*, Princeton University Press, Princeton.

[Goo80] IRVING J. GOOD (1980), *Pioneering work on computers at Bletchley,* in: J.HOWLETT and G.-C. ROTA (eds.), *A history of Computing in the Twentieth Century,* Academic Press, New York, 31–45.

[Gre82] ULF GRENANDER (1982), *Mathematical experiments on the Computer,* Pure and Applied Mathematics, Academic Press, New York.

[GvN46] HERMAN H. GOLDSTINE and JOHN VON NEUMANN (1946), *On the principles of large scale computing machines,* in: [vN63], 1–32. Also reprinted in [vN86], 317–348. This paper was never published in a journal. It contains material given by von Neumann in a given number of lectures in particular one at a meeting on May 15, 1946 of the Mathematical Computing Advisory Panel, Office of Research and Inventions, Navy Department, Washington D.C.

[GvN47] ——— (1947), *Planning and Coding of Problems for an Electronic Computing Instrument. Part I,* report prepared for U. S. Army Ord. Dept. under Contract W-36-034-ORD-7481. Also published in [vN63].

[GvN48a] ——— (1948), *Planning and Coding of Problems for an Electronic Computing Instrument. Part II,* report prepared for U. S. Army Ord. Dept. under Contract W-36-034-ORD-7481. Also published in [vN63].

[GvN48b] ——— (1948), *Planning and Coding of Problems for an Electronic Computing Instrument. Part III,* report prepared for U. S. Army Ord. Dept. under Contract W-36-034-ORD-7481. Also published in [vN63].

[Hay86] BRIAN HAYES (1986), *Theory and Practice: Tag-You're it,* Computer Language, 21–28.

[Hay6a] ——— (1996a), *A question of Numbers,* American Scientist, **84**, 10–14, available at:

http://www.americanscientist.org/amsci/issues/Comsci96/compsci96-01.html.

[Her88] Rolf Herken (ed.) (1988), *The Universal Turing machine*, Oxford University Press, Oxford, republication (1994), Springer Verlag, New York.

[Hey59] Arend Heyting (ed.) (1959), *Constructivity in Mathematics. Proceedings of the Colloquium held at Amsterdam, 1957*, North-Holland, Amsterdam.

[Hil99] David Hilbert (1899), *Grundlagen der Geometrie*, B.G. Teubner, Leipzig, authorized translation by E.J. Townsend, The foundations of Geometry, 1950, La Salle, Illinois.

[Hil01] ——— (1901), *Mathematische Probleme*, Archiv für Mathematik und Physik, **1**, 44–63, 213–237, lecture delivered before the International Congress of Mathematicians at Paris in 1900, Republished in [Hil32], vol. III, 290–297.

[Hil05] ——— (1905), *Über die Grundlagen der Logik und der Arithmetik*, in: *Verhandlungen des Dritten Internationalen Mathematiker-Kongresses in Heidelberg von 8. bis 13. August 1904*, Teubner, Leipzig, 174–185, translated in English in [vH67], 130–138.

[Hil26] ——— (1926), *Über das Unendliche*, Mathematische Annalen, **95**, 161–190, translated in English in [vH67], 369–392.

[Hil28] ——— (1928), *Die Grundlagen der Mathematik*, Abhandlungen aus dem mathematischen Seminar der Hamburgischen Universität, **6**, 65–85, translated in English in [vH67], 464–479.

[Hil30] ——— (1930), *Naturerkennen und Logik*, Naturwissen, **18**, 959–963, republished in [Hil32], vol. III, 278–385.

[Hil32] ——— (1932), *Gesammelte Abhandlungen*, Springer Verlag, Berlin.

[HMR80] JOHN HOWLETT, NICOLAS METROPOLIS and GIAN-CARLO ROTA (eds.) (1980), *A History of Computing in the Twentieth Century*, Academia Press, New York, proceeding of the International Research Conference on the History of Computing, Los Alamos, 1976.

[Hod83] ANDREW HODGES (1983), *Alan M. Turing. The enigma*, Burnett Books, London, republication (1992), 2nd edition, Vintage, London.

[Hod04] ——— (2004), *What would Alan M. Turing have done after 1954?*, in: [Teu04], 43–57.

[Hoo65] PHILIP K. HOOPER (1965), *Post normal systems: the unrestricted halting problem*, Notices of the American MAthematical Society, **12** (3), 371.

[Hoo6a] ——— (1966a), *The undecidability of the Turing machine immortality problem*, The Journal of Symbolic Logic, **31** (2), 219–234.

[Hoo6b] ——— (1966b), *The immortality problem for Post normal systems*, Journal of the ACM, **13**, 594–599.

[Hop81] CAPTAIN GRACE HOPPER (1981), *Keynote address*, in: [Wex81], 7–20.

[HR00] ULF HASHAGEN and RAUL ROJAS (eds.) (2000), *The first computers – History and Architectures*, History of Computing, MIT Press, Cambridge, MA.

[HS65] JURIS HARTMANIS and RICHARD STEARNS (1965), *On the computational complexity of algorithms*, Transactions of the American Mathematical Society, (117), 285–306.

[Hug73] CHARLES E. HUGHES (1973), *Many-One Degrees Associated With Problems of Tag*, The Journal of Symbolic Logic, **38** (1), 1–17.

[Ike58] SHINICHI IKENO (1958), *A 6-symbol 10-state universal Turing machine*, Proceedings Institute of Electrical Communications.

[Jon82] JAMES P. JONES (1982), *Universal Diophatine Equation*, The journal of symbolic logic, **47** (3), 549–571.

[Kal59] LÁSZLÓ KALMÁR (1959), *An argument against the Plausability of Church's Thesis*, in: [Hey59], 72–79.

[Kan96] AKIHIRO KANAMORI (1996), *The mathematical development of Set Theory. From Cantor to Cohen.*, The bulletin of symbolic logic, **2** (1), 1–71.

[Kar72] RICHARD KARP (1972), *Reducibility among combinatorial problems*, in: RAYMOND E. MILLER and JAMES W. THATCHER (eds.), *Complexity of Computer Computations*, Plenum Press, New York, 85–103.

[Kas92] FRANTISEK KASCAK (1992), *Small Universal One-state Linear Operator Algorithm*, in: L.M. HAVEL and V. KOUBEK (eds.), *Mathematical foundations of Computer Science, Lecture notes in Computer Science*, vol. 629, 327–335.

[Kle35a] STEPHEN C. KLEENE (1935), *A theory of positive integers in formal logic. Part I*, American Journal of Mathematics, **57** (1), 153–173.

[Kle35b] ——— (1935), *A theory of positive integers in formal logic. Part II*, American Journal of Mathematics, **57** (2), 219–244.

[Kle36a] ——— (1936), *General Recursive Functions of Natural Numbers*, Mathematische Annalen, **112**, 727–742.

[Kle36b] ——— (1936), *λ-definability and recursiveness*, Duke Mathematical Journal, **2**, 240–253.

[Kle38] ——— (1938), *On notation for ordinal numbers*, Journal of Symbolic Logic, **3** (4), 150–155.

[Kle43] ——— (1943), *Recursive predicates and quantifiers*, Transactions of the American Mathematical Society, **53** (1), 41–73.

[Kle52] ——— (1952), *Introduction to Metamathematics*, Van Nostrand, New York, 13th reprint, 2000, Bibliotheca Mathematica, North-Holland: Amsterdam.

[Kle79] FELIX KLEIN (1979), *Development of mathematics in the 19th century (English ed.)*, Mathematical Science Press, Brookline, Massachusetts, translated by M. Ackerman.

[Kle81a] STEPHEN C. KLEENE (1981), *Origins of recursive function theory*, Annals of the history of computing, **3**, 52–67.

[Kle81b] ——— (1981), *The theory of recursive functions, approaching its centennial*, Bulletin of the american mathematical society, **5**, 43–60.

[Kle87] ——— (1987), *Reflections on Church's thesis*, Notre Dame Journal of formal logic, **28** (4), 490–498.

[Knu62] DONALD E. KNUTH (1962), *A History of Writing Compilers*, Computers and Automaton, **11** (12), 8–18.

[Kop81] RONA J. KOPP (1981), *The Busy Beaver Problem*, Mathematical sciences, State University of New York at Binghamton, girl name of Rona Machlin.

[KP80] DONAL E. KNUTH and LUIS TRABB PARDO (1980), *Early development of programming languages*, in: [HMR80], 197–274.

[KR35] STEPHEN C. KLEENE and BARKLEY J. ROSSER (1935), *The inconsistency of certain formal logics*, The annals of Mathematics, **36** (3), 630–636.

[KR01] MANFRED KUDLEK and YURII ROGOZHIN (2001), *New Small Universal Circular Post Machines*, in: *Fundamentals of Computation Theory : 13th International Symposium, FCT 2001, Riga, Latvia, August 22-24, 2001.*, *Lecture notes in computer science*, vol. 2138, 217–226.

[KR02] ——— (2002), *A universal Turing machine with 3 states and 9 symbols*, in: G. ROZENBERG W. KUICH and A. SALOMAA (eds.), *Proc. 5th International Conference on Developments in Language Theory, Lectore Notes in Computer Science*, vol. 2295, 311–318.

[Krä88] SYBILLE KRÄMER (1988), *Symbolische Maschinen: Die Idee der Formalisierung in geschichtlichem Abriss*, Wissenschaftliche Buchgesellschaft, Darmstadt.

[Lag85] JEFFREY C. LAGARIAS (1985), *The 3x + 1 problem and its generalizations*, American Mathematical Monthly, **92** (1), 3–23, available at: http://www.cecm.sfu.ca/organics/papers/lagarias/paper/html/paper.html.

[Lag95] ——— (1995), *The 3x+1 problem and its generalisations*, in: J. BORWEIN ET AL. (ed.), *Organic Mathematics. Proceedings Workshop Simon Fraser University, Burnaby*, AMS, Providence, available at http://www.cecm.sfu.ca/organics/papers/lagarias.

[Lag06] ——— (2006), *The 3x + 1 problem: An annotated bibliography (1963–2000)*, available at: http://arxiv.org/PS-cache/math/pdf/0608/0608208.pdf.

[Lam86] J.H. LAMBERT (1786), *Theorie der Parallellinien*, Leipziger Magazin für reine und angewandte Mathematik, **1** (2 & 3), 137–164 (2) & 325–358 (3), hrsg. von Bernouilli, J. und Hindenburg, C.F.

[Leh33] DERRICK H. LEHMER (1933), *Numerical Notations and Their Influence on Mathematics*, Mathematics News Letter, **7** (6), 8–12.

[Leh51] ——— (1951), *Mathematical methods in large scale computing units*, in: *Proceedings of Second Symposium on Large-Scale Digital Calculating Machinery, 1949*, Harvard University Press, Cambridge, Massachussetts, 141–146.

[Leh63] ——— (1963), *Some high-speed logic*, in: *Experimental Arithmetic, High Speed Computing and MAthematics, Proceedings of Symposia in Applied Mathematics*, vol. 15, 141–376.

[Leh74] ——— (1974), *The influence of computing on research in number theory*, in: J. P. LaSalle (ed.), *The Influence of Computing on Mathematical Research and Education, Proceedings of Symposia in Applied Mathematics*, vol. 20, 3–12.

[Leh80] ——— (1980), *A history of the sieve process*, in: [HMR80], 445–456.

[Lew18] Clarence I. Lewis (1918), *A survey of Symbolic Logic*, University of California Press, Berkeley.

[LLMS62] Derrick H. Lehmer, Emma Lehmer, W.H. Mills and John L. Selfridge (1962), *Machine proof of a theorem on Cubic residues*, Mathematics of computation, **16** (80), 407–415.

[LMSS56] Karel De Leeuw, Edward F. Moore, Claude E. Shannon and Norman Shapiro (1956), *Computability by probabilistic machines*, [MS56a], 183–212.

[Lor55] Paul Lorenzen (1955), *Einführung in die operative Logik und Mathematik, Grundlehren der mathematischen Wissenschaften*, vol. 78, Springer Verlag, Berlin.

[LR65] Shen Lin and Tibor Rádo (1965), *Computer studies of Turing Machine Problems*, Journal of the ACM, **12** (2), 196–212.

[LS06] Amy N. Langville and William J. Stewart (eds.) (2006), *MAM 2006: Markov Anniversary Meeting*, Boson Books.

[Luc61] John R. Lucas (1961), *Minds, machines and Gödel*, Philosophy, **36**, 112–127.

[Mah00] Michael S. Mahoney (2000), *The Structures of Computation*, in: [HR00], 17–31.

[Man98] Paolo Mancosu (1998), *From Brouwer to Hilbert: The debate on the foundations of mathematics in the 1920s*, Oxford University Press, Oxford.

[Man03] ——— (2003), *The Russellian influence on Hilbert and his school*, Synthese, **137** (1–2), 59–101.

[Mar54] ANDREI A. MARKOV (1954), *Theory of Algorithms (russian)*, Academy of sciences of the USSR, Moscow, Leningrad, translated in English by Jacques J. Schorr-Kon and PST Staff, Israel Program for Scientific Translations, Jerusalem, 1961.

[Mar76] DAVID MARR (1976), *Artificial Intelligence – A personal view*, artificial Intelligence Project – RLE and MIT Computation Center, memo 355.

[Mar84] GEORGE MARSAGLIA (1984), *A current view of random number generators*, in: *Proceedings of the 16th Symposium on the Interface between Computer Science and Statistics (Atlanta)*, Elsevier, Amsterdam.

[Mar96] ——— (1996), *DIEHARD: A battery of tests of randomness*, available at: http://stat.fsu.edu/pub/diehard/.

[Mar97] ——— (1997), *Instructions for using DIEHARD: a battery of tests of randomness.*, available at: http://stat.fsu.edu/pub/diehard/.

[Mar00] MAURICE MARGENSTERN (2000), *Frontier between Decidability and Undecidability: A survey*, Theoretical Computer Science, **231** (2), 217–251.

[Mas67] SERGEI. J. MASLOV (1967), *The concept of strict representability in the general theory of calculi.*, Trudy Matematicheskogo Instituta imeni V.A. Steklova, **93**, 3–42, english translation in: American Mathematical Society Translations Series 2.

[Mas4a] ——— (1964a), *Certain properties of E.L. Post's apparatus of canonical systems (Russian)*, Trudy Matematicheskogo Instituta imeni V.A. Steklova, (72), 57–68, translated in English in *American Mathemtical Society Translations, series 2*, vol. 97, nr. 2, 1970, 1 – 14.

[Mas4b] ——— (1964b), *On E. L. Post's 'Tag' Problem. (Russian),* Trudy Matematicheskogo Instituta imeni V.A. Steklova, (72), 5–56, english translation in: American Mathematical Society Translations Series 2, 97, 1–14, 1971.

[Mat70] YURI MATIYASEVICH (1970), *Solution of the Tenth Problem of Hilbert,* Matematikai Lapok, (21), 83–87.

[Mau80] JOHN W. MAUCHLY (1980), *The Eniac,* in: [HMR80], 541–550.

[McC60] JOHN MCCARTHY (1960), *Recursive Functions of Symbolic Expressions and Their Computation by Machine, Part 1,* Communications of the ACM, **3** (3), 184–195.

[McC81] ——— (1981), *History of Lisp,* in: [Wex81], 173–183.

[McC99] SCOTT MCCARTNEY (1999), *ENIAC: The Triumphs and Tragedies of the World's First Computer,* Walker & Co, New York.

[Men63] ELLIOTT MENDELSON (1963), *On some recent criticism of Church's Thesis,* Notre Dame Journal of Formal Logic, **IV** (3), 201–205.

[Met87] NICHOLAS METROPOLIS (1987), *The Beginning of the Monte Carlo Method,* Los Alamos Science (Special Issue, Stanislaw Ulam 1909-1984), **15**, 125–130.

[Mic93] PASCAL MICHEL (1993), *Busy beaver competition and Collatz-like problems,* Archive for Mathematical Logic, **32** (5), 351–367.

[Mic04] ——— (2004), *Small Turing Machines and generalized busy beaver competition,* Theoretical Computer Science, **326** (1–3), 45–56.

[Min61] MARVIN MINSKY (1961), *Recursive unsolvability of Post's problem of tag and other topics in the theory of Turing machines,* Annals of Mathematics, **74**, 437–455.

[Min62a] ——— (1961/1962?), *A simple direct proof of Post's Normal Form Theorem*, artificial Intelligence Project – RLE and MIT Computation Center, memo 44.

[Min62b] ——— (1962), *Size and Structure of Universal Turing Machines using Tag systems: a 4-symbol 7-state machine*, Proceedings Symposia Pure Mathematics, American Mathematical Society, **5**, 229–238.

[Min67] ——— (1967), *Computation. Finite and Infinite Machines*, Series in Automatic Computation, Prentice Hall, Englewood Cliffs, New Jersey.

[Min62] ——— (1961/62?), *Universality of (p = 2) Tag systems and a 4 symbol 7 state Universal Turing Machine*, artificial Intelligence Project – RLE and MIT Computation Center, memo 33.

[Mol05] LIESBETH DE MOL (2005), *Study of Fractals derived from IFS-fractals through metric procedures*, Fractals, **13** (3), 237–244.

[Mol06a] ——— (2006), *Closing the circle: An analysis of Emil Post's early work.*, The Bulletin of Symbolic Logic, **12** (2), 267–289.

[Mol06b] ——— (2006), *Facing the Computer. Some techniques to understand technique (abstract)*, in: COLIN SCHMIDT (ed.), *International Conference on Computers and Philosophy (I-C&P), Laval, France, 3–5 May, 2006*.

[Mol06c] ——— (2006), *Questions concerning the Usefulness of Small Universal Systems (abstract)*, in: [AB06], 303.

[Mol07] ——— (2007), *Tag systems and Collatz-like functions*, submitted to Theoretical Computer Science (under revision).

[MP43] WARREN S. MCCULLOUGH and WALTER H. PITTS (1943), *A logical Calculus of the Ideas Immanent in Nervous Activity*, Bulletin of Mathematical Biophysics, **5**, 115–133.

[MP47] ——— (1947), *How we know universals: The perception of auditory and visual forms*, Bulletin of Mathematical Biophysics, **9**, 127–147.

[MP95] MAURICE MARGENSTERN and LUDMILA PAVLOTSKAYA (1995), *Deux machines de Turing universelles à au plus deux instructions gauches*, Comptes rendus de l'Académie des sciences. Séries 1, Mathematique, **320** (11), 1395–1400.

[MRvN50] NICHOLAS METROPOLIS, GEORGE REITWIESNER and JOHN VON NEUMANN (1950), *Statistical Treatment of Values of First 2000 Decimal Digits of e and of π Calculated on the ENIAC*, Mathematical tables and other aids to Computations, **4** (30), 109–112, also published in [vN63].

[MS56a] JOHN MCCARTHY and CLAUDE E. SHANNON (eds.) (1956), *Automata Studies*, no. 34 in Annals of Mathematics Studies, Princeton University Press, Princeton, second Printing 1958.

[MS56b] JOHN MCCARTHY and CLAUDE E. SHANNON (1956), *Preface*, in: [MS56a], v–viii.

[MS90] RONA MACHLIN and QUENTIN F. STOUT (1990), *The complex behaviour of simple machines*, Physica D, **42**, 85–98.

[Muc56] A.A. MUCHNIK (1956), *Nerazreshimost' problemy svodimosti algoritmov (Negative answer to the problem of reducibility of the theory of algorithms)*, Doklady Akademii Nauk SSSR, (108), 194–197.

[Mur98] ROMAN MURAWSKI (1998), *E.L. Post and the development of mathematical logic and recursion theory*, Studies in Logic, Grammar and Rhetoric, **2** (15), 17–30.

[MZ93] GEORGE MARSAGLIA and ASAD ZAMAN (1993), *Monkey Tests for Random Number Generators*, Computers and Mathematics with Applications, **26** (9), 1–10.

[Nea06] TURLOUGH NEARY (2006), *Small Polynomial time universal Turing machines*, in: *Proceedings of MFCSIT 2006*, Cork, Ireland.

[Neu06] HANS NEUKOM (2006), *The second life of ENIAC (+ web extras)*, IEEE Annals of the history of computing, **28** (2), 4–16.

[NW06a] TURLOUGH NEARY and DAMIEN WOODS (2006), *On the time complexity of 2-tag systems and small universal Turing machines*, in: *Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science*, 439–448.

[NW06b] ——— (2006), *P-completeness of cellular automaton rule 110*, in: *International Colloquium on Automata Languages and Programming (ICALP), Lecture Notes in Computer Science*, vol. 4051, 132–143.

[NW06c] ——— (2006), *Remarks on the computational complexity of small universal Turing machines*, in: *Proceedings of MFCSIT 2006*, Cork, Ireland, 334–338.

[NW06d] ——— (2006), *Small fast universal Turing machines*, Technical report NUIM-CS-2005-TR-11, Department of Computer Science, NUI Maynooth, accepted for publication in Theoretical Computer Science.

[oLAA43] COUNTESS OF LOVELACE ADA AUGUSTA (1843), *Sketch of the Analytical Engine Invented by Charles Babbage, With notes upon the Memoir by the Translator Ada Augusta, Countess of Lovelace, Taylor's Scientific Memoirs*, vol. 3.

[Ove71] ROSS OVERBEEK (1971), *Representation of many-one degrees by decision problems associated with Turing machines*, The Journal of Symbolic Logic, **36**, 706, abstract.

[Pag70] DAVID PAGER (1970), *The categorization of Tag systems in terms of decidability*, Journal of the London Mathematical Society, **2** (2), 473–480.

[Pav73] LUDMILA PAVLOTSKAYA (1973), *Solvability of the halting problem for certain classes of Turing machines (in Russian),* Mathematical Notes Academy of Science USSR,, **13** (6), 537–541.

[Pav78] ——— (1978), *Sufficient conditions for the halting problem decidability of Turing machines*, Avtomaty i Mashiny, 91–118.

[Pea89] GIUSEPPE PEANO (1889), *Arithmetices principia, nova methodo exposita*, Bocca, Turin, translated in English in [vH67], 83–97.

[Pen89] ROGER PENROSE (1989), *The Emperor's new Mind. Concerning Computers, minds, and the Laws of Physics.*, Oxford University Press, Oxford.

[Pen94] ——— (1994), *Shadows of the Mind. A search for the missing science of consciousness.*, Oxford University Press, Oxford.

[PER79] MARIAN B. POUR-EL and IAN RICHARDS (1979), *A computable ordinary differential equation which possesses no computable solution*, Annals of Mathematical Logic, **17**, 61–90.

[Pét59] RÓSZA PÉTER (1959), *Rekursivität und Konstruktivität*, in: [Hey59], 226–233.

[PJS92] HEINZ-OTTO PEITGEN, HARTMUT JÜRGENS and DIETMAR SAUPE (1992), *Chaos and Fractals. New Frontiers of Science*, Springer Verlag, New York.

[Por60] J. PORTE (1960), *Quelques pseudo-paradoxes de la "calculabilité effective"*, in: *Actes de deuxième Congrès International de Cybernetique*, Namur, Belgium, 332–334.

[Pos21a] EMIL LEON POST (1921), *Introduction to a general theory of elementary propositions.*, American Journal of Mathematics, (43), 163–185.

[Pos21b] ——— (1921), *On a simple class of deductive systems*, Bulletin of the American Mathematical Society, **27**, 396–397.

[Pos36] ——— (1936), *Finite Combinatory Processes - Formulation 1,* The Journal of Symbolic Logic, **1** (3), 103–105, also published in [Dav65b], 289–291.

[Pos40] ——— (1940), *Polyadic Groups,* Transactions of the American mathematical society, **48**, 208–350.

[Pos43] ——— (1943), *Formal Reductions of the General Combinatorial Decision Problem,* American Journal of Mathematics, **65** (2), 197–215.

[Pos44] ——— (1944), *Recursively Enumerable Sets of Positive Integers and their Decision Problems,* Bulletin of The American Mathematical Society, (50), 284–316, also published in [Dav65b], 305–337.

[Pos46] ——— (1946), *A Variant of a recursively Unsolvable Problem,* Bulletin of the American Mathematical Society, (52), 264–268.

[Pos47] ——— (1947), *Recursive Unsolvability of a problem of Thue,* The Journal of Symbolic Logic, (12), 1–11, also published in [Dav65b], 293–303.

[Pos53a] ——— (1953), *A Necessary Condition for Definability for Transfinite von Neumann-Gĺodel Set Theory Sets, with an Application to the Problem of the Existence of a Definable Well-Ordering of the Continuum,* Bulletin of the American Mathematical Society, **59**, 246.

[Pos53b] ——— (1953), *Solvability, Definability, Provability; History of an error,* Bulletin of the American Mathematical Society, **59**, 245–246.

[Pos65] ——— (1965), *Absolutely unsolvable problems and relatively undecidable propositions - account of an anticipation,* in: [Dav65b], 340–433. Also published in [Pos94].

[Pos94] ——— (1994), *Solvability, Provability, Definability: The collected works of Emil L. Post,* Birkhauser, Boston, edited by Martin Davis.

[Rád62] TIBOR RÁDO (1962), *On non-computable functions,* The Bell System Technical Journal, **41** (3), 877–884.

[Rád63] ——— (1963), *On a simple source for non-computable functions,* in: [Fox63], 75–81.

[Ran80] BRIAN RANDELL (1980), *The Colossus,* in: [HMR80], 47–92.

[Rei50] GEORGE W. REITWIESNER (1950), *An ENIAC determination of pi and e to more than 2000 decimal places,* Mathematical Tables and Other Aids to Computation, **4** (29), 11–15.

[Rog82] YURII ROGOZHIN (1982), *Seven Universal Turing Machines (in Russian),* Mat. Issledovaniya, **69**, 76–90.

[Rog96] ——— (1996), *Small universal Turing Machines,* Theoretical Computer Science, **168**, 215–240.

[Roj98] RAÚL ROJAS (1998), *How to make Zuse's Z3 a universal computer,* IEEE Annals of the History of Computing, **20** (3), 51–54, also available at www.zib.de/zuse.

[Roj00] ——— (2000), *Die Architektur der Rechenmaschinen Z1 und Z3,* available at www.zib.de/zuse.

[Rojnd] ——— (n.d.), *Plankalkül,* available at www.zib.de/zuse.

[Ros35] BARKLEY J. ROSSER (1935), *A mathematical logic without variables,* Annals of Mathematics (2nd Series), **36**, 127–150.

[Ros82] ——— (1982), *Highlights of the history of the lmabda-calculus,* in: *Proceedings of the 1982 ACM Symposium on LISP and functional programming,* 216–225.

[Ros84] ——— (1984), *Highlights of the history of the lambda-calculus,* Annals of the history of computing, **6** (4), 337–349.

[Rub88] FRANK RUBIN (1988), *The Cryptographic Uses of Post Tag Systems.*, Cryptologia, **12** (1), 25–33.

[Rus02] BERTRAND RUSSELL (1902), *Letter to Frege*, Published in [vH67], 124–125.

[Rus03] ——— (1903), *Principles of Mathematics*, Cambridge University Press, Cambridge.

[Rus08] ——— (1908), *Mathematical Logic as Based on the Theory of Types*, American Journal of Mathematics, (30), 222–262, translated in English in [vH67], 153–182.

[RV96] YURII ROGOZHIN and SERGEY VERLAN (1996), *On the rule complexity of universal tissue P systems*, in: *Proceedings of the 6th International Workshop on Membrane Computing (Vienna)*, 510–516.

[RW13] BERTRAND RUSSELL and ALFRED NORTH WHITEHEAD ((1910, 1912, 1913)), *Principia Mathematica, vol. I-III*, Cambridge University Press, Cambridge.

[Sad98] ZENON SADOWSKI (1998), *On the development of Emil Post's ideas in structural complexity theory*, Studies in Logic, Grammar and Rhetoric, **2** (15), 55–59.

[SB96] WILFRIED SIEG and JOHN BYRNES (1996), *K-graph machines: generalizing Turing's machines and arguments*, in: P. HAJEK (ed.), *Gödel '96, Lecture Notes in Logic*, vol. 6, 98–119.

[Sca63] BRUNO SCARPELLINI (1963), *Zwei unentscheidbare Probleme in der Analysis*, Zeitschrifr fur Mathematische LogiK und Grundlagen der Mathematik, **9**, 265–289, republished and translated in English as "Two Undecidable Problems of Analysis", Minds and Machines 13, 49Ű77, 2003.

[Sch24] MOSES SCHÖNFINKEL (1924), *Über die Bausteine der mathematischen Logik*, Mathematische Annalen, **92**, 305–316, republished and translated in [vH67], 357–366.

[SE95] P. STÄCKEL and F. ENGEL (1895), *Die Theorie der Parallellinien von Euklid bis auf Gauss*, Teubner, Leipzig.

[Sha38] CLAUDE E. SHANNON (1938), *A symbolic analysis of relay and switching circuits*, Transactions of the American Institute of Electrical Engineers, **57**, 713–723.

[Sha48] ——— (1948), *A mathematical theory of communication*, Bell System Technical Journal, **27**, 379–423 en 623–656.

[Sha56] ——— (1956), *A Univeral Turing Machine with Two Internal States*, in: [MS56a], 157–165.

[She65] JOHN C. SHEPHERDSON (1965), *Machine configuration and word problems of given degree of unsolvability*, Zeitschrift für Mathematische Logik und Grundlagen der Mathematik, **11**, 149–175.

[She96] JAMES B. SHEARER (1996), *Periods of strings (Letter to the editor)*, American Scientist, **86**, 207.

[Sho96] JOSEPH R. SHOENFIELD (1996), *The Mathematical Work of S.C. Kleene*, The Bulletin of Symbolic Logic, **1** (1), 9–43.

[Sho97] PETER SHOR (1997), *Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer*, SIAM journal on computing, **26** (5), 1484 – 1509.

[Sie94] WILFRIED SIEG (1994), *Mechanical procedures and mathematical experience*, in: *Mathematics and Mind*, Oxford University Press, Oxford, 71–117.

[Sie97] ——— (1997), *Step by Recursive Step: Church's analysis of Effective Calculability*, Bulletin of Symbolic Logic, **3** (2), 154–180.

[Sie99] ——— (1999), *Hilbert's programme 1917–1922*, Bulletin of Symbolic Logic, **5** (1), 1–44.

[Sie05] ——— (2005), *Only two letters: The correspondence between Herbrand and Gödel*, The Bulletin of Symbolic Logic, **11** (2), 172–184.

[Sie06] ——— (2006), *Computability Theory*, available at: http://www.phil.cmu.edu/summerschool/2006/Sieg/computability-theory.pdf.

[Sie07] ——— (2007), *Church Without Dogma. Axioms for computability*, available at: http://www.hss.cmu.edu/philosophy/faculty-sieg.php.

[Sip92] MICHAEL SIPSER (1992), *The History and Status of the P versus NP Question*, in: *Proceedings of the 24th Annual ACM Symposium on Theory of Computing*, 603–618.

[Sko28] THORALF SKOLEM (1928), *Über die mathematische Logik*, Norsk matematisk tidsskrift, **10**, 125–142, translated in English in [vH67], 512–524.

[Smu61] RAYMON M. SMULLYAN (1961), *Elementary formal systems*, Journal of the Mathematical Society of Japan, **13**, 38–44.

[Soa96] ROBERT L. SOARE (1996), *Computability and Recursion*, The bulletin of Symbolic Logic, **2** (3), 284–321.

[Sta04] MIKE STANNETT (2004), *Hypercomputational models*, in: [Teu04], 135–157.

[Sti04] JOHN STILLWELL (2004), *Emil Post and His Anticipation of Gödel and Turing*, Mathematics Magazine, **77** (1), 3–14.

[Sut03] KLAUS SUTNER (2003), *Cellular automata and intermediate degrees*, Theoretical Computer Science, **296** (2), 365–375.

[Sut05] ——— (2005), *Universality and Cellular Automata,* in: *Machines, Computations, and Universality: 4th International Conference (2004), Lecture Notes in Computer Science,* vol. 3354, 50–59.

[Swi04] JONATHAN SWINTON (2004), *Watching the daisies grow: Turing and Fibonacci Phyllotaxis,* in: [Teu04], 477–497.

[Tar35] ALFRED TARSKI (1935), *Der Wahrheitsbegriff in den formalisierten Sprachen,* Studia Philosophica, **1**, 261–405, translated in German from Polish book *Projęcie prawdy w językach nauk dedukcyjnych,* 1933. Republished in Karel Berka, Lothar Kreiser (ed.), *Logik Texte. Kommentierte Auswahl zur Geschichte der modernen Logik,* Akademie Verlag, Berlin, 447–559.

[Teu04] CHRISTOPH TEUSCHER (ed.) (2004), *Alan M. Turing: Life and Legacy of a Great Thinker,* Springer Verlag, Berlin.

[Thu14] AXEL THUE (1914), *Probleme über Veränderungen von Zeichenreihen nach gegebenen Regeln,* Videnskaps-Akademi i Kristiani Skrifter, **10**, 3–33.

[Tra88] BORIS A. TRAKHTENBROT (1988), *Comparing the Church and Turing Approaches: Two Prophetical Messages,* in: [Her88], 603–630.

[Tur37] ALAN M. TURING (1936–37), *On computable numbers with an application to the Entscheidungsproblem,* Proceedings of the London Mathematical Society, (42), 230–265, a correction to the paper was published in the same journal, vol. 43, 1937, 544–546. Both were published in [Dav65b], 116–151.

[Tur34] ——— (1934), *On the Guassian error function,* unpublished fellowship Dissertation, King's College Library, Cambridge.

[Tur35] ——— (1935), *Equivalence of Left and Right Almost periodicity,* Journal of the London Mathematical Society, **10**, 284–285.

[Tur37] ——— (1937), *Computability and λ-definability*, The journal of Symbolic Logic, **2** (4), 153–163.

[Tur39] ——— (1939), *Systems of logic based on ordinals*, Proceedings of the London Mathematical society, **45**, 161–228, also published in [Dav65b], 155–222.

[Tur45] ——— (1945), *Proposals for Development in the Mathematics Division of an Automatic Computing Engine (ACE). Report to the NAtional Physics Laboratory*, in: [CD86], 20–105. Also published in [Tur92a], 87–105.

[Tur47] ——— (1947), *Lecture to the London Mathematical Society on 20 February 1947.*, in: [CD86], 106–124. Also published in [Tur92a], 87–105.

[Tur50] ——— (1950), *Computing machinery and Intelligence*, Mind, **59**, 433–460, also published in [Tur92a], 133–159.

[Tur51a] ——— (1951), *Intelligent machinery, a heretical theory*, lecture given at Manchester (2 versions, one numbered 1–10, the other numbered 96–101). Available at the digital Turing archive at http://www.turingarchive.org, archive number AMT/B/4.

[Tur51b] ——— (1951), *Programmers' handbook for Manchester electronic Computer. Mark II. with errata sheets and Programme sheets*, available at the digital Turing archive at http://www.turingarchive.org, archive number AMT/B/32.

[Tur52a] ——— (1952), *Can automatic calculating machines be said to think?*, transcript of a broadcast discussion transmitted on BBC Third Programme, 14 and 23 Jan. 1952, between M.H.A. Newman, AMT, Sir Geoffrey Jefferson and R.B. Braithwaite, Available at: http://www.turingarchive.org, archive number AMT/B/6.

[Tur52b] ——— (1952), *The Chemical Basis for Morphogenesis*, Philosophical transactions of the Royal Society of London, **237** (641), 37–72, also published in [Tur92b], 1–35.

[Tur53] ——— (1953), *Some calculations of the Riemann-Zeta function*, Proceedings of the London Mathematical Society, **3** (3), 99–117.

[Tur69] ——— (1969), *Intelligent Machinery*, in: B. MELTZER and D. MICHIE (eds.), *Machine Intelligence*, vol. 5, Edinburgh University Press, Edinburgh, 3–23, report written in 1948, National Physics Laboratory, Also published in [Tur92a], 107–127.

[Tur92a] ——— (1992), *Collected Works of A.M.Turing. Mechanical Intelligence.*, North-Holland, Amsterdam.

[Tur92b] ——— (1992), *Collected Works of A.M.Turing. Morphogenesis.*, North-Holland, Amsterdam.

[Ula80] STANISLAW M. ULAM (1980), *von Neumann: The Interaction of Mathematics and Computing*, in: [HMR80], 93–99.

[Usp53] VLADIMIR A. USPENSKY (1953), *Gödel's theorem and the theory of algorithms (Russian)*, Dokl. Akad. Nauk USSR, **91**, english translation in: American Mathematical Society Translations Series 2, 23, 103–107, 1963.

[Usp83] ——— (1983), *Post's machine*, Mir Publishers (Little Mathematics Libarary), Moscow, originally published in 1979. Translated from Russian by R. Alavina.

[Van58] HARRY S. VANDIVER (1958), *The Rapid Computing Machine as an Instrument in the Discovery of New Relations in the Theory of Numbers*, Proceedings of the National Academy of Sciences in the United States of America, **44**, 459–464.

[vH67] JEAN VAN HEIJENOORT (1967), *From Frege to Gödel: A source book in Mathematical Logic 1879–1931*, reprint third printing (paperback), 2002 ed., Harvard University Press, Cambridge, Massachusetts.

[vN27] JOHN VON NEUMANN (1927), *Zur Hilbertschen Beweistheorie*, Mathematische Zeitschrift, **26**, 1–46.

[vN45] ——— (1945), *First Draft of a Report on the EDVAC, Contract No. W-670-ORD-492, Moore School of Electrical Engineering, Univiversity of Pennsilvania, Philadelphia.*, also published in IEEE Annals of the History of Computing, Vol. 15, No. 4, 27-75, 1993.

[vN47] ——— (1947), *The Mathematician*, in: R.B. HEYWOOD (ed.), *The works of the Mind*, University of Chicago Press, Chicago, 180–196.

[vN51] ——— (1951), *The General and Logical Theory of Automata*, in: L.A. JEFFRES (ed.), *Cerebral Mechanisms in Behaviour – The Hixon Symposium*, John Wiley, New-York, 1–31, (Also published as [vN63], 288–328).

[vN56] ——— (1956), *Probabilistic Logics and the Synthesis of Reliable organisms from Unreliable Components*, in: [MS56a], 43–98. Also published as [vN63], 329–378.

[vN58] ——— (1958), *The Computer and the Brain*, Yale University Press, Yale.

[vN63] ——— (1963), *The Collected Works V. Design of Computers, Theory of Automata and Numerical Analysis.*, Pergamon Press, Oxford.

[vN66] ——— (1966), *The General and Logical Theory of Automata*, University of Illinois Press, Urbana, London.

[vN86] ——— (1986), *Papers of John von Neumann on Computers and Computing Theory, Charles Babbage Institute Reprint Series for the History of Computing.*, vol. 12, MIT Press.

[Wan87] HAO WANG (1987), *Reflections on Kurt Gödel*, MIT Press, Cambridge, Massachusetts.

[Wan96] ——— (1996), *A Logical Journey. From Gödel to Philosophy*, MIT Press, Cambridge, Massachusetts.

[Wan3a] ——— (1963a), *Tag systems and lag systems*, Mathematische Annalen, **152**, 65–74.

[Wat60] SHIGERU WATANABE (1960), *On a minimal Universal Turing machine*, Mcb report, Tokyo.

[Wat61] ——— (1961), *5-Symbol 8-State and 5-Symbol 6-State Universal Turing Machines*, Journal of the ACM, **8** (4), 476–483.

[Wat63] ——— (1963), *Periodicity of Post's normal Process of Tag*, in: [Fox63], 83–99.

[Web80] JUDSON C. WEBB (1980), *Mechanism, Mentalism and Metamathematics*, Reidel Publishing Company, Dordecht.

[Wei61] MARTIN H. WEIK (1961), *A Third Survey of Domestic Electronic Digital Computing Systems (Report No. 1115, March 1961)*, Ballistic Research Laboratories, Aberdeen Proving Ground, Maryland, available at: http://ed-thelen.org/comp-hist/BRL61.html.

[Wex81] RICHARD L. WEXELBLAT (ed.) (1981), *History of Programming Languages*, ACM Monograph Series, Academic Press, New York, proceedings of the first ACM SIGPLAN conference on History of programming languages (HOPL), 1978, Los Angeles.

[Wij] MICHIEL WIJERS, *Bibliography of the Busy Beaver Problem*, available at: www.win.tue.nl/ wijers/bbbibl.pdf.

[Wol02] STEPHEN WOLFRAM (2002), *A New Kind of Science*, Wolfram Inc., Champaign.

[Yan02] Ben Yandell (2002), *The honors class: Hilbert's problems and their solvers.*, AK Peters, Natick, Massachusetts.

[Zab95] Sandy L. Zabell (1995), *Alan M. Turing and the Central Limit Theorem*, The American Mathematical Monthly, **102** (6), 483–494.

[Zac99] Richard Zach (1999), *Completeness before Post: Bernays, Hilbert, and the development of propositional logic*, The Bulletin of Symbolic Logic, **5** (3), 331–366.

[Zac01] ——— (2001), *Hilbert's finitism: Historical, Philosophical and metamathematical Perspectives*, Ph.D. thesis, University of California, Berkley.

[Zus37] Konrad Zuse (1937), *Einführung in die allgemeine Dyadik*, available at Konrad Zuse Internet Archive (ZuP 009/004): www.zib.de/zuse.

[Zus44] ——— (1944), *Deutsche Patentanmeldung Z394, 11 October 1944*, available at Konrad Zuse Internet Archive (ZuP 005/017): www.zib.de/zuse.

[Zus45a] ——— (1945), *Bericht über meine Rechengeräte*, available at Konrad Zuse Internet Archive (ZuP 010/010): www.zib.de/zuse.

[Zus45b] ——— (1945), *Theorie der angewandten Logistik*, available at Konrad Zuse Internet Archive (ZuP 007/001): www.zib.de/zuse, indicated as *Urschrift des Plankalkl̈s*.

[Zus72] ——— (1972), *Der Plankalkül*, Gesellschaft für Mathematik und Datenverarbeitung., (63), bMBW-GMD-63.

[Zus80] ——— (1980), *Some Remarks on the History of Computing in Germany*, in: [HMR80], 611–627.

[Zus93] ——— (1993), *The Computer – My Life*, Springer Verlag, Berlin, translated from the German by Patricia McKenna and J. Andrew

Ross. Originally published in German as *Der Computer. Mein Lebenswerk*, Verlag Moderne Industrie, Munchen, 1970.