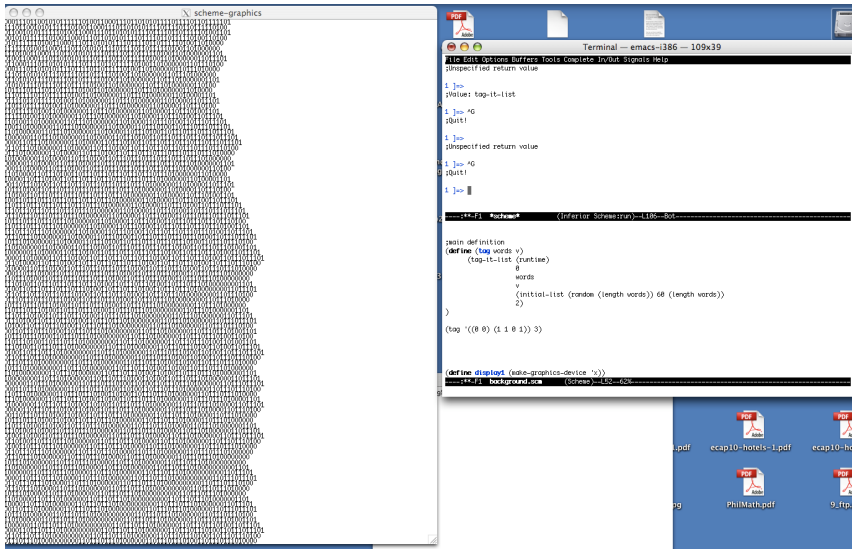
When logic faces digital circuitry. Early interactions between notation, formalism, programming and machinery.



Liesbeth De Mol
Centre for Logic and Philosophy of Science, Belgium
Fellow of the Fund for Scientific Research – FWO
elizabeth.demol@ugent.be

Some publicity first...

www.computing-conference.ugent.be

Intro.

Introduction (1)

⇒ **Topic** (Mathematical) Logic —- (physical) Computers

⇒ **Motivation (local)** "*[P]rofessional 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*" (Skolem, 1928)
⇒ Rendered "concrete" with computer (science) – a meeting point between logic, math, engineering and humans
⇒ *Opportunity* for philosophers to think *through* some high-speed logic, towards a more down-to-earth philosophy of computing (NOT Minds and Machines!)

⇒ **Approach** Tracing interaction/relation between formalism, notation, programming and machines in the early history of computer science

Introduction (2)
Why Logic (here)?

- (Assumption) Logic as a product of the human mind
- (Assumption) Logic as a "suitable" language for the computer

$\Rightarrow$ Logic as *an* intermediary between human and computer

Three opportunities (historical):

I. *Logic as Computation* **in** H/C: Emil Post's "Church-Turing" thesis

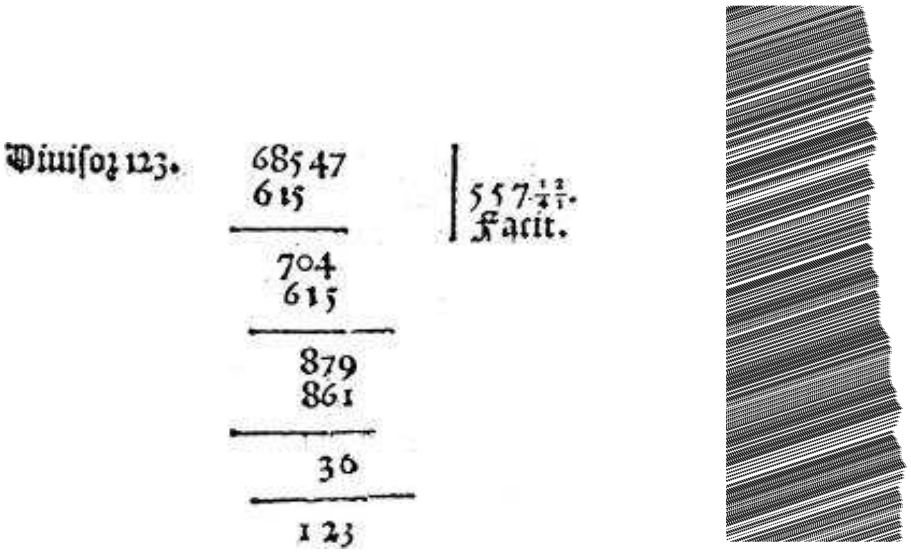
II. *Logic as Language* **between** H/C: Curry's theory of programming

III. *Logic as Reasoning* **by** H/C: Lehmer's criticism on automated theorem proving

Human ||| **Machine**

⇑

Computation



Post's CTT

Computation: Post's CTT

Classic CTT (1936) Identification between vague notion of effective calculability (Church)/computability (Turing) and λ-definability (Church)/Turing machines – definitional

Post's CTT I & II Identification between vague notion of generated set (1921)/solvability (1936) and normal set (1921)/formulation I (1936)

Computation: Post's CTT
From *Principia* to tag systems and normal form

- Strongly influenced by *PM*, Russell and Whitehead + Lewis' *Survey of SL*: "*The clear separation of [the form of a system from its content] is the ideal set by "mathematics without meaning""* (Lewis, 1918) – the heterodox view on math
- Formalism as a method to study mathematics as a whole , "*to obtain theorems about all [...] possible assertions*" (Post,1921) → quest for an algorithm that decides any mathematical problem
- The Lewisian method: "*Perhaps the chief [characteristic of this] method 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.*"
- Development of a series of more and more "meaningless" forms: canonical form A, B, tag systems, canonical form C, normal system

Definition of tag systems. A (relatively) famous Example

Let T_{Post} be defined by $\Sigma = \{0, 1\}, v = 3, 1 \to 1101, 0 \to 00$

$A_0 = 10111011101000000 \Rightarrow$ Primitive assertion

Definition of tag systems. A (relatively) famous Example

Let T_{Post} be defined by $\Sigma = \{0, 1\}, v = 3, 1 \to 1101, 0 \to 00$

$A_0 = 10111011101000000 \Rightarrow$ Primitive assertion
~~101~~1101110100000001101

Definition of tag systems. A (relatively) famous Example

Let T_{Post} be defined by $\Sigma = \{0, 1\}, v = 3, 1 \to 1101, 0 \to 00$

$A_0 = 10111011101000000 \Rightarrow$ Primitive assertion
~~101~~1101110100000001101
~~110~~1110100000011011101

Definition of tag systems. A (relatively) famous Example

Let T_{Post} be defined by $\Sigma = \{0, 1\}, v = 3, 1 \to 1101, 0 \to 00$

$A_0 = 10111011101000000 \Rightarrow$ Primitive assertion
~~101~~1101110100000001101
~~110~~1110100000011011101
~~111~~0100000011011101101

Definition of tag systems. A (relatively) famous Example

Let T_{Post} be defined by $\Sigma = \{0, 1\}, v = 3, 1 \to 1101, 0 \to 00$

$A_0 = 10111011101000000 \Rightarrow$ Primitive assertion

~~101~~1101110100000001101

~~110~~1110100000011011101

~~111~~0100000011011101101

~~010~~0000011011101110100

Definition of tag systems. A (relatively) famous Example

Let T_{Post} be defined by $\Sigma = \{0, 1\}, v = 3, 1 \to 1101, 0 \to 00$

$A_0 = 10111011101000000 \Rightarrow$ Primitive assertion

~~101~~1101110100000001101

~~110~~1110100000011011101

~~111~~0100000011011101101

~~010~~0000011011101110100

~~000~~0011011101110100000

Definition of tag systems. A (relatively) famous Example

Let T_{Post} be defined by $\Sigma = \{0, 1\}, v = 3, 1 \to 1101, 0 \to 00$

$A_0 = 10111011101000000 \Rightarrow$ Primitive assertion

~~101~~1101110100000001101

~~110~~1110100000011011101

~~111~~0100000011011101101

~~010~~0000011011101110100

~~000~~0011011101110100000

~~001~~$\underbrace{10111011101000000}_{A_0}$ $\Rightarrow$ Periodicity!

Definition of tag systems. A (relatively) famous Example

Let T_{Post} be defined by $\Sigma = \{0, 1\}, v = 3, 1 \to 1101, 0 \to 00$

$A_0 = 10111011101000000 \Rightarrow$ Primitive assertion

~~101~~1101110100000001101

~~110~~1110100000011011101

~~111~~0100000011011101101

~~010~~0000011011101110100

~~000~~0011011101110100000

~~001~~$\underbrace{10111011101000000}_{A_0}$ $\Rightarrow$ Periodicity!

- $\Rightarrow$ Definition of a *class* of symbolic logics according to a form
- $\Rightarrow$ Two decision problems (finiteness problems) for tag systems: the halting and reachability problem
- $\Rightarrow$ Exploration of tag systems $\to$ study of the *dynamic behavior* rather than logical concepts/coding; focus on the *process of generating* words
- $\Rightarrow$ **The reversal** "[T]he general problem of "tag" appeared hopeless, and with it our entire program of the solution of finiteness problems."

Post's thesis I

- **"Mathematics without meaning" → tag systems → normal form → Normal form theorem:**
  "In view of the generality of the system of *Principia Mathematica*, and its seeming inability to lead to any other generated set of sequences on a given set of letters than those given by our normal systems, we are led to the following generalization", i.e., Post's thesis I (Davis,1982)

- **Given thesis I + idea reversal programme:**

  "[...] the finiteness problem for the class of all normal systems is unsolvable"

  "A complete logic is impossible"

Post's "philosophy" of computing

- "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.].
- "For if symbolic logic has failed to give wings to mathematicians this study of symbolic logic opens up a new field concerned with the fundamental limitations of mathematics, more precisely the mathematics of Homo Sapiens." (Post to Church, March 24, 1936)
- "[T]he 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."
- Focus on and sensitive to processes and time ("time accounts")

Post's formalism and philosophy in human and machine computations (1)

⇒ Analysis of systems of symbolic logic and search for form without meaning results in time sensitive and very down-to-earth view on "computation" ≈ computer science attitude

⇒ Emphasizes that CTT is *not* computationalist (as often assumed) but, on the contrary, about limitations it imposes on human and machine computation, with logic being a method for exploring those limits.

⇒ Post's systems are highly suitable for a study of the "dynamics" of computation and logic-as-computation through computer-assisted research (re: form without meaning)

⇒ It are Post's "meaningless" forms rather than Turing's machines that are considered as the more natural for more practical aspects of computer science

Post's formalism and philosophy in human and machine computations (2)

- Backus-Naur form (Notation and programming!): "It was only in trying to describe ALGOL 58 that I realized that there was trouble about syntax description. *It was obvious that Post's productions were just the thing* [m.i.], and I hastily adapted them to that use." (Backus, 1981)

  <letter string> ::= <letter> | <letter string><letter>

  <letter> ::= A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | Y | W | X | Y | Z

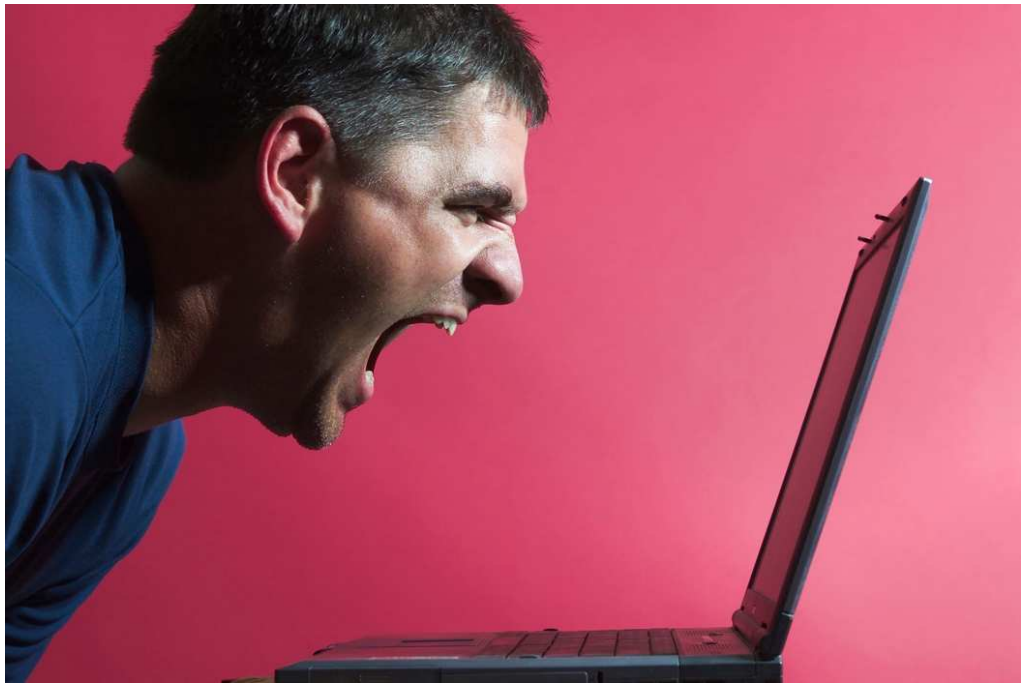
- Patterson and Burroughs company: "We are interested in your "productions" and your theory of canconical languages, which *appeal to us as possibly the most natural approach to the theory of computabiliy from the standpoint of one interested in syntactical machines.*" (letter to Post, Dec. 17, 1952)

- Chomsky and formal language theory: Post's canonical form C = Post production systems: "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" (personal communication, 2005)

Human $\|\|$ **Machine**

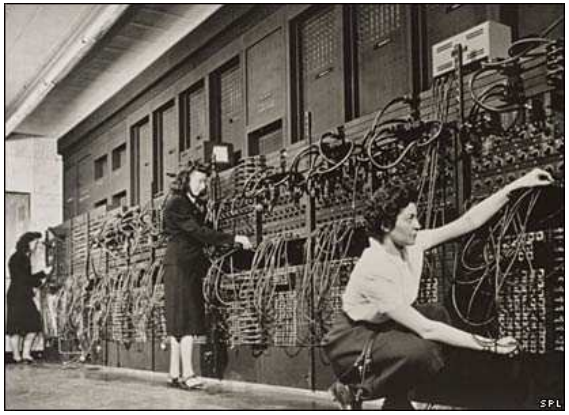
$\Uparrow$

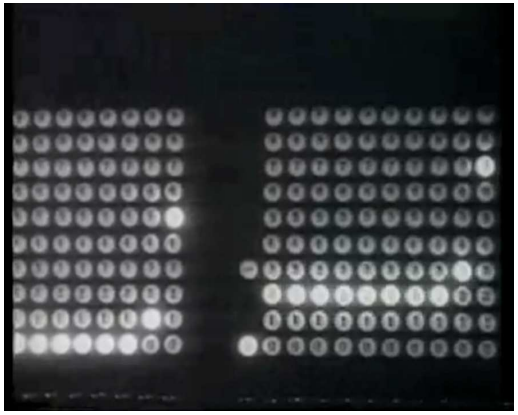
Language



Curry's theory of programming

ENIAC – A machine without language (1946)





- "to talk": Programming through direct physical contact: the ENIAC "was a son-of-a-bitch to program" (De Mol & Bullynck, 2008)
- "to listen": Direct access to the computational process through sound and lights: "[W]hen you were doing calculations these lights were flashing as the numbers built up and as you transferred numbers and things of this kind. They were very essential to debugging, very essential. [...] That's the only way you read what the machine [...] stored, what it was doing. [I]t was [...] where people saw for the first time, saw calculations taking place" (Jean Bartik, 1973)

⇒ *Absence of an intermediary language: a major bottleneck or not?*

Haskell Curry's reply to the bottleneck

- **Use of a prototypical "logical" problem**: "*[The problem of inverse interpolation] is almost ideal for the study of programming; because, although it is simple enough to be examined in detail by hand methods, it is complex enough to contain a variety of kinds of program compositions*"
- **The problem of composition**: *Suppose that we wish to perform a computation which is a complex of simple processes that have already been planned. Suppose that for each of these component processes we have a plan recorded in the form of what is here called a program, by means of a system of symbolization called a code. It is required to form a program for the composite computation. This problem is here attacked theoretically by using techniques similar to those used in some phases of mathematical logic."*
- **New notation and introduction of automated composition**: *"The present theory develops in fact a notation for program construction which is more compact than the "flow charts" of [Goldstine and Von Neumann]. Flow charts will be used [...] primarily as an expository device. By means of this notation a composite program can be exhibited as a function of its components in such a way that the actual formation of the composite program can be carried out by a suitable machine."*

Haskell Curry's reply to the bottleneck (detailed)
Steps of program composition I: Substitution

Basic Definition and notation for transformation of first and second kind + recplacement

Substitution – Notation: $Z = X \to Y$

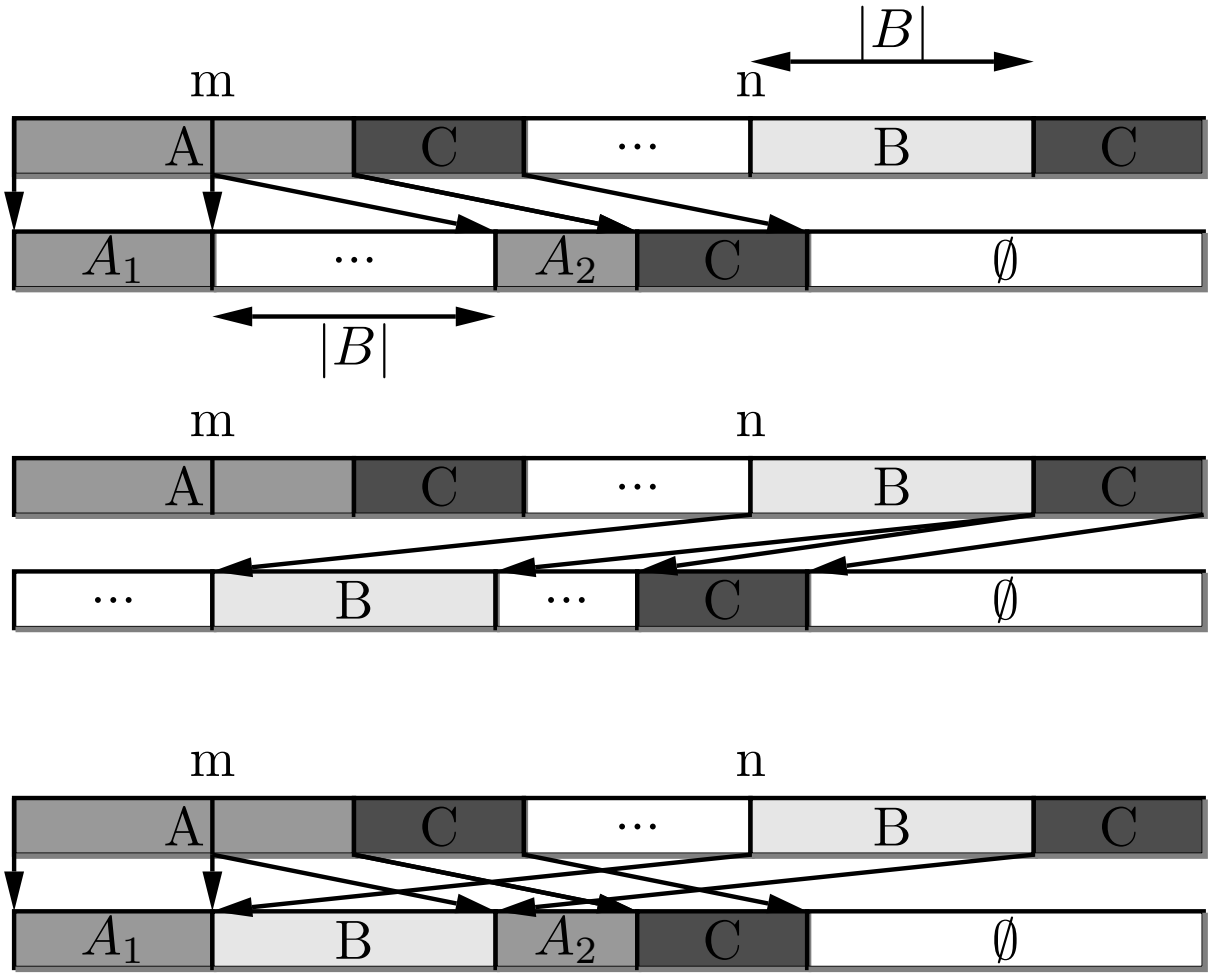
$X = AC$ and $Y = BC$ are normal, m is the location number ($m \in A$) at which Y ist to be substituted, then $Z = X \to Y = S_Y(X) = [\frac{\Theta T_1}{[T_2](Y)}](X) = \frac{\Theta T_1}{[T_2](Y)}(T_1)(X)$ is defined by

$$T_1(k) = \begin{cases} k & \text{for } 0 < k < m \\ m + |B| - 1 & \text{for } k = m \\ k + |B| - 1 & \text{for } m < k \leq |A| + |C| \end{cases}$$

$$T_2(k) = \begin{cases} m + k - n & \text{for } n \leq k \leq n + |B| - 1 \\ |A| + k - n & \text{for } n + |B| < k \leq n + |B| + |C| - 1 \end{cases}$$

Haskell Curry's reply to the bottleneck (detailed)

Steps of program composition I: Substitution



Haskell Curry's reply to the bottleneck (detailed)

Steps of program composition II: Development of Notation

$$Y \to (It(m,i) \to I \text{ and } O_2)$$

$$It(m,i) = \{i : A\} \to \{A+1 : A\} \to \{A : i\} \to \{m : A\} \to \{A-i : A\} \to \{A < 0\}$$

$$n! = \{1 : A\} \to \{A : x\} \to \{A : i\} \to Y \to It(n,i)$$

$$Y = \{ix : x\}$$

"The present theory develops in fact a notation for program construction [...] which is more compact than the "flow charts"'

Haskell Curry's reply to the bottleneck (detailed)
Steps of program composition III: Compilers and basic programs

- **Analysis into basic programs** "This analysis can, in principle at least, be carried clear down until the ultimate constituents are the simplest possible programs [...] Of course, it is a platitude that the practical man would not be interested in composition techniques for programs of such simplicity, but it is a common experience in mathematics that one can deepen ones insight into the most profound and abstract theories by considering trivially simple examples." (Curry, 1949)
- **Extremely sensitive to machine issues**: motivation basic programs related to time and (especially) space efficiency: "[T]he possibility of making such [arithmetic] programs without using auxiliary memory is a great advantage to the programmer. Therefore, it is recommended that, if it is not practical to design the machine so as to allow these additional orders, then a position in the memory should be permanently set aside for making the reductions contemplated"
- Basis for construction arithmetic compiler (complete) and first steps compiler for branching and secondary programs

Curry's approach on the development of a "logical" language between human and computer

- Theory of program composition ≈ combinatorial calculus
- The mathematical logician's attitude: "[I]t is evident that one can formalize in various ways and that some of these ways constitute a more profound analysis than others. Although from some points of view one way of formalization is as good as any other, yet a certain interest attaches to the problem of simplification [...] **In fact we are concerned with constructing systems of an extremely rudimentary character, which analyze processes ordinarily taken for granted.**" (Curry, 1942)
- In sharp contrast with von Neumann approach (the programmer vs. the quick hacker) + awareness of necessity of a distance between human and machine through intermediary language:

  > "[O]ne comment seems to be in order in regard to this arrangement. The scheme allows certain data to be inserted directly into the machine by means of a typewriter-like device. Such an arrangement is very desirable for trouble-shooting and computations of a tentative sort, but for final computations of major importance it would seem preferable to proceed entirely from a program or programs
  >
  > "It is said that during the war an error in one of the firing tables was caused by using the wrong lead screw in the differential analyser. Such

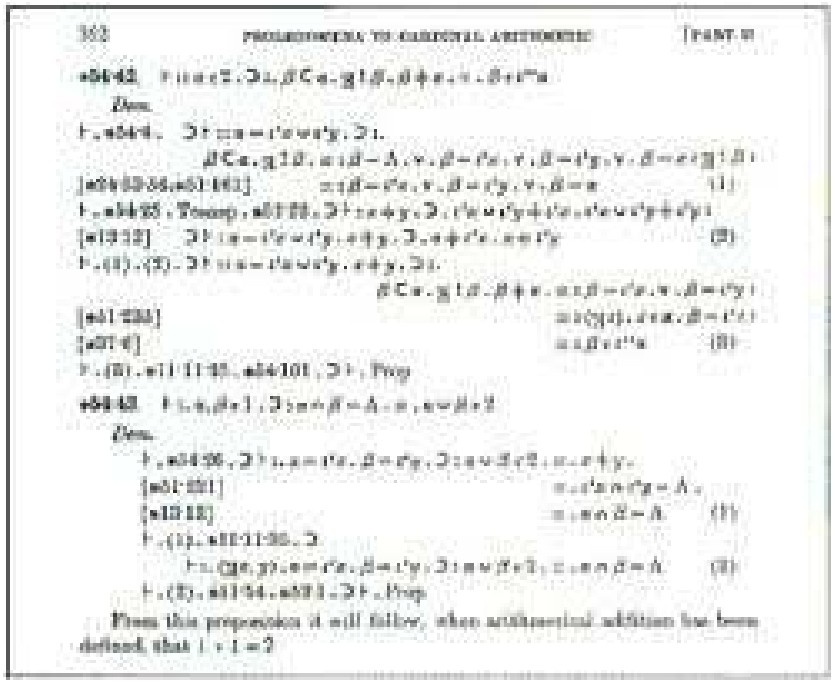
an error would have been impossible if the calculation had been completely programmed."

- BUT, with Curry, distance does not imply "ambient intelligence": Sensitivity to machine issues and awareness of feed-back between human, language (logic) and machine.
- Systematic analysis towards a calculus of programming $\approx$ systematic analysis towards a theory of computing (Emil Post)

Human ||| Machine

⇑

Reasoning



362 PROLEGOMENA TO CARDINAL ARITHMETIC [PART II

*54·42. $\vdash :\alpha \cap \beta = \Lambda .\equiv. \alpha \subseteq -\beta$

Dem.

$\vdash .*54·4. \quad \supset \vdash :\alpha = \iota'x \cup \iota'y .\supset. \alpha \subseteq -\beta .\equiv. x \neq y .\supset. \beta = -\alpha$

[*51·231] $\equiv . \beta = -\iota'x .\vee. \beta = -\iota'y$ (1)

$\vdash .*54·25 .\text{Transp} .*51·23 .\supset \vdash :x \neq y .\supset . -\iota'x \cup -\iota'y \neq -\iota'x \cup \iota'y$

[*13·12] $\supset \vdash :\alpha = \iota'x \cup \iota'y .\supset. x \neq y .\supset. \alpha \neq \iota'x \cup \iota'y$ (2)

$\vdash .(1).(2). \supset \vdash :\alpha = \iota'x \cup \iota'y .\supset. x \neq y .\supset.$

$\alpha \subseteq -\beta .\equiv. \beta = -\iota'x .\vee. \beta = -\iota'y$

[*51·235] $\equiv . \alpha = \iota'(-\iota'x) .\vee. \alpha = \iota'(-\iota'y)$

[*37·6] $\equiv . \alpha \neq \iota'\Lambda$ (3)

$\vdash .(3). *11·11·35 . *54·101 . \supset \vdash . \text{Prop}$

*59·43 $\vdash :\alpha, \beta \in 1 . \supset : \alpha \cap \beta = \Lambda .\equiv. \alpha \cup \beta \in 2$

Dem.

$\vdash .*54·26 . \supset \vdash :. \alpha = \iota'x . \beta = \iota'y . \supset : \alpha \cup \beta \in 2 .\equiv. x \neq y .$

[*51·231] $\equiv . \iota'x \cap \iota'y = \Lambda .$

[*13·12] $\equiv . x \cap y = \Lambda$ (1)

$\vdash .(1). *11·11·35 . \supset$

$\vdash :. \exists x, y . \alpha = \iota'x . \beta = \iota'y . \supset : \alpha \cup \beta \in 2 .\equiv. x \cap y = \Lambda$ (2)

$\vdash .(2). *11·54 . *59·1 . \supset \vdash . \text{Prop}$

From this proposition it will follow, when arithmetical addition has been defined, that $1 + 1 = 2$

Lehmer's critcism on automated theorem proving

Lehmer's criticism on automated theorem proving ⇒ Lehmer's view on computers and math

- **Computer as a means to disclose and explore the universe of math**
- **The machine is only useful in tackling humanly impractical problems.**: "*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.*" (Lehmer, 1963)

Lehmer's criticism on automated theorem proving
Two pioneers: Davis and Wang

- "A readily quotable [...] indication of the power of the program P is the fact that it disposed of nine chapters of *Principia* in about 8.4 minutes, with an output of about 110 pages of 60 lines each, containing full proofs of all the theorems (over 350). [...] [T]he theorems of the propositional calculus (over 200) were proved in about 37 minutes with the on-line printer, and it was estimated that the computing time was only about 3 minutes [...] the 200 strong theorems in the propositional calculus took about 5 minutes while the 150 strong theorems with quantifiers took less than 4 minutes." (Wang, 1960)
- "Since it is now known that Presburger's procedure was worse than exponential complexity, it is not surprising that this program did not perform very well. Its great triumph was to prove that the sum of two even numbers is even" (Davis, 1957)

Lehmer's criticism on automated theorem proving

- "[E]xamples of [...] programs for doing elaborate logic are [...]those of Wang for proving theorems on foundations of mathematics. These striking achievements are only the beginning of today's theorem proving developments. From one point of view [...] **these efforts may be regarded as contributions to the art of simulation. In particular the theorems proved are not new and are easily proved by a human being at a blackboard. In fact, one gets a spurious feeling of satisfaction in thus being able to compete so successfully with a multi-million dollar monster at (what we mistakenly think is) its own game.** I would like to speak briefly of some theorem proving programs that we have been running in which the human is completely ouclassed in what [...] are fair contests. [...] I hope I have convinced you that one *can* get to first base in the game of theorem proving. Perhaps the professional logician will consider it unfair that we didn't start from home base, namely Peano's axioms. But this requirement is waived for human beings so why should one require it of the machine? (Lehmer, *Some high-speed logic*1963)

Lehmer's criticism on automated theorem proving
A tradition of anti-simulation?

- Licklider, 1960: "Computing machines can do readily, well, and rapidly many things that are difficult or impossible for man, and men can do readily and well, though not rapidly, many things that are difficult or impossible for computers. **That suggests that a symbiotic cooperation, if successful in integrating the positive characteristics of men and computers, would be of great value.**"
- Dijkstra, 1985: "feel that the effort to use machines to try to mimic human reasoning is both foolish and dangerous. It is foolish because if you look at human reasoning as is, it is pretty lousy; even the most trained mathematicians are amateur thinkers. Instead of trying to imitate what we are good at, I think it is much more fascinating to investigate what we are poor at. **It is foolish to use machines to imitate human beings, while machines are very good at being machines, and *that* is precisely something that human beings are very poor at.** Any successful AI project by its very nature would castrate the machine."

$\Rightarrow$ $\approx$ Post and search for "form" *without* meaning.

$\Rightarrow$ Impose ideal of Logic on computer? "*HAL: Dave, I don't know how else to put this, but it just happens to be an unalterable fact that I am incapable of being wrong.*"

5. Discussion

Discussion

Human-Logic-Machines: Three situations/opportunities

1. Development of "pure" (meaningless) form out of logic – limitations *and* possibilities of computing in H and C; logic as a calculating *process* in time
2. Logic as a method to develop the necessary distance between Humans/machines, BUT, not to hide the machine! In search for a logic that lies in-between.
3. Logic and human/machine reasoning – a tricky problem: *not simulation* of human reasoning but engage and intertwine human with machine "reasoning/doing". Logic and formalism can help here.

Questions....

- La logique est-elle multiple/ relative?
- La logique doit-elle être mathématique (pour le développement de l'informatique)?
- La logique résout-elle des problèmes philosophiques?
- Raisonner est-ce calculer?
- La logique est-elle nécessaire (pour le développement de l'informatique)?