Proceedings of the Ninth International Workshop on Non-Monotonic Reasoning, Action and Change (NRAC'11)

Editors:

Sebastian Sardina
School of Computer Science and IT
RMIT University
Melbourne, VIC, 3000
Australia
sebastian.sardina@rmit.edu.au

Stavros Vassos
Department of Informatics and Telecommunications
National and Kapodistrian University of Athens
Athens, 15784
Greece
stavrosv@di.uoa.gr

Technical Report
RMIT-TR-11-02
July 2011

School of Computer Science and Information Technology
RMIT University
Melbourne 3000, Australia

Preface

We present here the informal proceedings for the Ninth International Workshop on Non-Monotonic Reasoning, Action and Change (NRAC'11), a well-established forum to foster discussion and sharing of experiences among researchers interested in the broad areas of nonmonotonic reasoning, and reasoning about action and change, including belief revision, planning, logic programming, argumentation, causality, probabilistic and possibilistic approaches to KR, and other related topics.

Since its inception in 1995, NRAC has always been held in conjunction with International Joint Conference on Artificial Intelligence (IJCAI), each time with growing success, and showing an active and loyal community. Previous editions were held in 2009 in Pasadena, USA; in 2007 in Hyderabad, India; in 2005 in Edinburgh, Scotland; in 2003 in Acapulco, Mexico; in 2001 in Seattle, USA; in 1999 in Stockholm, Sweden; in 1997 in Nagoya, Japan; and in 1995 in Montreal, Canada. This time, NRAC'11 is held as a 1.5-day satellite workshop of IJCAI'11, in Barcelona, Spain, and will take place on July 16 & 17.

An intelligent agent exploring a rich, dynamic world, needs cognitive capabilities in addition to basic functionalities for perception and reaction. The abilities to reason nonmonotonically, to reason about actions, and to change one's beliefs, have been identified as fundamental high-level cognitive functions necessary for common sense. Many deep relationships have already been established between the three areas and the primary aim of this workshop is to further promote this cross-fertilization. A closer look at recent developments in the three fields reveals how fruitful such cross-fertilization can be. Comparing and contrasting current formalisms for Nonmonotonic Reasoning, Reasoning about Action, and Belief Revision helps identify the strengths and weaknesses of the various methods available. It is an important activity that allows researchers to evaluate the state-of-the-art. Indeed a significant advantage of using logical formalisms as representation schemes is that they facilitate the evaluation process. Moreover, following the initial success, more complex real-world applications are now within grasp. Experimentation with prototype implementations not only helps to identify obstacles that arise in transforming theoretical solutions into operational solutions, but also highlights the need for the improvement of existing formal integrative frameworks for intelligent agents at the ontological level.

This workshop will bring together researchers from all three areas with the aim to compare and evaluate existing formalisms, report on new developments and innovations, identify the most important open problems in all three areas, identify possibilities of solution transferal between the areas, and identify important challenges for the advancement of the areas. As part of the program we will be considering the status of the field and discussing questions such as: What nonmonotonic logics and what theories of action and change have been implemented?; how to compare them?; which frameworks are implementable?; what can be learned from existing applications?; what is needed to improve their scope and performance?

In addition to the paper sessions, this year's workshop features invited talks by two internationally renowned researchers: Jürgen Dix (TU Clausthal University, Germany) on '*'How to test and compare Multi-agent systems?"* and Grigoris Antoniou (University of Crete, Greece) on *"Nonmonotonic Reasoning in the Real: Reasoning about Context in Ambient Intelligence Environments"*

The programme chairs would like to thank all authors for their contributions and are also very grateful to the program committee for their hard work during the review phase and for providing excellent feedback to the authors. The programme chairs are also very grateful to Pavlos Peppas and Mary-Anne Williams from the steering committee for always being available for consultation, and to Maurice Pagnucco for helping us to put these Proceedings together.

June 2011

Sebastian Sardina RMIT University
Stavros Vassos National and Kapodistrian University of Athens

Organization

Organizing Committee

Sebastian Sardina	RMIT University, Australia
Stavros Vassos	National and Kapodistrian University of Athens, Greece

Steering Committee

Gerhard Brewka	University of Leipzig, Germany
Michael Thielscher	University of NSW, Australia
Leora Morgenstern	SAIC Advanced Systems and Concepts, USA
Maurice Pagnucco	University of NSW, Australia
Pavlos Peppas	University of Patras, Greece
Mary-Anne Williams	University of Technology, Sydney, Australia
Andreas Herzig	Universite Paul Sabatier, France
Benjamin Johnston	University of Technology, Sydney, Australia

Program Committee

Xiaoping Chen	University of Science and Technology China, China
Jim Delgrande	Simon Fraser University, Canada
Jérôme Lang	Universite Paul Sabatier, France
Thomas Meyer	Meraka Institute, South Africa
Michael Thielscher	University of NSW, Australia
Sheila McIlraith	University of Toronto, Canada
Eduardo Fermé	University of Madeira, Portugal
Dongmo Zhang	University of Western Sydney, Australia
Mehdi Dastani	Utrecht University, The Netherlands
Giuseppe De Giacomo	Sapienza Universita' di Roma, Italy
Christian Fritz	PARC (Palo Alto Research Center), USA
Leora Morgenstern	SAIC Advanced Systems and Concepts, USA
Pavlos Peppas	University of Patras, Greece
Sajjad Haider	Institute of Business Administratio, Pakistan
Alfredo Gabaldon	Universidade Nova de Lisboa, Portugal

Table of Contents

An Adaptive Logic-based Approach to Abduction in AI*

(Preliminary Report)

Tjerk Gauderis
Centre for Logic and Philosophy of Science
Ghent University, Belgium
Tjerk.Gauderis@UGent.be

Abstract

In a logic-based approach to abductive reasoning, the background knowledge is represented by a logical theory. A sentence ϕ is then considered as an explanation for ω if it satisfies some formal conditions. In general, the following three conditions are considered crucial: (1) ϕ together with the background knowledge implies ω; (2) ϕ is logically consistent with what is known; and (3) ϕ is the most 'parsimonious' explanation. But, since abductive reasoning is a non-monotonic form of reasoning, each time the background knowledge is extended, the status of previously abduced explanations becomes once again undefined.

The adaptive logics program is developed to address these types of non-monotonic reasoning. In addition to deductive reasoning steps, it allows for direct implementation of defeasible reasoning steps, but it adds to each formula the explicit set of conditions that would defeat this formula. So, in an adaptive logic for abduction, a formula is an abduced hypothesis as long as none of its conditions is deduced. This implies that we will not have to recheck all hypotheses each time an extension to our background knowledge is made. This is the key advantage of this approach, which allows us to save repetitive re-computations in fast growing knowledge bases.

1 The Adaptive Logics Framework

The adaptive logics program is established to offer insight in the direct application of defeasible reasoning steps.[1] This is done by focussing on which formulas would falsify a defeasible reasoning step. Therefore, in adaptive logics a *formula* is a pair (A, Δ) with A a regular well-formed formula in the language of the logic over which the considered theory $\mathcal{T}$ is defined and Δ, the *condition* of the formula, is a set of regular well-formed formulas that are assumed to be false. To express this assumption, these formulas are generally called *abnormalities* in adaptive logic literature.[2] For an adaptive logic in standard format, the abnormalities are characterized by a logical form.

The set of *plausibly derivable formulas* $\mathcal{P}$ from a logical theory $\mathcal{T}$ is formed in the following way:

1. *Premise Rule:* if $A \in \mathcal{T}$, then $(A, \emptyset) \in \mathcal{P}$
2. *Unconditional Inference Rule:*
   if $A_1, \ldots, A_n \vdash B$
   and $(A_1, \Delta_1), \ldots, (A_n, \Delta_n) \in \mathcal{P}$,
   then $(B, \Delta_1 \cup \ldots \cup \Delta_n) \in \mathcal{P}$
3. *Conditional Inference Rule:*
   if $A_1, ..., A_n \vdash B \vee Dab(\Theta)$
   and $(A_1, \Delta_1), \ldots, (A_n, \Delta_n) \in \mathcal{P}$,
   then $(B, \Delta_1 \cup \ldots \cup \Delta_n \cup \Theta) \in \mathcal{P}$

where $Dab(\Theta)$ stands for *disjunction of abnormalities*, i.e. the classical disjunction of all elements in the finite set of abnormalities Θ. This third rule, which adds new conditions, makes clear how defeasible steps are modeled. The idea is that if we can deductively derive the disjunction of a defeasible result B and the formulas, the truth of which would make us to withdraw B, we can defeasibly derive B on the assumption that none of these formulas is true.

Apart from the set of plausible formulas $\mathcal{P}$ we need a mechanism that selects which defeasible results should be withdrawn. This is done by defining a marking strategy. In the adaptive logics literature, several strategies have been developed, but for our purposes it is sufficient to consider the *simple strategy*. According to this strategy, the set of the *derivable formulas* or *consequences* $\mathcal{D} \subseteq \mathcal{P}$ consists of :

1. *Deductive Results:* if $(A, \emptyset) \in \mathcal{P}$, then $(A, \emptyset) \in \mathcal{D}$
2. *Unfalsified Defeasible Results:*
   if $(A, \Theta) \in \mathcal{P}$ (with $\Theta \neq \emptyset$)
   and if for every $\omega \in \Theta : (\omega, \emptyset) \notin \mathcal{P}$,
   then $(A, \Theta) \in \mathcal{D}$

*Research for this paper was supported by project subventions from the Special Fund for Research (BOF) of Ghent University. I am grateful to the anonymous referees for their helpful suggestions.

[1] The adaptive logics program is founded by Batens in the eighties. For a more recent overview of the general results, see [Batens, 2007]. For a philosophical defense of the use of adaptive logics, see [Batens, 2004].

[2] This representation of adaptive logics is a reinterpretation of the standard representation of adaptive logics, which is in terms of a proof theory. I made this reinterpretation for purposes of comparison with other approaches in AI.

So, apart from the deductive results – which are always derivable – this strategy considers all defeasible results as derived, as long as none of the elements of their condition is deductively derived.

From the definitions of the sets $\mathcal{P}$ and $\mathcal{D}$, we can understand how adaptive logics model the non-monotonic character of defeasible reasoning. If our theory $\mathcal{T}$ is extended to the new theory $\mathcal{T}'$ ($\mathcal{T} \subset \mathcal{T}'$), then we can define the corresponding sets $\mathcal{P}'$ and $\mathcal{D}'$. On the one hand, the set of plausibly derivable formulas will be monotonic ($\mathcal{P} \subset \mathcal{P}'$), since there is no mechanism to withdraw elements from this set and it can only grow larger.[3] On the other hand, we know that the set of derivable formulas is non-monotonic ($\mathcal{D} \not\subset \mathcal{D}'$). It is possible that a condition of a defeasible result in $\mathcal{D}$, is suddenly – in light of the new information in $\mathcal{T}'$ – deductively derivable. So, this result will not be part of $\mathcal{D}'$ any more. Obviously, no deductive result will ever be revoked.

This makes this kind of logics very apt to model fast growing knowledge bases.[4] If one needs a previously defeasibly derived result at a certain point, we cannot be sure whether it is still valid, because there might have been several knowledge base updates in the meantime. But, since the set of plausible formulas is monotonic, we know this formula will still be in $\mathcal{P}$. So, instead of recalculating the whole non-monotonic set $\mathcal{D}$ after each knowledge base extension (which is the traditional approach), it is sufficient to expand the monotonic set $\mathcal{P}$. Of course, in this approach, if we want to use a defeasible result at a certain stage of knowledge base expansion, we will first have to check its condition. Still, it is easily seen that a lot of repetitive re-computation is avoided, certainly in situations in which we only need a small percentage of the defeasible results at every stage of knowledge base expansion.

Moreover, it is proven that if the adaptive logic is in standard format, which means that the abnormalities have a fixed logical form, the corresponding logic will have all interesting meta-theoretic properties. The logic for abduction developed in this article will be in standard format and will therefore be sound, complete, proof invariant and have the fixed-point property.[5]

2 Other "conditional" approaches

As far as I can see, two other approaches in AI have used the idea of directly adding conditions or restrictions to formulas. On the one hand, there is a line of research, called "Cumulative Default Reasoning", going back to a paper of [Brewka, 1991] with the same title. On the other hand, in the area of argumentation theory, some work on defeasible logic programs (see, for instance, [García and Simari, 2004]) is also based on formulas together with consistency conditions that need to be satisfied to make these formulas acceptable.

The main difference with these research programs is that the abnormalities in adaptive logics are based on a fixed logical form. This means that, for instance, the logical form for abduction – explained in this paper – is the form of abnormalities for any premise set on which we want to apply abductive reasoning. Put in other words, as soon as a fully classical premise set is given, all the possible abnormalities and, therefore, all the plausible and finally derivable abductive results can be calculated. There is no element of choice. In the other approaches, the conditions of defeasible steps must be given in the premise set, which leaves an element of choice which conditions we want to add to which defeasible implications. In adaptive logics, the defeasible consequences can be derived as soon as we have a classical premise set and as soon as we have chosen the appropriate logic for the kind of reasoning we want to do (e.g. abduction).

3 The problem of multiple explanatory hypotheses in Abduction

If we focus our attention now to the abductive problem, we cannot allow that the different defeasible results – the abduced hypotheses – are together in the set $\mathcal{P}$. For instance, if Tweety is a non-flying bird, he may be a penguin or an ostrich. But a set containing both the formulas $(penguin(Tweety), \Theta_1)$ and $(ostrich(Tweety), \Theta_2)$ is inconsistent.[6]

An elegant solution to this problem is found by translating this problem to a modal framework. When we introduce a possibility operator $\Diamond$ to indicate hypotheses and the corresponding necessity operator ($\Box =_{df} \neg\Diamond\neg$) to represent background knowledge, we evade this problem. The Tweety-example translates, for instance, as such (for variables ranging over the domain of all birds):

Background Knowledge:

$$\begin{aligned}(\Box\forall x(penguin(x) \supset \neg flies(x)), \emptyset)\\ (\Box\forall x(ostrich(x) \supset \neg flies(x)), \emptyset)\\ (\Box\neg flies(Tweety), \emptyset)\end{aligned}$$

Plausible defeasible results:

$$(\Diamond penguin(Tweety), \Theta_1)$$
$$(\Diamond ostrich(Tweety), \Theta_2)$$

So, with this addition the sets $\mathcal{P}$ and $\mathcal{D}$ are consistent again. Though, in this situation it is not really necessary to maintain the modal operators, because we can quite easily make a translation to a hierarchical set-approach, by borrowing some ideas of the Kripke-semantics for modal logics.[7] In these semantics, a hypothesis is said to be true in a possible world that is accessible from the world in which the hypothesis is stated, while necessities are true in all accessible worlds.

[3] It is important to understand "plausible" as "initially plausible" (at the time of derivation) and not as "plausible according to our present insights". The second definition would, of course, have led to a non-monotonic set.

[4] In that way, this kind of logic can offer a solution to what [Paul, 2000] mentioned as one of the main problems of both set-cover-based and some logic-based approaches to abduction.

[5] For an overview of the generic proofs of these properties, see [Batens, 2007].

[6] At this point, we make abstraction of the exact conditions. The details of the conditions will be explained below.

[7] It is important to remember that we are constructing a syntactical representation, not a semantics for the underlying logic.

If we define now a world(-set) as the set of formulas assigned to that world, we can finish our translation from modalities to sets. We define the actual world w as the set of all formulas of the knowledge base and all deductive consequences. The elements of the set w are the only formulas that have a $\Box$-operator in our modal logic, and are thus the only elements that will be contained in every world-set in our system. Subsequently, for every abduced hypothesis we define a new world-set that contains it. This world is hierarchically directly beneath the world from which the formula is abduced. This new set contains further the formulas of all the world-sets hierarchically above, and will be closed under deduction. To make this hierarchy clear, we will use the names $w_1, w_2, \ldots$ for the worlds containing hypotheses directly abduced from the knowledge base, $w_{1.1}, w_{1.2}, \ldots, w_{2.1}, \ldots$ for hypotheses abduced from a first-level world, etc.

With this translation in mind, we can omit the modal operators and just keep for every formula track of the hierarchically highest world-set that contains it. So, our Tweety example can be respresented as such:

$$\begin{array}{rl}
(\forall x(penguin(x) \supset \neg flies(x)), \emptyset) & w \\
(\forall x(ostrich(x) \supset \neg flies(x)), \emptyset) & w \\
(\neg flies(Tweety), \emptyset) & w \\
(penguin(Tweety), \Theta_1) & w_1 \\
(ostrich(Tweety), \Theta_2) & w_2
\end{array}$$

Since the hierarchical system of sets w_i is equivalent to the set $\mathcal{P}$ (the plausibly derivable results) of a logic for abduction, the definition of the set $\mathcal{D}$ (of this logic) can be applied to this system of sets too. It is clear that only the deductive consequences – the only formulas with an empty condition – will be the formulas in the set w. Further, since all formulas in a world-set have the same conditions, i.e. the condition of the hypothesis for which the world is created, the definition of $\mathcal{D}$ does not only select on the level of the formulas, but actually also on the level of the world-sets.[8] Put in other words, $\mathcal{D}$ selects a subsystem of the initial system of hierarchically ordered sets. The different sets in this subsystem are equivalent with what [Flach and Kakas, 2000] called *abductive extensions* of some theory. In this way, the logic can handle mutually contradictory hypotheses,[9] without the risk that any set of formulas turns out to be inconsistent.

4 Reformulation of the abductive problem in the adaptive logics format

So far, in this paper we have shown – in the first section – how we can represent the standard format of adaptive logics in terms of two sets $\mathcal{P}$ and $\mathcal{D}$, and – in the third section – how we can cope with contradictory hypotheses by using a hierarchical system of world-sets. In this section we will now use this set representation to reformulate the syntax of the logic $\mathbf{MLA^s}$, which is previously developed in [Gauderis, 2011].[10] This adaptive logic, the name of which stands for Modal Logic for Abduction, is an adaptive logic designed to handle contradictory hypotheses in abduction. The reformulation in terms of sets is performed with the goal to integrate the adaptive approach with other AI-approaches. First we need to define the abductive problem in a formal way.

Definition 1. *An* abductive system $\mathcal{T}$ *is a triple* $(\mathcal{H}, \mathcal{O}, d)$ *of the following three sets*

- *a set of clauses* $\mathcal{H}$ *of the form*

  $$\forall x(A_1(\alpha) \wedge \ldots \wedge A_n(\alpha) \supset B(\alpha))$$

  with $A_1(\alpha), \ldots, A_n(\alpha), B(\alpha)$ *literals and* α *ranging over* d.
- *a set of observations* $\mathcal{O}$ *of the form* $C(\gamma)$ *with* C *a literal and a constant* $\gamma \in d$.
- *a domain* d *of constants.*

All formulas are closed formulas defined over a standard predicative first order logic.

Furthermore, the notation does not imply that predicates should be of rank 1. Predicates can have any rank, the only preliminaries are that in the clauses all A_i and B share a common variable, and that the observations have at least one variable that is replaced by a constant. Obviously, for predicates of higher rank, extra quantifiers for the other variables need to be added to make sure that all formulas are closed.

Definition 2. *The* background knowledge *or* actual world w *of an abductive system* $\mathcal{T} = (\mathcal{H}, \mathcal{O}, d)$ *is the set*

$$w = \{(P, \emptyset) \mid \mathcal{H} \cup \mathcal{O} \vdash P\}$$

Since it was the goal of an adaptive logic-approach to implement directly defeasible reasoning steps, we will consider instances of the Peircean schema for abduction [Peirce, 1960, 5.171]:

> The surprising fact, C is observed;
> But if A were true, C would be a matter of course,
> Hence, there is reason to suspect that A is true.

When we translate his schema to the elements of $\mathcal{T} = (\mathcal{H}, \mathcal{O}, d)$, we get the following schema:

$$\begin{array}{l}
\forall x(A_1(\alpha) \wedge \ldots \wedge A_n(\alpha) \supset B(\alpha)) \\
B(\gamma) \\
\hline
A_1(\gamma) \wedge \ldots \wedge A_n(\gamma)
\end{array}$$

To implement this schema – better-known as the logical fallacy *Affirming the Consequent* – in an adaptive logic, we need to specify the logical form of the conditions that would falsify the application of this rule. As we can see from how the conditional inference rule is introduced in the first section, the disjunction of the hypothesis and all defeating conditions needs to be derivable from the theory. To specify these conditions, we will first overview the different *desiderata* for our abductions.

[8] Strictly speaking, each world-set contains also all formulas of the world-sets hierarchically above. But since these formulas are also contained in those worlds above, no information is lost if we allow that $\mathcal{D}$ can select on the level of the world-sets.

[9] Consider, for instance, the famous quaker/republican example: our approach will lead to two different abductive extensions, one in which Nixon will be a pacifist and another one in which he isn't.

[10] In the original article, the syntax of the logic $\mathbf{MLA^s}$ is defined in terms of a proof theory.

Obviously, it is straightforward that if the negation of the hypothesis can be derived from our background knowledge, the abduction is falsified. If we know that Tweety lives in Africa, we know that he cannot be a penguin. So, in light of this information, the hypothesis cannot longer be considered as derivable: $(penguin(Tweety), \Theta_1) \notin \mathcal{D}$. But the hypothesis still remains in the monotonic set of 'initially' plausible results: $(penguin(Tweety), \Theta_1) \in \mathcal{P}$.

So, if we define $A(\alpha)$ to denote the full conjunction,

$$A(\alpha) =_{def} A_1(\alpha) \wedge \ldots \wedge A_n(\alpha)$$

the first formal condition that could falsify the defeasible step will be

$$\forall \alpha (A_1(\alpha) \wedge \ldots \wedge A_n(\alpha) \supset B(\alpha)) \wedge B(\gamma) \wedge \neg A(\gamma).$$

To avoid self-explanations we will further add the condition that $A(\alpha)$ and $B(\alpha)$ share no predicates.

The reason why this condition also states the two premises of the abductive schema is because, in an adaptive logic, we can apply the conditional rule each time the disjunction is derivable. So, if we didn't state the two premises in the abnormality, we could derive anything as a hypothesis since $\vdash A(\gamma) \vee \neg A(\gamma)$ for any $A(\gamma)$. But with the current form, only hypotheses for which the two premises are true can be derived. This abnormality would already be sufficient to create an adaptive logic.

Still, we want to add some other defeating conditions. This could be done by replacing the abnormality by a disjunction of the already found abnormality and the other wanted conditions. Then, each time one of the conditions is derivable, the whole disjunction is derivable (by addition), and so, the formula defeated. But this result is obtained in the same way if we allow that one defeasible inference step adds more than one element to the condition instead of this complex disjunction. Hence, we will add these extra conditions in this way.

A lot of times, it is stated that the abduced hypothesis must be as parsimonious as possible. One of the main reasons for this is that one has to avoid random explanations. For instance, have a look at the following example:

$$\begin{aligned} \mathcal{H} &= \{\forall x(penguin(x) \supset \neg flies(x))\} \\ \mathcal{O} &= \{\neg flies(Tweety)\} \\ d &= \{x \mid x \text{ is a bird}\} \end{aligned}$$

The following formulas are derivable from this:

$$\begin{array}{rl} (\forall x(penguin(x) \wedge is\_green(x) \supset \neg flies(x)), \emptyset) & w \\ (penguin(Tweety) \wedge is\_green(x), \Theta_1) & w_1 \\ (is\_green(Tweety), \Theta_1) & w_1 \end{array}$$

The fact that $Tweety$ is green is not an explanation for the fact that $Tweety$ doesn't fly, nor is it something that follows from our background knowledge. Since we want to avoid that our abductions yield this kind of random hypotheses, we will add a mechanism to control that our hypothesis is the most parsimonious.

A final condition that we have to add is that our observation is not a tautology. Since we use a material implication, anything could be derived as an explanation for a tautology, because $\vdash B(\alpha) \supset \top$ for any $B(\alpha)$.

Now we can define the defeasible reasoning steps. Therefore we will need a new notation, which has the purpose to lift out one element from the conjunction $A_1(\alpha) \wedge \ldots \wedge A_n(\alpha)$. This will be used to check for more parsimonious explanations.

Notation 1 ($A_i^{-1}(\alpha)$)**.**

$$\begin{array}{lcl} \textit{if } n > 1 & : & A_i^{-1}(\alpha) =_{df} (A_1(\alpha) \wedge \ldots \wedge A_{i-1}(\alpha) \wedge \\ & & \quad A_{i+1}(\alpha) \wedge \ldots \wedge A_n(\alpha)) \\ \textit{if } n = 1 & : & A_1^{-1}(\alpha) =_{df} \top \end{array}$$

Definition 3. *The set of abnormalities Ω for an abductive system T is given by*

$$\begin{array}{rcl} \Omega & = & \{(\forall x(A_1(\alpha) \wedge \ldots \wedge A_n(\alpha) \supset B(\alpha)) \wedge B(\gamma) \wedge \neg A(\gamma)) \\ & & \vee\ \forall \alpha B(\alpha) \vee \bigvee_{i=1}^{n} \forall \alpha (A_i^{-1}(\alpha) \supset B(\alpha) \mid \gamma \in d, \\ & & \alpha \textit{ ranging over } d, A_i \textit{ and } B \textit{ literals}, B \notin \{A_i\}\} \end{array}$$

It is easily seen that the generic conditional rule for adaptive logics – as defined in section 1 – defined by this set of abnormalities is equivalent with the following inference rule that is written in the style of the Peircean schema stated above.

Definition 4. Defeasible Inference rule for Abduction

$$\begin{array}{llrl} (& \forall \alpha (A_1(\alpha) \wedge \ldots \wedge A_n(\alpha) \supset B(\alpha)), & \emptyset) & w \\ (& B(\gamma), & \emptyset) & w_i \\ \hline (& A_1(\gamma) \wedge \ldots \wedge A_n(\gamma), & \Theta) & w_{ij} \end{array}$$

with w_{ij} a new world hierarchically directly beneath w_i and $\Theta = \{\neg(A_1(\gamma) \wedge \ldots \wedge A_n(\gamma)), \forall \alpha B(\alpha),$
$\forall \alpha (A_1^{-1}(\alpha) \supset B(\alpha)), \ldots, \forall \alpha (A_n^{-1}(\alpha) \supset B(\alpha))\}$

So, it is possible to abduce further on hypothetical observations (and generate in that way further abductive extensions), but the implications need to be present in the background knowledge w. It is quite obvious, that if the abduced hypothesis is already abduced before (from, for instance, another implication), the resulting world-set will contain the same formulas, but with other conditions.

Finally, as explained in section 1, this body of definitions is formulated in the general framework of adaptive logics. This means that we have the following property.

Property 1. *The logic* $\mathbf{MLA^s}$ *is a fixed-point logic which has a sound and complete semantics with respect to its syntax.*

For the semantics and proof theory of this logic, and the proof that this logic is in the standard format of adaptive logics, we refer to [Gauderis, 2011]. For the soundness and completeness proof, we refer to the generic proof provided in [Batens, 2007] for all adaptive logics in standard format.

5 Example

Motivation and comparison with other approaches In this section we will consider an elaborate example of the dynamics of this framework. The main goal is to illustrate the key advantage of this approach, i.e. that there is no longer the need to recalculate all non-monotonic results at any stage of a growing knowledge base, but that one only needs to check the non-monotonic derivability of the needed formulas at a certain stage against the monotonic plausibility.

This is the main difference with other approaches to abduction such as the ones explicated in, for instance, [Paul, 2000], [Flach and Kakas, 2000] or [Kakas and Denecker, 2002]. Since these approaches focus on a fixed and not an expanding knowledge base, they require in cases of expansion a full recomputation to keep the set of derived non-monotonic results updated. It is not claimed that the adaptive approach yields better results than these other approaches in cases of a fixed knowledge base. In fact, it is an issue for future research to investigate whether the integration of the existing approaches for fixed knowledge bases with the adaptive approach does not yield better results.

Initial system $\mathcal{T}$ Our elaborate example will be an abductive learning situation about the observation of a non-flying bird, called Tweety. Initially, our abductive system $\mathcal{T} = (\mathcal{H}, \mathcal{O}, d)$ contains in addition to this observation only very limited background knowledge.

$$\begin{aligned}&\mathcal{H} = \{\forall x(penguin(x) \supset \neg flies(x)),\\ &\ \forall x(ostrich(x) \supset \neg flies(x))\}\\ &\mathcal{O} = \{\neg flies(Tweety)\}\\ &d = \{x \mid x \text{ is a bird}\}\end{aligned}$$

Thus, our background knowledge contains the following formulas:

$$(\forall x(penguin(x) \supset \neg flies(x)), \emptyset) \quad w \tag{1}$$

$$(\forall x(ostrich(x) \supset \neg flies(x)), \emptyset) \quad w \tag{2}$$

$$(\neg flies(Tweety), \emptyset) \quad w \tag{3}$$

And the following abductive hypotheses can be derived:

$$(penguin(Tweety), \Theta_1) \quad w_1 \tag{4}$$

$$(ostrich(Tweety), \Theta_2) \quad w_2 \tag{5}$$

with the sets Θ_1 and Θ_2 defined as

$$\begin{aligned}\Theta_1 &= \{\neg penguin(Tweety), \forall x\, \neg flies(x)\}\\ \Theta_2 &= \{\neg ostrich(Tweety), \forall x\, \neg flies(x)\}\end{aligned}$$

Since both implications have only one conjunct in the antecedent, their parsimony conditions – as defined in the general logical form – trivially coincide with the second condition. Since none of the conditions is deductively derivable in w, both (4) and (5) are elements of the set of derivable formulas $\mathcal{D}$.

First Extension $\mathcal{T}'$ At this stage, we discover that Tweety can swim, something we know ostriches can't.

$$\begin{aligned}&\mathcal{H}' = \mathcal{H} \cup \{\forall x(ostrich(x) \supset \neg swims(x))\},\\ &\mathcal{O}' = \mathcal{O} \cup \{swims(Tweety)\}\\ &d = \{x \mid x \text{ is a bird}\}\end{aligned}$$

From which the following formulas can be derived:

$$(\forall x(swims(x) \supset \neg ostrich(x)), \emptyset) \quad w \tag{6}$$

$$(\neg ostrich(Tweety), \emptyset) \quad w \tag{7}$$

Since the background information is extended, we only know that all previously derived hypotheses are still in the set of plausible hypotheses $\mathcal{P}$. If we want to check whether they are in the set of derivable hypotheses $\mathcal{D}$, we need to check whether their conditions are derivable from this extended information or not. But – this has already been cited several times as the key advantage of this system – we don't need to check all hypotheses. Since we don't have any further information on the penguin case, we just leave the hypothesis (4) for what it is. Thus, we save a computation, because at this stage we are not planning on reasoning or communicating on the penguin hypothesis. We just want to check whether this new information is a problem for the ostrich hypothesis; and indeed, it is easily seen that $(5) \notin \mathcal{D}'$.

Second Extension $\mathcal{T}''$ At this stage, we will investigate further the penguin hypothesis and retrieve additional background information about penguins.

$$\begin{aligned}&\mathcal{H}'' = \mathcal{H}' \cup \{\forall x(penguin(x) \supset eats\_fish(x)),\\ &\ \forall x(on\_south\_pole(x) \wedge in\_wild(x) \supset penguin(x))\}\\ &\mathcal{O}'' = \mathcal{O}'\\ &d = \{x \mid x \text{ is a bird}\}\end{aligned}$$

The following formulas can now further be retrieved:

$$(eats\_fish(Tweety), \Theta_1) \quad w_1 \tag{8}$$

$$(on\_south\_pole(Tweety), \Theta_{1.1}) \quad w_{1.1} \tag{9}$$

$$(in\_wild(Tweety), \Theta_{1.1}) \quad w_{1.1} \tag{10}$$

with the set $\Theta_{1.1}$ defined as

$$\begin{aligned}\Theta_{1.1} \;=\; &\{\neg(on\_south\_pole(Tweety) \wedge in\_wild(Tweety)),\\ &\forall x\, penguin(x),\\ &\forall x(on\_south\_pole(x) \supset penguin(x)),\\ &\forall x(in\_wild(x) \supset penguin(x))\}\end{aligned}$$

Since the first element of $\Theta_{1.1}$ is actually a disjunction, the first condition can even be split in two.

This stage is added to illustrate the other aspects of adaptive reasoning. Firstly, as (8) illustrates, there is no problem in reasoning further on previously deductively derived hypotheses. Only, to reason further, we must first check the condition of these hypotheses (This poses no problem here, because we can easily verify that $(4) \in \mathcal{D}''$). The deductively derived formula has the same conditions as the hypothesis on which it is built (and is contained in the same world). So, these results stand as long as the hypotheses on which assumption they are derived, hold. This characteristic of adaptive logics is very interesting, because it allows to derive predictions that can be tested in further investigation. In this example, we can test whether Tweety eats fish. In case this experiment fails and $\neg eats\_fish(Tweety)$ is added to the observations in the next stage, the hypothesis (and all results derived on its assumption) will be falsified. Secondly, the set of conditions $\Theta_{1.1}$ for the formulas (9) and (10) contains now also conditions that check for parsimony. Let us illustrate their functioning with the final extension.

Third Extension $\mathcal{T}'''$ At this stage, we learn that even in captivity the only birds that can survive on the South Pole are penguins. In addition to that, we get to know that Tweety is held in captivity.

$$\mathcal{H}''' = \mathcal{H}'' \cup \{\forall x(on\_south\_pole(x) \supset penguin(x))\},$$
$$\mathcal{O}''' = \mathcal{O}'' \cup \{\neg in\_wild(Tweety)\}$$
$$d = \{x \mid x \text{ is a bird}\}$$

If we now check the parsimony conditions of $\Theta_{1.1}$, we see that an element of this condition can be derived from our background knowledge. This means that all formulas assigned to world $w_{1.1}$ are not derivable anymore on this condition. Still, one might wonder whether this parsimony condition should not keep (9) and only withdraw (10). But, that this is not a good road is proven by the fact that in that case (10) would be falsified by the extra observation that Tweety does not live in the wild. In fact, that it was a good decision to withdraw the whole world $w_{1.1}$ is illustrated by the fact that the South Pole hypothesis of (9) can also be derived from $\mathcal{H}'''$ in another world.

$$(on\_south\_pole(Tweety), \Theta_{1.2}) \quad w_{1.2} \tag{11}$$

with the set $\Theta_{1.2}$ defined as

$$\Theta_{1.1} \quad = \quad \{\neg on\_south\_pole(Tweety), \forall x\, penguin(x)\}$$

So, at the end, we find that the set $\mathcal{D}'''$ of derivable formulas consists of all formulas derivable in the worlds w, w_1 and $w_{1.2}$. The formulas of w_2 and $w_{1.1}$ are not an element of this final set of derivable results.

6 Conclusion

In this article we presented a new logic-based approach to abduction which is based on the adaptive logics program. The main advantages of this approach are :

1. Each abduced formula is presented together with the specific conditions that would defeat it. In that way, it is not necessary to check the whole system for consistency after each extension of the background knowledge. Only the formulas that are needed at a certain stage need to be checked. Furthermore, it allows for the conditions to contain additional requirements, such as parsimony.
2. In comparison with other approaches that add conditions to formulas, the conditions are here fixed by a logical form and hence only determined by the (classical) premise set. In this way, there is no element of choice in stating conditions (as, for instance, in default logics).
3. By integrating a hierarchical system of sets, it provides an intuitive representation of multiple hypotheses without causing conflicts between contradictory hypotheses.
4. It allows for further deductive and abductive reasoning on previous retrieved abduced hypotheses.
5. The approach is based on a proper sound and complete fixed point logic ($\mathbf{MLA^s}$).

Limitations and Future Research It has been argued that these advantages make this approach apt for systems in which not all non-monotonic derivable results are needed at every stage of expansion of a knowledge base. Still, it needs to be examined whether an integration with existing systems (for a fixed knowledge base) do not yield better results. Furthermore, since the key feature of this approach is the saving of computations in expanding knowledge bases, it needs to be investigated whether there is no integration possible with assumption-based Truth Maintenance Systems (building on the ideas of [Reiter and de Kleer, 1987]).

References

[Batens, 2004] Diderik Batens. The need for adaptive logics in epistemology. In D. Gabbay, S. Rahman, J. Symons, and J.P. Van Bendegem, editors, *Logic, Epistemology and the Unity of Science*, pages 459–485. Kluwer Academic Publishers, Dordrecht, 2004.

[Batens, 2007] Diderik Batens. A universal logic approach to adaptive logics. *Logica Universalis*, 1:221–242, 2007.

[Brewka, 1991] Gerhard Brewka. Cumulative Default Logic. *Artificial Intelligence*, 50(2):183–205, 1991.

[Flach and Kakas, 2000] Peter A. Flach and Antonis C. Kakas. Abductive and Inductive Reasoning: Background and Issues. In Peter A. Flach and Antonis C. Kakas, editors, *Abduction and Induction. Essays on their Relation and their Integration*, volume 18 of *Applied Logic Series*, pages 1–27. Kluwer Academic Publishers, Dordrecht, 2000.

[García and Simari, 2004] Alejandro J. García and Guillermo R. Simari. Defeasible Logic Programming: An Argumentative Approach. *Theory and Practice of Logic Programming*, 4(1):95–2004, 2004.

[Gauderis, 2011] Tjerk Gauderis. Modelling Abduction in Science by means of a Modal Adaptive Logic. *Foundations of Science*, 2011. Forthcoming.

[Kakas and Denecker, 2002] Antonis Kakas and Marc Denecker. Abduction in Logic Programming. In A. Kakas and F. Sadri, editors, *Computational Logic: Logic Programming and Beyond. Part I*, pages 402–436. Springer Verlag, 2002.

[Paul, 2000] Gabriele Paul. AI Approaches to Abduction. In Dov M. Gabbay and Rudolf Kruse, editors, *Abductive Reasoning and Uncertainty Management Systems*, volume 4 of *Handbook of Defeasible Reasoning and Uncertainty Management Systems*, pages 35–98. Kluwer Academic Publishers, Dordrecht, 2000.

[Peirce, 1960] Charles S. Peirce. *Collected Papers*. Belknap Press of Harvard University Press, Cambridge, Massachusetts, 1960.

[Reiter and de Kleer, 1987] Raymond Reiter and Johan de Kleer. Foundations of Assumption-based Truth Maintenance Systems: Preliminary Report. In *Proceedings of the Sixth National Conference on Artificial Intelligence (AAAI'87)*, pages 183–188, 1987.

Default Reasoning about Conditional, Non-Local and Disjunctive Effect Actions

Hannes Strass
Institute of Computer Science
University of Leipzig
strass@informatik.uni-leipzig.de

Abstract

Recently, Baumann et al. [2010] provided a comprehensive framework for default reasoning about actions. Alas, the approach was only defined for a very basic class of domains where all actions have mere unconditional, local effects. In this paper, we show that the framework can be substantially extended to domains with action effects that are conditional (i.e. are context-sensitive to the state in which they are applied), non-local (i.e. the range of effects is not pre-determined by the action arguments) and even disjunctive (thus nondeterministic). Notably, these features can be carefully added without sacrificing important nice properties of the basic framework, such as modularity of domain specifications or existence of extensions.

1 Introduction

Reasoning about actions and non-monotonic reasoning are two important fields of logic-based knowledge representation and reasoning. While reasoning about actions deals with dynamic domains and their evolution over time, default reasoning is usually concerned with closing gaps in incomplete static knowledge bases. Both areas have received considerable attention and have reached remarkable maturity by now. However, a unifying approach that combines the full expressiveness of both fields was still lacking, until a recent paper [Baumann *et al.*, 2010] took an important first step into the direction of uniting these two lines of research. There, a logical framework was proposed that lifted default reasoning about a domain to a temporal setting where defaults, action effects and the frame assumption interact in a well-defined way.

In this paper, we develop a substantial extension of their work: we significantly generalise the theoretical framework to be able to deal with a broad class of action domains where effects may be conditional, non-local and non-deterministic. As we will show in the paper, extending the approach to conditional effects is straightforward. However, retaining their construction of defaults leads to counterintuitive conclusions. Roughly, this is due to eager default application in the presence of incomplete knowledge about action effects. As an example, consider the classical drop action that breaks fragile objects. In the presence of a (simple) state default expressing that objects are to be considered not broken unless there is information to the contrary, this could lead to the following reasoning: After dropping an object x of which nothing further is known, we can apply the default and infer it is not broken. But this means it cannot have been fragile before (since otherwise it *would* be broken). This line of reasoning violates the principle of causality: while a fragile object will be broken after dropping it, this does not mean that objects should be assumed not fragile *before* dropping them. We will formally define when such undesired inferences arise and devise a modification to the basic framework that provably disables them. Interestingly, the counterintuitive consequences occur already with conditional, local-effect actions; our modification however prevents them also for actions with nondeterministic, non-local effects. Since the introduction of effect preconditions represents our most significant change, we will prove that it is a proper generalisation of the original framework: for all action default theories with only unconditional, local effect actions, the "old" and "new" approach yield the same results. For the subsequent extensions it will be straightforward to see that they are proper generalisations.

The paper proceeds as follows. In the next section, we provide the necessary background. The sections thereafter extend the basic approach introduced in [Baumann *et al.*, 2010] by conditional effects (Section 3), non-local effects (Section 4) and disjunctive effects (Section 5). In the penultimate section, we prove several desirable properties of the extended framework; Section 7 discusses related work and concludes.

2 Background

2.1 Unifying Action Calculus

The unifying action calculus (UAC) was proposed in [Thielscher, 2011] to allow for a treatment of problems in reasoning about actions that is independent of a particular calculus. It is based on a finite, sorted logic language with equality which includes the sorts FLUENT, ACTION and TIME along with the predicates $<$: TIME × TIME, that denotes a (possibly partial) ordering on time points; *Holds* : FLUENT × TIME, that is used to state that a fluent is true at a given time point; and *Poss* : ACTION × TIME × TIME, expressing that an action is possible for given starting and ending time points.

As a most fundamental notion in the UAC, a *state formula*

$\Phi[\vec{s}]$ *in* $\vec{s}$ is a first-order formula with free TIME variables $\vec{s}$ where (1) for each occurrence of $Holds(f, s)$ in $\Phi[\vec{s}]$ we have $s \in \vec{s}$ and (2) predicate *Poss* does not occur. State formulas allow to express properties of action domains at given time points. Although this definition is quite general in that it allows an arbitrary finite sequence of time points, for our purposes two time points will suffice. For a function A into sort ACTION, a *precondition axiom for* $A(\vec{x})$ is of the form

$$Poss(A(\vec{x}), s, t) \equiv \pi_A[s] \tag{1}$$

where $\pi_A[s]$ is a state formula in s with free variables among $s, t, \vec{x}$. The formula $\pi_A[s]$ thus defines the necessary and sufficient conditions for the action A to be applicable for the arguments $\vec{x}$ at time point s, resulting in t. The UAC also provides a general form for effect axioms; we however omit this definition because we only use a special form of effect axioms here. The last notion we import formalises how action domains are axiomatised in the unifying action calculus.

Definition 1. A *(UAC) domain axiomatisation* consists of a finite set of foundational axioms Ω defining a time structure, a set Π of precondition axioms (1) and a set Υ of effect axioms; the latter two for all functions into sort ACTION; lastly, it contains uniqueness-of-names axioms for all finitely many function symbols into sorts FLUENT and ACTION.

The foundational axioms Ω serve to instantiate the UAC by a concrete time structure, for example the branching situations with their usual ordering from the situation calculus. We restrict our attention to domains that make intuitive sense; one of the basic things we require is that actions actually consume time: A domain axiomatisation is *progressing*, if $\Omega \models (\exists s : \text{TIME})(\forall t : \text{TIME}) s \leq t$ and $\Omega \cup \Pi \models Poss(a, s, t) \supset s < t$. Here, we are only concerned with progressing domain axiomatisations; we use the macro $Init(t) \stackrel{\text{def}}{=} \neg(\exists s) s < t$ to refer to the unique initial time point.

For presentation purposes, we will make use of the concept of *fluent formulas*, where terms of sort FLUENT play the role of atomic formulas, and complex formulas can be built using the usual first-order constructors. For a fluent formula Φ, we will denote by $\Phi[s]$ the state formula that is obtained by replacing all fluent literals $[\neg]f$ in Φ by $[\neg]Holds(f, s)$. The operator $|\cdot|$ will be used to extract the affirmative component of a fluent literal, that is, $|\neg f| = |f| = f$; the polarity of a fluent literal is given by $sign(\neg f) = -$ and $sign(f) = +$.

2.2 Default Logic

Default logic as introduced by [Reiter, 1980] uses *defaults* to extend incomplete world knowledge. They are of the form[1]

$$\frac{\alpha : \beta}{\gamma} \qquad \text{(shorthand: } \alpha : \beta/\gamma\text{)}$$

Here, α, the *prerequisite*, the β, the *justification*, and γ, the *consequent*, are first-order formulas. These expressions are to be read as "whenever we know α and nothing contradicts β, we can safely conclude γ". A default is *normal* if $\beta = \gamma$, that is, justification and consequent coincide. A default is *closed* if its prerequisite, justification and consequent are sentences, that is, have no free variables; otherwise, it is *open*.

The semantics of defaults is defined via the notion of extensions for default theories. A *default theory* is a pair (W, D), where W is a set of sentences in first-order logic and D is a set of defaults. A default theory is *closed* if all its defaults are closed; otherwise, it is *open*. For a set T of formulas, we say that a default $\alpha : \beta/\gamma$ is *applicable to* T iff $\alpha \in T$ and $\neg\beta \notin T$; we say that the default has been *applied to* T if it is applicable and additionally $\gamma \in T$. Extensions for a default theory (W, D) are deductively closed sets of formulas which contain all elements of W, are closed under application of defaults from D and which are grounded in the sense that each formula in them has a non-cyclic derivation. For closed default theories this is captured by the following definition.

Definition 2 (Theorem 2.1, [Reiter, 1980]). Let (W, D) be a closed default theory and E be a set of closed formulas. Define $E_0 \stackrel{\text{def}}{=} W$ and $E_{i+1} \stackrel{\text{def}}{=} Th(E_i) \cup D_i$ for $i \geq 0$, where

$$D_i \stackrel{\text{def}}{=} \left\{ \gamma \;\middle|\; \frac{\alpha : \beta}{\gamma} \in D, \alpha \in E_i, \neg\beta \notin E \right\}$$

Then E is an *extension for* (W, D) iff $E = \bigcup_{i=0}^{\infty} E_i$.

We will interpret open defaults as schemata representing all of their ground instances. Therefore, open default theories can be viewed as shorthand notation for their closed counterparts.[2] When we use an extension E or set of defaults D with an integer subscript, we refer to the E_i and D_i from above. We write $(W, D) \mathrel{|\!\!\approx} \Psi$ to express that the formula Ψ is contained in each extension of the default theory (W, D).

2.3 Default Reasoning in Action Domains with Unconditional, Local Effect Actions

The approach of [Baumann *et al.*, 2010] combines default logic with the unifying action calculus: domain axiomatisations are viewed as incomplete knowledge bases that are completed by defaults. It takes as input a description of a particular action domain with normality statements. This description comprises the following: (1) a domain signature, that defines the vocabulary of the domain; (2) a description of the direct effects of actions; (3) a set of *state defaults* $\Phi \rightsquigarrow \psi$, constructs that specify conditions Φ under which a fluent literal ψ normally holds in the domain.[3]

The state defaults from the domain description are translated into Reiter defaults, where the special predicates $DefT(f, s, t)$ and $DefF(f, s, t)$ are used to express that a fluent f becomes normally true (false) from s to t.[4] For each state default δ, two Reiter defaults are created: δ_{Init}, that is used for default conclusions about the initial time point; and δ_{Reach}, that is used for default conclusions about time points that can be reached via action application.

[1] Reiter [1980] introduces a more general version of defaults with an arbitrary number of justifications, which we do not need here.

[2] Free variables of formulas not in a default will however be implicitly universally quantified from the outside.

[3] Here, Φ, the *prerequisite*, is a fluent formula; ψ, the *consequent*, being a fluent *literal* also allows to express that a fluent normally does *not* hold in the domain.

[4] It should be noted that $DefF(f, s, t)$ is not the same as $\neg DefT(f, s, t)$ – the latter only means that f becomes not normally true from s to t.

Definition 3. Let $\delta = \Phi \rightsquigarrow \psi$ be a state default.

$$\delta_{Init} \stackrel{\text{def}}{=} \frac{Init(t) \wedge \Phi[t] : \psi[t]}{\psi[t]} \tag{2}$$

$$\delta_{Reach} \stackrel{\text{def}}{=} \frac{Pre_{\delta}(s,t) : Def(\psi, s, t)}{Def(\psi, s, t)} \tag{3}$$

$$Pre_{\delta}(s,t) \stackrel{\text{def}}{=} \Phi[t] \wedge \neg(\Phi[s] \wedge \neg\psi[s])$$

$$Def(\psi, s, t) \stackrel{\text{def}}{=} \begin{cases} DefT(\psi, s, t) & \text{if } \psi = |\psi| \\ DefF(|\psi|, s, t) & \text{otherwise} \end{cases}$$

For a set Δ of state defaults, the corresponding defaults are

$$\Delta_{Init} \stackrel{\text{def}}{=} \{\delta_{Init} \mid \delta \in \Delta\} \text{ and } \Delta_{Reach} \stackrel{\text{def}}{=} \{\delta_{Reach} \mid \delta \in \Delta\}.$$

For the *Reach* defaults concerning two time points s, t connected via action application, we ensure that the state default δ was not violated at the starting time point s by requiring $\neg(\Phi[s] \wedge \neg\psi[s])$ in the prerequisite.[5] The consequent is then inferred unless there is information to the contrary.

Being true (or false) by default is then built into the effect axiom by accepting it as a possible "cause" to determine a fluent's truth value. The other causes are the ones already known from monotonic formalisms for reasoning about actions: direct action effects, and a notion of persistence that provides a solution to the frame problem [McCarthy and Hayes, 1969].

Definition 4. Let f : FLUENT and s, t : TIME be variables. The following macros express that f persists from s to t:

$$FrameT(f, s, t) \stackrel{\text{def}}{=} Holds(f, s) \wedge Holds(f, t) \tag{4}$$

$$FrameF(f, s, t) \stackrel{\text{def}}{=} \neg Holds(f, s) \wedge \neg Holds(f, t) \tag{5}$$

Let A be a function into sort ACTION and Γ_A be a set of fluent literals with free variables in $\vec{x}$ that denote the positive and negative direct effects of $A(\vec{x})$, respectively. The following pair of macros expresses that f is a direct effect of $A(\vec{x})$:

$$DirectT(f, A(\vec{x}), s, t) \stackrel{\text{def}}{=} \bigvee_{F(\vec{x}') \in \Gamma_A,\, \vec{x}' \subseteq \vec{x}} f = F(\vec{x}') \tag{6}$$

$$DirectF(f, A(\vec{x}), s, t) \stackrel{\text{def}}{=} \bigvee_{\neg F(\vec{x}') \in \Gamma_A,\, \vec{x}' \subseteq \vec{x}} f = F(\vec{x}') \tag{7}$$

An *effect axiom with unconditional effects, the frame assumption and normal state defaults* is of the form

$$\begin{aligned} & Poss(A(\vec{x}), s, t) \supset \\ & \quad (\forall f)(Holds(f, t) \equiv CausedT(f, A(\vec{x}), s, t)) \wedge \\ & \quad (\forall f)(\neg Holds(f, t) \equiv CausedF(f, A(\vec{x}), s, t)) \end{aligned} \tag{8}$$

where

$$\begin{aligned} CausedT(f, A(\vec{x}), s, t) \stackrel{\text{def}}{=} {} & DirectT(f, A(\vec{x}), s, t) \vee \\ & FrameT(f, s, t) \vee DefT(f, s, t) \end{aligned} \tag{9}$$

$$\begin{aligned} CausedF(f, A(\vec{x}), s, t) \stackrel{\text{def}}{=} {} & DirectF(f, A(\vec{x}), s, t) \vee \\ & FrameF(f, s, t) \vee DefF(f, s, t) \end{aligned} \tag{10}$$

Note that a default conclusion of a state property in a non-initial state crucially depends on an action execution leading to that state. Hence, whenever it is definitely known that $Holds(f, t)$ after $Poss(a, s, t)$, it follows from the effect axiom that $\neg DefF(f, s, t)$; a symmetrical argument applies if $\neg Holds(f, t)$. This means that definite knowledge about a fluent inhibits the opposite default conclusion. But observe that the addition of *DefT* and *DefF* as "causes" to the effect axiom weakened the solution to the frame problem established earlier. The following definition ensures that the persistence assumption is restored in its full generality.

Definition 5. Let Δ be a set of state defaults, ψ be a fluent literal and s, t be variables of sort TIME. The *default closure axiom for* ψ *with respect to* Δ is

$$\left(\bigwedge_{\Phi \rightsquigarrow \psi \in \Delta} \neg Pre_{\Phi \rightsquigarrow \psi}(s, t) \right) \supset \neg Def(\psi, s, t) \tag{11}$$

For a fluent literal ψ not mentioned as a consequent in Δ the default closure axiom is just $\top \supset \neg Def(\psi, s, t)$. Given a domain axiomatisation Σ and a set Δ of state defaults, we denote by Σ_{Δ} the default closure axioms with respect to Δ and the fluent signature of Σ.

The fundamental notion of the solution to the state default problem by [Baumann *et al.*, 2010] is now a default theory where the incompletely specified world consists of a UAC domain axiomatisation augmented by suitable default closure axioms. The default rules are the automatic translations of user-specified, domain-dependent state defaults. For a domain axiomatisation Σ and a set Δ of state defaults, the corresponding *domain axiomatisation with state defaults* is the pair $(\Sigma \cup \Sigma_{\Delta}, \Delta_{Init} \cup \Delta_{Reach})$. We use a well-known example domain [Reiter, 1991] to illustrate the preceding definitions. To ease the presentation, in this example we instantiate the UAC to the branching time structure of situations.

Example 1 (Breaking Objects)**.** Imagine a robot that can move around and carry objects, among them a vase. When the robot drops an object x, it does not carry x any more and additionally x is broken. Usually, however, objects are not broken unless there is information to the contrary.

The fluents that we use to describe this domain are $\mathsf{Carries}(x)$ (the robot carries x) and $\mathsf{Broken}(x)$ (x is broken); the only function of sort ACTION is $\mathsf{Drop}(x)$. Dropping an object is possible if and only if the robot carries the object:

$$\begin{aligned} & Poss(\mathsf{Drop}(x), s, t) \equiv \\ & \qquad Holds(\mathsf{Carries}(x), s) \wedge t = Do(\mathsf{Drop}(x), s) \end{aligned}$$

The effects of dropping an object x are given by the set

$$\Gamma_{\mathsf{Drop}(x)} = \{\neg \mathsf{Carries}(x), \mathsf{Broken}(x)\}$$

The set of state defaults $\Delta^{break} = \{\top \rightsquigarrow \neg \mathsf{Broken}(x)\}$ says that objects are normally not broken. Applying the definitions from above to this specification results in the domain axiomatisation with defaults $(\Sigma^{break} \cup \Sigma^{break}_{\Delta}, \Delta^{break}_{Init} \cup \Delta^{break}_{Reach})$, where Σ^{break} contains effect axiom (8) and the above precondition axiom for Drop, the set Δ^{break}_{Init} contains only

$$\frac{Init(t) : \neg Holds(\mathsf{Broken}(x), t)}{\neg Holds(\mathsf{Broken}(x), t)}$$

[5] The reason for this is to prevent application of initially definitely violated state defaults through irrelevant actions. A default violation occurs when the prerequisite $\Phi[s]$ of a state default δ is known to be met, yet the negation of the consequent prevails, $\neg\psi[s]$.

and the defaults Δ^{break}_{Reach} for action application consist of

$$\frac{\neg Holds(\mathsf{Broken}(x), s) : DefF(\mathsf{Broken}(x), s, t)}{DefF(\mathsf{Broken}(x), s, t)}$$

Finally, the default closure axioms for the fluent Broken are $Holds(\mathsf{Broken}(x), s) \supset \neg DefF(\mathsf{Broken}(x), s, t)$ and $\neg DefT(\mathsf{Broken}(x), s, t)$, and $\neg Def(\psi, s, t)$ for all other fluent literals ψ. With $S_1 \stackrel{\text{def}}{=} Do(\mathsf{Drop}(\mathsf{Vase}), S_0)$, the default theory sanctions the sceptical conclusions that the vase is initially not broken, but is so after dropping it:

$$(\Sigma^{break} \cup \Sigma^{break}_{\Delta}, \Delta^{break}_{Init} \cup \Delta^{break}_{Reach}) \approx\!\!\!|$$
$$\neg Holds(\mathsf{Broken}(\mathsf{Vase}), S_0) \wedge Holds(\mathsf{Broken}(\mathsf{Vase}), S_1)$$

One of the main theoretical results of [Baumann *et al.*, 2010] was the guaranteed existence of extensions for the class of domain axiomatisations with defaults considered there. As we will see later on, a similar result holds for our generalisation of the theory.

Proposition 1 (Theorem 4, [Baumann *et al.*, 2010]). *Let Σ be a domain axiomatisation and Δ be a set of state defaults. Then the corresponding domain axiomatisation with state defaults $(\Sigma \cup \Sigma_\Delta, \Delta_{Init} \cup \Delta_{Reach})$ has an extension. If furthermore Σ is consistent, then so are all extensions for $(\Sigma \cup \Sigma_\Delta, \Delta_{Init} \cup \Delta_{Reach})$.*

3 Conditional Effects

We first investigate how the default reasoning framework of [Baumann *et al.*, 2010] can be extended to conditional effect actions. As we will show, there is subtle interdependence between conditional effects and default conclusions, which requires a revision of the defaults constructed in Definition 3. We begin by formalising how to represent conditional effects in the domain specification language. Recall that in the unconditional case, action effects were just literals denoting the positive and negative effects. In the case of conditional effects, theses literals are augmented with a fluent formula that specifies the conditions under which the effect materialises.

Definition 6. A *conditional effect expression* is of the form Φ/ψ, where Φ is a fluent formula and ψ a fluent literal. Φ/ψ is called *positive* if $sign(\psi) = +$ and *negative* if $sign(\psi) = -$. For an action A and sequence of variables $\vec{x}$ matching A's arity, a conditional effect expression ε is called *local for* $A(\vec{x})$ iff all free variables in ε are among $\vec{x}$.

Throughout the paper, we will assume given a set $\Gamma_{A(\vec{x})}$ of conditional effect expressions for each function A into sort ACTION with matching sequence of variables $\vec{x}$. Such a set $\Gamma_{A(\vec{x})}$ is called *local-effect* if all $\varepsilon \in \Gamma_{A(\vec{x})}$ are local for $A(\vec{x})$. By $\Gamma^+_{A(\vec{x})}$ we refer to the positive, by $\Gamma^-_{A(\vec{x})}$ to the negative elements of $\Gamma_{A(\vec{x})}$.

With this specification of action effects, it is easy to express the implication "effect precondition implies effect" via suitable formulas. For this purpose, we introduce the new predicates *DirT* and *DirF*. Intuitively, $DirT(f, a, s, t)$ says that f is a direct positive effect of action a from s to t; symmetrically, $DirF(f, a, s, t)$ says that f is a direct *negative* effect.[6]

Definition 7. Let $\varepsilon = \Phi/\psi$ be a conditional effect expression and f : FLUENT and s, t : TIME be variables. The following macro expresses that ε *has been activated for f from s to t*:[7]

$$Activated_\varepsilon(f, s, t) \stackrel{\text{def}}{=} (f = |\psi| \wedge \Phi[s])$$

Let A be a function into sort ACTION with a set of conditional effect expressions $\Gamma_{A(\vec{x})}$ that is local-effect. The *direct positive and negative effect formulas for* $A(\vec{x})$ are

$$DirT(f, A(\vec{x}), s, t) \equiv \bigvee_{\varepsilon \in \Gamma^+_{A(\vec{x})}} Activated_\varepsilon(f, s, t) \qquad (12)$$

$$DirF(f, A(\vec{x}), s, t) \equiv \bigvee_{\varepsilon \in \Gamma^-_{A(\vec{x})}} Activated_\varepsilon(f, s, t) \qquad (13)$$

An *effect axiom with conditional effects, the frame assumption and normal state defaults* is of the form (8), where

$$CausedT(f, A(\vec{x}), s, t) \stackrel{\text{def}}{=} DirT(f, A(\vec{x}), s, t) \vee FrameT(f, s, t) \vee DefT(f, s, t) \qquad (14)$$

$$CausedF(f, A(\vec{x}), s, t) \stackrel{\text{def}}{=} DirF(f, A(\vec{x}), s, t) \vee FrameF(f, s, t) \vee DefF(f, s, t) \qquad (15)$$

The only difference between the effect axioms of [Baumann *et al.*, 2010] and the effect axioms defined here is the replacement of their macros *DirectT*, *DirectF* for unconditional direct effects with the predicates *DirT*, *DirF* for conditional effects. In the following, we will understand domain axiomatisations to contain – for each action – effect axioms of the form (8) along with the respective direct positive and negative effect formulas. To ease notation, for predicates with an obvious polarity (like *DirT*, *DirF*), we use a neutral version (like *Dir*) with fluent literals L, where $Dir(L, a, s, t)$ denotes $DirF(F, a, s, t)$ if $L = \neg F$ for some fluent F and $DirT(L, a, s, t)$ otherwise.

While this extended definition of action effects is straightforward, it severely affects the correctness of default reasoning in the action theory: as the following example shows, one cannot naïvely take this updated version of the effect axioms and use the Reiter defaults as before.

Example 1 (Continued). We add a unary fluent Fragile with the obvious meaning and modify the Drop action such that dropping only breaks objects that are fragile: $\Gamma_{\mathsf{Drop}(x)} = \{\top/\neg\mathsf{Carries}(x), \mathsf{Fragile}(x)/\mathsf{Broken}(x)\}$. Assume that all we know is that the robot initially carries the vase, $Holds(\mathsf{Carries}(\mathsf{Vase}), S_0)$. As before, the effect axiom tells us that the robot does not carry the vase any more at S_1. Additionally, since we do not know whether the vase was fragile at S_0, there is no reason to believe that it is broken after dropping it, hence $\neg\mathsf{Broken}(\mathsf{Vase})$ still holds by default at S_1. But now, due to the presence of conditional effects, the effect axiom for $\mathsf{Drop}(\mathsf{Vase})$ clearly entails $\neg Holds(\mathsf{Broken}(\mathsf{Vase}), S_1) \supset \neg Holds(\mathsf{Fragile}(\mathsf{Vase}), S_0)$,[8]

[6] Notice that these new predicates are in contrast to Definition 4, where *DirectT* and *DirectF* are merely syntactic sugar.

[7] The second time argument t of macro $Activated_\varepsilon(f, s, t)$ will only be needed later when we introduce non-deterministic effects.

[8] This is just the contrapositive of the implication expressed by the effect axiom.

and thus we can draw the conclusion

$$(\Sigma^{break} \cup \Sigma^{break}_{\Delta}, \Delta^{break}_{Init} \cup \Delta^{break}_{Reach}) \approx\!\!\!\!| \;\; \neg \mathit{Holds}(\mathsf{Fragile}(\mathsf{Vase}), S_0)$$

This is undesired as it lets us conclude something about the present (S_0) using knowledge about the future (S_1) which we could not conclude using only knowledge and default knowledge about the present (there is no default that could conclude $\neg\mathsf{Fragile}(\mathsf{Vase})$).

The flaw with this inference is that it makes default conclusions about a fluent whose truth value is affected by an action at the same time. This somewhat contradicts our intended usage of defaults about states: we originally wanted to express reasonable assumptions about fluents whose values are unknown.

Generalising the example, the undesired behaviour occurs whenever there exists a default $\Phi_D \rightsquigarrow \psi$ with conclusion ψ whose negation $\neg\psi$ might be brought about by a conditional effect $\Phi_C/\neg\psi$. The faulty inference then goes like this:

$$\Phi_D[t] \supset \mathit{Def}(\psi, s, t) \supset \psi[t] \supset \neg\mathit{Dir}(\neg\psi, s, t) \supset \neg\Phi_C[s]$$

Obviously, this inference is only undesired if there is no information about the effect's precondition at the starting time point of the action. This motivates our formal definition of the conditions under which a so-called *conflict* between an action effect and a default conclusion arises.

Definition 8. Let (Σ, Δ) be a domain axiomatisation with defaults, E be an extension for (Σ, Δ), α be a ground action and $\delta = \Phi \rightsquigarrow \psi$ be a ground state default. We say that there is a *conflict between* α *and* δ *in* E iff there exist ground time points σ and τ such that for some $i \geq 0$ we have

1. (a) $E_i \not\models \mathit{Poss}(\alpha, \sigma, \tau) \supset \neg\mathit{Dir}(\neg\psi, \alpha, \sigma, \tau)$
   (b) $E_i \not\models \mathit{Def}(\psi, \alpha, \sigma, \tau)$
2. (a) $E_{i+1} \models \mathit{Poss}(\alpha, \sigma, \tau) \supset \neg\mathit{Dir}(\neg\psi, \alpha, \sigma, \tau)$
   (b) $E_{i+1} \models \mathit{Def}(\psi, \sigma, \tau)$

In words, a conflict arises in an extension if up to some stage i, before we make the default conclusion ψ, we cannot conclude the effect $\neg\psi$ will not occur (1); after concluding ψ by default, we infer that $\neg\psi$ cannot occur as direct effect (2). We can now go back to the example seen earlier and verify that the counter-intuitive conclusion drawn there was indeed due to a conflict in the sense of the above definition.

Example 1 (Continued)**.** Consider the only extension E^{break} for $(\Sigma^{break} \cup \Sigma^{break}_{\Delta}, \Delta^{break}_{Init} \cup \Delta^{break}_{Reach})$. Before applying any defaults whatsoever, we know that dropping the vase is possible: $E^{break}_0 \models \mathit{Poss}(\mathsf{Drop}(\mathsf{Vase}), S_0, S_1)$; but we do not know if the vase is fragile and hence $E^{break}_0 \not\models \neg\mathit{DirT}(\mathsf{Broken}(\mathsf{Vase}), \mathsf{Drop}(\mathsf{Vase}), S_0, S_1)$ (item 1). After applying all the defaults, we know that the vase is not broken at S_1: $E^{break}_1 \models \mathit{DefF}(\mathsf{Broken}(\mathsf{Vase}), S_0, S_1)$. Hence, it cannot have been broken by dropping it in S_0, that is, $E^{break}_1 \models \neg\mathit{DirT}(\mathsf{Broken}(\mathsf{Vase}), \mathsf{Drop}(\mathsf{Vase}), S_0, S_1)$ (item 2), thus cannot have been fragile in the initial situation.

In the following, we will modify the definition of Reiter defaults from [Baumann *et al.*, 2010] to eliminate the possibility of such conflicts. The underlying idea is to apply a default only if it is known that a conflict cannot arise, that is, if it is known that the contradictory direct effect cannot materialise. To this end, we extend the original default prerequisite $\mathit{Pre}_\delta(s, t) = \Phi[t] \wedge \neg(\Phi[s] \wedge \neg\psi[s])$ that only requires the precondition to hold and the default not to be violated previously: we will additionally stipulate that any action a happening at the same time cannot create a conflict.

Definition 9. Let $\delta = \Phi \rightsquigarrow \psi$ be a state default and s, t : TIME be variables.

$$\mathit{Safe}_\delta(s, t) \stackrel{\text{def}}{=} (\forall a)(\mathit{Poss}(a, s, t) \supset \neg\mathit{Dir}(\neg\psi, a, s, t))$$

$$\delta_{\mathit{Poss}} \stackrel{\text{def}}{=} \frac{\mathit{Pre}_\delta(s, t) \wedge \mathit{Safe}_\delta(s, t) : \mathit{Def}(\psi, s, t)}{\mathit{Def}(\psi, s, t)} \qquad (16)$$

For a set Δ of state defaults, $\Delta_{\mathit{Poss}} \stackrel{\text{def}}{=} \{\delta_{\mathit{Poss}} \mid \delta \in \Delta\}$.

In the example domain, applying the above definition yields the following.

Example 1 (Continued)**.** For the state default δ^{break} saying that objects are usually not broken, we have $\mathit{Safe}_{\delta^{break}}(s, t) = (\forall a)(\mathit{Poss}(a, s, t) \supset \neg\mathit{DirT}(\mathsf{Broken}(x), a, s, t))$. This expresses that the state default can be safely applied from s to t whenever for any action a happening at the same time, it is known that a does not cause a violation of this default at the ending time point t. The resulting default $\delta^{break}_{\mathit{Poss}}$ is

$$\frac{\neg\mathit{Holds}(\mathsf{Broken}(x), s) \wedge \mathit{Safe}_{\delta^{break}}(s, t) : \mathit{DefF}(\mathsf{Broken}(x), s, t)}{\mathit{DefF}(\mathsf{Broken}(x), s, t)}$$

As we will see later (Theorem 3), the default closure axioms $\neg\mathit{Pre}_{\Phi \rightsquigarrow \psi}(s, t) \supset \neg\mathit{Def}(\psi, s, t)$ for preserving the commonsense principle of inertia in the presence of inapplicable defaults need not be modified. With our new defaults, we can now redefine the concept of a domain axiomatisation with defaults for conditional effect actions.

Definition 10. Let Σ be a domain axiomatisation where the effect axioms are given by Definition 7 and let Δ be a set of state defaults. The corresponding *domain axiomatisation with defaults* is the pair $(\Sigma \cup \Sigma_\Delta, \Delta_{\mathit{Init}} \cup \Delta_{\mathit{Poss}})$.

The direct effect formulas that determine *DirT* and *DirF* will be redefined twice in this paper. We will understand the above definition to be retrofitted with their latest version. The extension to conditional effects is a proper generalisation of the original approach of Section 2.3 for the special case of unconditional effect actions, as is shown below.

Theorem 2. *Consider a domain axiomatisation with only unconditional action effects and a set* Δ *of state defaults. Let* $\Xi_1 = (\Sigma \cup \Sigma_\Delta, \Delta_{\mathit{Init}} \cup \Delta_{\mathit{Reach}})$ *be the corresponding domain axiomatisation with defaults of [Baumann* et al.*, 2010], and let* $\Xi_2 = (\Sigma' \cup \Sigma_\Delta, \Delta_{\mathit{Init}} \cup \Delta_{\mathit{Poss}})$ *be the domain axiomatisation with defaults according to Definition 10. For a state formula* Ψ *and time point* τ*, we have* $\Xi_1 \approx\!\!\!\!| \; \Psi[\tau]$ *iff* $\Xi_2 \approx\!\!\!\!| \; \Psi[\tau]$.

Proof sketch. For unconditional effects, a ground *Dir* atom is by Definition 7 equivalent to the corresponding *Direct* macro, hence the effect axioms of the two approaches are equivalent. Furthermore, the truth values of ground *DirT* and *DirF* atoms are always fixed, and consequently each Reiter default (16) defined above is applicable whenever the original *Reach* default (3) of [Baumann *et al.*, 2010] is applicable. □

4 Non-Local Effects

Up to here, conditional effect expressions for an action $A(\vec{x})$ were restricted to contain only variables among $\vec{x}$. Considering a ground instance $A(\vec{\varsigma})$ of an action, this means that the set of objects that can possibly be affected by this action is already fixed to $\vec{\varsigma}$. This is a restriction because it can make the specification of certain actions at least cumbersome or utterly impossible, for example actions that affect a vast number of (or all of the) domain elements at once.

The gain in expressiveness when allowing non-local action effects comes at a relatively low cost: it suffices to allow additional free variables $\vec{y}$ in the conditional effect expressions. They represent the objects that may be affected by the action without being among the action arguments $\vec{x}$.

Definition 11. Let A be a function into sort ACTION and $\vec{x}$ a sequence of variables matching A's arity. Let ε be a conditional effect expression of the form $\Phi/F(\vec{x}', \vec{y})$ or $\Phi/\neg F(\vec{x}', \vec{y})$ with free variables $\vec{x}', \vec{y}$, where $\vec{x}' \subseteq \vec{x}$ and $\vec{y}$ is disjoint from $\vec{x}$.

For variables f : FLUENT and s, t : TIME, the following macro expresses that ε *has been activated for f from s to t*:

$$\mathit{Activated}_\varepsilon(f, s, t) \overset{\text{def}}{=} (\exists \vec{y})(f = F(\vec{x}', \vec{y}) \wedge \Phi[s])$$

The *direct positive and negative effect formulas* are of the form (12) and (13).

Note that according to this definition, free variables $\vec{y}$ are quantified existentially when they occur in the context Φ and universally when they occur in the consequence ψ. They thus not only express non-local effects but also non-local contexts.

Example 2 (Exploding Bomb [Reiter, 1991]). In this domain, objects might get broken not by getting dropped, but because a bomb in their proximity explodes: $\Gamma_{\mathsf{Detonate}(b)} = \{\mathsf{Bomb}(b) \wedge \mathsf{Near}(b, x)/\mathsf{Broken}(x)\}$. Def. 11 yields the direct effect formulas $\mathit{DirT}(f, \mathsf{Detonate}(b), s, t) \equiv (\exists x)(f = \mathsf{Broken}(x) \wedge \mathit{Holds}(\mathsf{Near}(x, b), s))$ and $\mathit{DirF}(f, \mathsf{Detonate}(b), s, t) \equiv \bot$.

In this example, the defaults from Definition 9 also prevented conflicts possibly arising from non-local effects. We will later see that this is the case for all domains with local and non-local effect actions.

Like the original framework, our extension implements a particular preference ordering between causes that determine a fluent's truth value. This means that whenever two causes are in conflict – for example, a state default says an object is not broken, and an action effect says it *is* – the preferred cause takes precedence. The preferences are

$$\text{direct effects} < \text{default conclusions} < \text{persistence},$$

where $a < b$ means "a is preferred to b". The theorem below proves that this preference ordering is indeed established.

Theorem 3. *Let Σ be a domain axiomatisation, Δ be a set of state defaults, $\delta = \Phi \rightsquigarrow \psi \in \Delta$ be a state default, E be an extension for the domain axiomatisation with state defaults $(\Sigma \cup \Sigma_\Delta, \Delta_{\mathit{Init}} \cup \Delta_{\mathit{Poss}})$, φ be a ground fluent, and $E \models \mathit{Poss}(\alpha, \sigma, \tau)$ for ground action α and time points σ, τ.*

1. *Effects override everything:*

$$\Phi/(\neg)\varphi \in \Gamma_\alpha \text{ and } E \models \Phi[\sigma] \text{ imply } E \models (\neg)\varphi[\tau].$$

2. *Defaults override persistence:*
   (A) Let $\Phi''/\psi, \Phi''/\neg\psi \notin \Gamma_\alpha$ for all Φ'';
   (B) for each $\delta' = \Phi' \rightsquigarrow \neg\psi \in \Delta$, let δ' be not applicable to E; and
   (C) $E \models \mathit{Pre}_\delta(\sigma, \tau) \wedge \mathit{Safe}_\delta(\sigma, \tau)$.
   Then $E \models \psi[\tau]$.
3. *The frame assumption is correctly implemented:*
   For all fluent formulas Φ'', let $\Phi''/\psi, \Phi''/\neg\psi \notin \Gamma_\alpha$ and for all state defaults δ' with consequent ψ or $\neg\psi$, let $E \models \neg\mathit{Pre}_{\delta'}(\sigma, \tau)$. Then $E \models \psi[\sigma] \equiv \psi[\tau]$.

Proof sketch. Similar to the proof of Theorem 3 in [Baumann *et al.*, 2010], adapted to our definition of Reiter defaults. □

5 Disjunctive Effects

The next and final addition to effect axiom (8) is the step of generalising purely deterministic action effects. Disjunctive action effects have been studied in the past [Kartha, 1994; Shanahan, 1997; Giunchiglia *et al.*, 1997; Thielscher, 2000]. Our contribution in this paper is two-fold. First, we express disjunctive effects by building them into the effect axiom inspired by work on nonmonotonic causal theories [Giunchiglia *et al.*, 2004]. This works without introducing additional function symbols – called *determining fluents* [Shanahan, 1997] – for which persistence is not assumed and that are used to derive indeterminate effects via conditional effects. The second and more important contribution is the combination of non-deterministic effects with state defaults. We claim that it brings a significant representational advantage: Disjunctive effects can explicitly represent potentially different outcomes of an action of which none is necessarily predictable. At the same time, state defaults can be used to model the action effect that *normally* obtains. For example, dropping an object might not always completely break it, but most of the time only damage it. This can be modelled in our framework by specifying "broken or damaged" as disjunctive effect of the drop action, and then including the default "normally, dropped objects are damaged" to express the usual outcome.

Next, we define how disjunctive effects are declared by the user and accommodated into the theory. The basic idea is to allow disjunctions of fluent literals $\psi_1 \vee \ldots \vee \psi_n$ in the effect part of a direct effect expression. The intended meaning of these disjunctions is that after action execution, at least one of the effects ψ_i holds.

Definition 12. Let Φ be a fluent formula and $\Psi = \psi_1 \vee \ldots \vee \psi_n$ be a disjunction of fluent literals. The pair Φ/Ψ is called a *conditional disjunctive effect expression (or cdee)*.

Firstly, we want to guarantee that at least one effect out of $\psi_1 \vee \ldots \vee \psi_n$ occurs. To this end, we say for each ψ_i that non-occurrence of all the other effects ψ_j with $j \neq i$ is a sufficient cause for ψ_i to occur. We build into the effect axiom (in the same way as before) the n implications

$$\Phi[s] \wedge \neg\psi_2[t] \wedge \ldots \wedge \neg\psi_n[t] \supset \mathit{Caused}(\psi_1, a, s, t)$$

$$\vdots$$

$$\Phi[s] \wedge \neg\psi_1[t] \wedge \ldots \wedge \neg\psi_{n-1}[t] \supset \mathit{Caused}(\psi_n, a, s, t)$$

This, together with the persistence assumption, is in effect an exclusive or where only exactly one effect occurs (given that no other effects occur simultaneously). Thus we add, for each literal, its truth as sufficient cause for itself being true:

$$\begin{gathered}\Phi[s] \wedge \psi_1[t] \supset \mathit{Caused}(\psi_1, a, s, t)\\ \vdots \\ \Phi[s] \wedge \psi_n[t] \supset \mathit{Caused}(\psi_n, a, s, t)\end{gathered}$$

This makes every interpretation where at least one of the mentioned literals became true a model of the effect axiom. For the next definition, we identify a disjunction of literals $\Psi = \psi_1 \vee \ldots \vee \psi_n$ with the set of literals $\{\psi_1, \ldots, \psi_n\}$.

Definition 13. Let $\varepsilon = \Phi/\Psi$ be a conditional disjunctive effect expression, $\psi \in \Psi$ and f : FLUENT and s, t : TIME be variables. The following macro expresses that *effect ψ of cdee ε has been activated for f from s to t*:

$$\mathit{Activated}_{\varepsilon,\psi}(f, s, t) \stackrel{\text{def}}{=}$$

$$f = |\psi| \wedge \Phi[s] \wedge \left(\left(\bigwedge_{\psi' \in \Psi \setminus \{\psi\}} \neg\psi'[t]\right) \vee \psi[t]\right)$$

Let A be a function into sort ACTION and Γ_A be a set of conditional disjunctive effect expressions with free variables in $\vec{x}$ that denote the direct conditional disjunctive effects of $A(\vec{x})$. The *direct positive and negative effect formulas* are

$$\mathit{DirT}(f, A(\vec{x}), s, t) \equiv \bigvee_{\substack{\Phi/\Psi \in \Gamma_{A(\vec{x})},\\ \psi \in \Psi,\, \mathit{sign}(\psi) = +}} \mathit{Activated}_{\varepsilon,\psi}(f, s, t) \quad (17)$$

$$\mathit{DirF}(f, A(\vec{x}), s, t) \equiv \bigvee_{\substack{\Phi/\Psi \in \Gamma_{A(\vec{x})},\\ \psi \in \Psi,\, \mathit{sign}(\psi) = -}} \mathit{Activated}_{\varepsilon,\psi}(f, s, t) \quad (18)$$

The implementation of the example sketched above illustrates the definition.

Example 1 (Continued)**.** We once again modify the action $\mathsf{Drop}(x)$. Now a fragile object that is dropped becomes not necessarily completely broken, but might only get damaged. To this end, we record in the new fluent $\mathsf{Dropped}(x)$ that the object has been dropped and write the state default $\delta = \mathsf{Dropped}(x) \rightsquigarrow \mathsf{Damaged}(x)$ saying that dropped objects are usually damaged. Together, these two express the *normal* outcome of the action drop. Formally, the action effects are $\Gamma_{\mathsf{Drop}(x)} = \{\, \top/\neg\mathsf{Carries}(x), \top/\mathsf{Dropped}(x), \mathsf{Fragile}(x)/\mathsf{Broken}(x) \vee \mathsf{Damaged}(x)\}$. Constructing the direct effect formulas as per Definition 13 yields

$$\begin{aligned}&\mathit{DirT}(f, \mathsf{Drop}(x), s, t) \equiv\\ &\quad f = \mathsf{Dropped}(x)\\ &\quad \vee (f = \mathsf{Broken}(x) \wedge \mathit{Holds}(\mathsf{Fragile}(x), s) \wedge\\ &\qquad (\neg\mathit{Holds}(\mathsf{Damaged}(x), t) \vee \mathit{Holds}(\mathsf{Broken}(x), t)))\\ &\quad \vee (f = \mathsf{Damaged}(x) \wedge \mathit{Holds}(\mathsf{Fragile}(x), s) \wedge\\ &\qquad (\neg\mathit{Holds}(\mathsf{Broken}(x), t) \vee \mathit{Holds}(\mathsf{Damaged}(x), t)))\end{aligned}$$

Since the effect axiom of $\mathsf{Drop}(x)$ is itself not determined about the status of $\mathsf{Broken}(x)$ and $\mathsf{Damaged}(x)$ (but is determined about $\mathsf{Damaged}(x)$ not being among its negative effects), the default δ_{Poss} is applicable and we conclude

$$\begin{aligned}&(\Sigma^{\mathit{break}} \cup \Sigma^{\mathit{break}}_{\Delta}, \Delta^{\mathit{break}}_{\mathit{Init}} \cup \Delta^{\mathit{break}}_{\mathit{Poss}}) \mid\!\approx\\ &\mathit{Holds}(\mathsf{Carries}(\mathsf{Vase}), S_0) \wedge \mathit{Holds}(\mathsf{Damaged}(\mathsf{Vase}), S_1)\end{aligned}$$

If we now observe that the vase is broken after all – $\mathit{Holds}(\mathsf{Broken}(\mathsf{Vase}), S_1)$ – and add this information to the knowledge base, we will learn that this was an action effect:

$$\begin{aligned}&(\Sigma^{\mathit{break}} \cup \Sigma^{\mathit{break}}_{\Delta}, \Delta^{\mathit{break}}_{\mathit{Init}} \cup \Delta^{\mathit{break}}_{\mathit{Poss}}) \mid\!\approx\\ &\qquad \mathit{Holds}(\mathsf{Broken}(\mathsf{Vase}), S_1) \supset\\ &\qquad\quad \mathit{DirT}(\mathsf{Broken}(\mathsf{Vase}), \mathsf{Drop}(\mathsf{Vase}), S_0, S_1)\end{aligned}$$

Furthermore, the observation allows us to rightly infer that the vase was fragile at S_0.

It is worth noting that for a cdee Φ/Ψ with deterministic effect $\Psi = \{\psi\}$, the macro $\mathit{Activated}_{\Phi/\Psi,\psi}(f, s, t)$ expressing activation of this effect is equivalent to $\mathit{Activated}_{\Phi/\psi}(f, s, t)$ from Definition 7 for activation of the conditional effect; hence the direct effect formulas (17) for disjunctive effects are a generalisation of (12), the ones for deterministic effects. We have considered here only *local* non-deterministic effects to keep the presentation simple. Of course, the notion can be extended to non-local effects without harm.

6 Properties of the Extended Framework

We have already seen in previous sections that the approach to default reasoning about actions presented here has certain nice properties: it is a generalisation of the basic approach [Baumann *et al.*, 2010] and it implements a particular preference ordering among causes. While those results were mostly straightforward adaptations, the theorem below is novel. It states that conflicts between conditional effects and default conclusions in the sense of Definition 8 cannot occur.

Theorem 4. *Let (Σ, Δ) be a domain axiomatisation with defaults, E be an extension for (Σ, Δ) and $\delta = \Phi \rightsquigarrow \psi$ be a state default. Furthermore, let $i \geq 0$ be such that $\mathit{Def}(\psi, \sigma, \tau) \notin E_i$ and $\mathit{Def}(\psi, \sigma, \tau) \in E_{i+1}$. Then for all ground actions α, $\mathit{Poss}(\alpha, \sigma, \tau) \supset \neg\mathit{Dir}(\neg\psi, \alpha, \sigma, \tau) \in E_i$.*

Proof. According to Def. 2, we have $E_{i+1} = \mathit{Th}(E_i) \cup \Delta_i$; hence, $\mathit{Def}(\psi, \sigma, \tau) \in E_{i+1}$ can have two possible reasons:

1. $\mathit{Def}(\psi, \sigma, \tau) \in \mathit{Th}(E_i) \setminus E_i$. By construction, this can only be due to effect axiom (8), more specifically, we have (1) $E_i \models \mathit{Caused}(\psi, \alpha, \sigma, \tau) \wedge \neg\mathit{Frame}(\psi, \sigma, \tau) \wedge \neg\mathit{Dir}(\psi, \sigma, \tau)$ and (2) $E_i \models \neg\mathit{Caused}(\neg\psi, \alpha, \sigma, \tau)$, whence $E_i \models \neg\mathit{Dir}(\neg\psi, \alpha, \sigma, \tau)$ proving the claim.
2. $\mathit{Def}(\psi, \sigma, \tau) \in \Delta_i$. By definition of δ_{Poss} in Def. 9, $\mathit{Pre}_\delta(\sigma, \tau) \wedge \mathit{Safe}_\delta(\sigma, \tau) \in E_i$, whereby we can conclude $\mathit{Poss}(\alpha, \sigma, \tau) \supset \neg\mathit{Dir}(\neg\psi, \alpha, \sigma, \tau) \in E_i$. □

Note that conflicts already arise with conditional, local effects; the framework however makes sure there are no conflicts even for conditional, non-local, disjunctive effects.

Finally, the existence of extensions for domain axiomatisations with state defaults can still be guaranteed for the extended framework.

Theorem 5. *Let Σ be a domain axiomatisation and Δ be a set of state defaults. Then the corresponding domain axiomatisation with defaults $(\Sigma \cup \Sigma_\Delta, \Delta_{Init} \cup \Delta_{Poss})$ has an extension. If furthermore Σ is consistent, then so are all extensions for $(\Sigma \cup \Sigma_\Delta, \Delta_{Init} \cup \Delta_{Poss})$.*

Proof. Existence of an extension is a corollary of Theorem 3.1 in [Reiter, 1980] since the defaults in $\Delta_{Init} \cup \Delta_{Poss}$ are still normal. If Σ is consistent, then so is $\Sigma \cup \Sigma_\Delta$ by the argument in the proof of Theorem 4 in [Baumann *et al.*, 2010]. Consistency of all extensions then follows from Corollary 2.2 in [Reiter, 1980]. □

Additionally, it is easy to see that the domain specifications provided by the user are still modular: different parts of the specifications, such as conditional effect expressions and state defaults, are completely independent of each other from a user's point of view. Yet, the intricate semantic interactions between them are correctly dealt with.

7 Discussion

We have presented an extension to a recently introduced framework for default reasoning in theories of actions and change. The extension increases the range of applicability of the framework while fully retaining its desirable properties: we can now express context-dependent effects of actions, actions with a potentially global effect range and indeterminate effects of actions – all the while domain descriptions have not become significantly more complex, and default extensions of the framework still provably exist.

There is not much related work concerning the kind of default reasoning about actions we consider here. [Denecker and Ternovska, 2007] enriched the situation calculus [Reiter, 2001] with inductive definitions. While they provide a nonmonotonic extension of an action calculus, the intended usage is to solve the ramification problem rather than to do the kind of defeasible reasoning we are interested in this work. [Lakemeyer and Levesque, 2009] provide a progression-based semantics for state defaults in a variant of the situation calculus, but without looking at nondeterministic actions. In an earlier paper [Strass and Thielscher, 2009], we explored default effects of nondeterministic actions, albeit in a much more restricted setting: there, actions had only unconditional effects – either deterministic or disjunctive of the form $f \vee \neg f$ –, and defaults had only atomic components, that is, they were of the form $(\neg)Holds(f, t) : (\neg)Holds(g, t)/(\neg)Holds(g, t)$. Most recently, [Michael and Kakas, 2011] gave an argumentation-based semantics for propositional action theories with state defaults. While being more flexible in terms of preferences between causes, their approach is constricted to a linear time structure built into the language and does not make a clear ontological distinction between fluents and actions.

References

[Baumann *et al.*, 2010] Ringo Baumann, Gerhard Brewka, Hannes Strass, Michael Thielscher, and Vadim Zaslawski. State Defaults and Ramifications in the Unifying Action Calculus. In *Proceedings of KR*, pages 435–444, Toronto, Canada, May 2010.

[Denecker and Ternovska, 2007] Marc Denecker and Eugenia Ternovska. Inductive Situation Calculus. *AIJ*, 171(5–6):332–360, 2007.

[Giunchiglia *et al.*, 1997] Enrico Giunchiglia, G. Neelakantan Kartha, and Vladimir Lifschitz. Representing Action: Indeterminacy and Ramifications. *AIJ*, 95(2):409–438, 1997.

[Giunchiglia *et al.*, 2004] Enrico Giunchiglia, Joohyung Lee, Vladimir Lifschitz, Norman McCain, and Hudson Turner. Nonmonotonic Causal Theories. *AIJ*, 153(1-2):49–104, 2004.

[Kartha, 1994] G. Neelakantan Kartha. Two Counterexamples Related to Baker's Approach to the Frame Problem. *AIJ*, 69(1–2):379–391, 1994.

[Lakemeyer and Levesque, 2009] Gerhard Lakemeyer and Hector Levesque. A Semantical Account of Progression in the Presence of Defaults. In *Proceedings of IJCAI*, pages 842–847, 2009.

[McCarthy and Hayes, 1969] John McCarthy and Patrick J. Hayes. Some Philosophical Problems from the Standpoint of Artificial Intelligence. In *Machine Intelligence*, pages 463–502. Edinburgh University Press, 1969.

[Michael and Kakas, 2011] Loizos Michael and Antonis Kakas. A Unified Argumentation-Based Framework for Knowledge Qualification. In E. Davis, P. Doherty, and E. Erdem, editors, *Proceedings of the Tenth International Symposium on Logical Formalizations of Commonsense Reasoning*, Stanford, CA, March 2011.

[Reiter, 1980] Raymond Reiter. A Logic for Default Reasoning. *AIJ*, 13:81–132, 1980.

[Reiter, 1991] Raymond Reiter. The Frame Problem in the Situation Calculus: A Simple Solution (Sometimes) and a Completeness Result for Goal Regression. In *Artificial Intelligence and Mathematical Theory of Computation – Papers in Honor of John McCarthy*, pages 359–380. Academic Press, 1991.

[Reiter, 2001] Raymond Reiter. *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. The MIT Press, September 2001.

[Shanahan, 1997] Murray Shanahan. *Solving the Frame Problem: A Mathematical Investigation of the Common Sense Law of Inertia*. The MIT Press, February 1997.

[Strass and Thielscher, 2009] Hannes Strass and Michael Thielscher. Simple Default Reasoning in Theories of Action. In *Proceedings of AI*, pages 31–40, Melbourne, Australia, December 2009. Springer-Verlag Berlin Heidelberg.

[Thielscher, 2000] Michael Thielscher. Nondeterministic Actions in the Fluent Calculus: Disjunctive State Update Axioms. In *Intellectics and Computational Logic (to Wolfgang Bibel on the occasion of his 60th birthday)*, pages 327–345, Deventer, The Netherlands, The Netherlands, 2000. Kluwer, B.V.

[Thielscher, 2011] Michael Thielscher. A Unifying Action Calculus. *AIJ*, 175(1):120–141, 2011.

A Logic for Specifying Partially Observable Stochastic Domains

Gavin Rens^{1,2} and **Thomas Meyer**^{1,2} and **Alexander Ferrein**^{3} and **Gerhard Lakemeyer**^{3}
{grens,tmeyer}@meraka.org.za {ferrein,gerhard}@cs.rwth-aachen.de

^{1}CSIR Meraka Institute, Pretoria, South Africa
^{2}University of KwaZulu-Natal, School of Computer Science, South Africa
^{3}RWTH Aachen University, Informatik, Germany

Abstract

We propose a novel modal logic for specifying agent domains where the agent's actuators and sensors are noisy, causing uncertainty in action and perception. The logic draws both on POMDP theory and logics of action and change. The development of the logic builds on previous work in which a simple multi-modal logic was augmented with first-class *observation* objects. These *observations* can then be used to represent the set of observations in a POMDP model in a natural way. In this paper, a subset of the simple modal logic is taken for the new logic, in which modal operators may not be nested. The modal operators are then extended with notions of probability. It will be shown how stochastic domains can be specified, including new kinds of axioms dealing with perception and a *frame solution* for the proposed logic.

1 Introduction and Motivation

In the physical real world, or in extremely complex engineered systems, things are not black-and-white. We live in a world where there can be shades of truth and degrees of belief. Part of the problem is that agents' actuators and sensors are noisy, causing uncertainty in their action and perception. In this paper, we propose a novel logic that draws on partially observable Markov decision process (POMDP) theory and on logics for reasoning about action and change, combining both in a coherent language to model change and uncertainty.

Imagine a robot that is in need of an oil refill. There is an open can of oil on the floor within reach of its gripper. If there is nothing else in the robot's gripper, it can grab the can (or miss it, or knock it over) and it can drink the oil by lifting the can to its 'mouth' and pouring the contents in (or miss its mouth and spill). The robot may also want to confirm whether there is anything left in the oil-can by weighing its contents. And once holding the can, the robot may wish to place it back on the floor. In situations where the oil-can is full, the robot gets 5 units of reward for grabbing the can, and it gets 10 units for a *drink* action. Otherwise, the robot gets no rewards. Rewards motivate an agent to behave as desired.

The domain is (partially) formalized as follows. The robot has the set of actions $\mathfrak{A} = \{grab, drink, weigh, replace\}$ with expected meanings. The robot can perceive observations only from the set $\Omega = \{\mathit{obsNil}, \mathit{obsLight}, \mathit{obsMedium}, \mathit{obsHeavy}\}$. When the robot performs a $weigh$ action (i.e., it activates its 'weight' sensor) it will perceive either $\mathit{obsLight}$, $\mathit{obsMedium}$ or $\mathit{obsHeavy}$; for other actions, it will perceive obsNil. The robot experiences its environs through three Boolean features: $\mathfrak{P} = \{\mathit{full}, drank, holding\}$ meaning respectively that the oil-can is full, that the robot has drunk the oil and that it is currently holding something in its gripper.

Given a formalization $\mathcal{K}$ of our scenario, the robot may have the following queries:

- Is the probability of perceiving that the oil-can is light 0.7 when the can is not full, and have I drunk the oil, and am I holding the can? Does $(\mathit{obsLight} \mid weigh)_{0.7}(\neg \mathit{full} \land drank \land holding)$ follow from $\mathcal{K}$?
- If the oil-can is empty and I'm not holding it, is there a 0.9 probability that I'll be holding it after grabbing it, and a 0.1 probability that I'll have missed it? Does $(\neg \mathit{full} \land \neg holding) \rightarrow ([grab]_{0.9}(\neg \mathit{full} \land holding) \land [grab]_{0.1}(\neg \mathit{full} \land \neg holding))$ follow from $\mathcal{K}$?

In order for robots and intelligent agents in stochastic domains to reason about actions and observations, they must first have a model of the domain over which to reason. For example, a robot may need to represent available knowledge about its *grab* action in its current situation. It may need to represent that when 'grabbing' the oil-can, there is a 5% chance that it will knock over the oil-can. As another example, if the robot has access to information about the weight of an oil-can, it may want to represent the fact that the can weighs heavy with a 90% chance in 'situation A', but that it is heavy with a 98% chance in 'situation B'.

Logic-based artificial intelligence for agent reasoning is well established. In particular, a domain expert choosing to represent domains with a *logic* can take advantage of the progress made in cognitive robotics [Levesque and Lakemeyer, 2008] to specify domains in a compact and transparent manner. Modal logic is considered to be well suited to reasoning about beliefs and changing situations.

POMDP theory has proven to be a good general framework for formalizing *dynamic, stochastic* systems. A drawback of traditional POMDP models is that they cannot include information about general facts and laws. Moreover, succinct axioms describing the dynamics of a domain cannot be written in POMDP theory. In this work, we develop a logic that will further our goal of combining modal logic with POMDP theory. That is, here we design a modal logic that can represent POMDP problems specifically for reasoning tasks in cognitive robotics (with domain axioms). The logic for actual decision-making will be developed in later work. To facilitate the correspondence between POMDPs and an agent logic, we require *observation objects* in the logic to correspond to the POMDPs' set of observations. Before the introduction of the Logic of Actions and Observations (LAO) [Rens *et al.*, 2010], no modal logic had explicit observations as first-class elements; sensing was only dealt with via special actions or by treating actions in such a way that they somehow get hold of observations. LAO is also able to accommodate models of nondeterminism in the actions and models of uncertainty in the observations. But in LAO, these notions are non-probabilistic.

In this paper we present the Specification Logic of Actions and Observations with Probability (SLAOP). SLAOP is derived from LAO and thus also considers observations as first-class objects, however, a probabilistic component is added to LAO for expressing uncertainty more finely. We have invented a new knowledge representation framework for our observation objects, based on the established approaches for specifying the behavior of actions.

We continue our motivation with a look at the related work, in Section 2. Section 3 presents the logic and Section 4 provides some of the properties that can be deduced. Section 5 illustrates domain specification with SLAOP, including a solution to the frame problem. Section 6 concludes the paper.

2 Related Work

Although SLAOP uses probability theory, it is not for reasoning *about* probability; it is for reasoning about (probabilistic) actions and observations. There have been many frameworks for reasoning *about* probability, but most of them are either not concerned with *dynamic environments* [Fagin and Halpern, 1994; Halpern, 2003; Shirazi and Amir, 2007] or they *are* concerned with change, but they are not actually *logics* [Boutilier *et al.*, 2000; Bonet and Geffner, 2001]. Some probabilistic logics for reasoning about action and change do exist [Bacchus *et al.*, 1999; Iocchi *et al.*, 2009], but they lack some desirable attributes, for example, a solution to the frame problem, nondeterministic actions, or catering for sensing. There are some logics that come closer to what we desire [Weerdt *et al.*, 1999; Van Diggelen, 2002; Gabaldon and Lakemeyer, 2007; Van Benthem *et al.*, 2009], that is, they are modal and they incorporate notions of probability, but they were not created with POMDPs in mind and they don't take observations as first-class objects. One non-logical formalism for representing POMDPs [Boutilier and Poole, 1996] exploits structure in the problems for more compact representations. In (logic-based) cognitive robotics, such compact representation is the norm, for example, specifying only local effects of actions, and specifying a value related to a *set* of states in only one statement.

On the other hand, there are three formalisms for specifying POMDPs that employ *logic-based* representation [Wang and Schmolze, 2005; Sanner and Kersting, 2010; Poole, 1998]. But for two of these, the frameworks are not *logics* per se. The first [Wang and Schmolze, 2005] is based on *Functional STRIPS*, "which is a simplified first-order language that involves constants, functions, and predicate symbols but does not involve variables and quantification". Their representations of POMDPs are relatively succinct and they have the advantage of using first-order predicates. The STRIPS-like formalism is geared specifically towards planning, though, and their work does not mention reasoning about general facts. Moreover, in their approach, action-nondeterminism is modeled by associating sets of deterministic action-outcomes per nondeterministic action, whereas SLAOP will model nondeterminism via action *effects*—arguably, ours is a more natural and succinct method. Sanner and Kersting [2010] is similar to the first formalism, but instead of Functional STRIPS, they use the situation calculus to model POMDPs. Although reified situations make the meaning of formulae perspicuous, and reasoning with the situation calculus, in general, has been accepted by the community, when actions are nondeterministic, 'action histories' cause difficulties in our work: The set of possible alternative histories is unbounded and some histories may refer to the same state [Rens, 2010, Chap. 6]. When, in future work, SLAOP is extended to express belief states (i.e., sets of possible alternative states), dealing with duplicate states will be undesirable.

The Independent Choice Logic [Poole, 1998] is relatively different from SLAOP; it is an extension of Probabilistic Horn Abduction. Due to its difference, it is hard to compare to SLAOP, but it deserves mentioning because it shares its application area with SLAOP and both are inspired by decision theory. The future may tell which logic is better for certain representations and for reasoning over the representations.

Finally, SLAOP was not conceived as a new approach to represent POMDPs, but as the underlying specification language in a larger *meta*-language for reasoning robots that include notions of probabilistic uncertainty. The choice of POMDPs as a semantic framework is secondary.

3 Specification Logic of Actions and Observations with Probability

SLAOP is a non-standard modal logic for POMDP specification for robot or intelligent agent design. The specification of robot movement has a 'single-step' approach in SLAOP. As such, the syntax will disallow nesting of modal operators; sentences with *sequences* of actions, like $[grab][drink][replace]drank$ are not allowed. Sentences will involve at most unit actions, like $[grab]holding \vee [drink]drank$. Nevertheless, the 'single-step' approach is sufficient for *specifying* the probabilities of transitions due to action executions. The logic to be defined in a subsequent paper will allow an agent to query the probability of some propositional formula φ after an *arbitrary* sequence of actions and observations.

3.1 Syntax

The vocabulary of our language contains four sorts:

1. a finite set of *fluents* (alias *propositional atoms*) $\mathfrak{P} = \{p_1, \ldots, p_n\}$,
2. a finite set of names of atomic *actions* $\mathfrak{A} = \{\alpha_1, \ldots, \alpha_n\}$,
3. a finite set of names of atomic *observations* $\Omega = \{\varsigma_1, \ldots, \varsigma_n\}$,
4. a countable set of names $\mathfrak{Q} = \{q_1, q_2, \ldots\}$ of *rational numbers* in $\mathbb{Q}$.

From now on, denote $\mathbb{Q} \cap (0, 1]$ as $\mathbb{Q}^{\cap}$. We refer to elements of $\mathfrak{A} \cup \Omega \cup \mathfrak{Q}$ as *constants*. We are going to work in a multi-modal setting, in which we have modal operators $[\alpha]_q$, one for each $\alpha \in \mathfrak{A}$, and predicates $(\varsigma \mid \alpha)_q$ and $(\varsigma \mid \alpha)^{\diamond}$, for each pair in $\Omega \times \mathfrak{A}$.

Definition 3.1 *Let $\alpha, \alpha' \in \mathfrak{A}$, $\varsigma, \varsigma' \in \Omega$, $q \in (\mathfrak{Q} \cap (0, 1])$, $r, c \in \mathfrak{Q}$ and $p \in \mathfrak{P}$. The language of SLAOP, denoted $\mathcal{L}_{SLAOP}$, is the least set of Φ defined by the grammars:*

$$\begin{aligned} \varphi &::= p \mid \top \mid \neg\varphi \mid \varphi \wedge \varphi. \\ \Phi &::= \ \varphi \mid [\alpha]_q\varphi \mid (\varsigma \mid \alpha)_q \mid (\varsigma \mid \alpha)^{\diamond} \mid \alpha = \alpha' \mid \\ &\quad\ \varsigma = \varsigma' \mid Reward(r) \mid Cost(\alpha, c) \mid \neg\Phi \mid \Phi \wedge \Phi. \end{aligned}$$

As usual, we treat $\bot, \vee, \rightarrow$ and $\leftrightarrow$ as abbreviations.

We shall refer to formulae $\varphi ::= p \mid \top \mid \neg\varphi \mid \varphi \wedge \varphi$ as *static*. If a formula is static, it mentions no actions and no observations.

$[\alpha]_q\varphi$ is read 'The probability of reaching a world in which φ holds after executing α, is equal to q'. $[\alpha]$ abbreviates $[\alpha]_1$. $\langle\alpha\rangle\varphi$ abbreviates $\neg[\alpha]\neg\varphi$. $(\varsigma \mid \alpha)_q$ can be read 'The probability of perceiving ς is equal to q, given α was performed'. $(\varsigma \mid \alpha)^{\Box}$ abbreviates $(\varsigma \mid \alpha)_1$. $(\varsigma \mid \alpha)^{\diamond}$ is read 'It is possible to perceive ς', given α was performed'.

The definition of a POMDP reward function $\mathcal{R}(a, s)$ may include not only the expected rewards for being in the states reachable from s via a, but it may deduct the cost of performing a in s. To specify rewards and execution costs in SLAOP, we require $Reward$ and $Cost$ as special predicates. $Reward(r)$ can be read 'The reward for being in the current situation is r units,' and we read $Cost(\alpha, c)$ as 'The cost for executing α is c units.'

Let $V_{\mathfrak{A}} = \{v_1^{\alpha}, v_2^{\alpha}, \ldots\}$ be a countable set of *action variables* and $V_{\Omega} = \{v_1^{\varsigma}, v_2^{\varsigma}, \ldots\}$ a countable set of *observation variables*. Let $\varphi|_{\alpha_1}^{v^{\alpha}} \wedge \ldots \wedge \varphi|_{\alpha_n}^{v^{\alpha}}$ be abbreviated by $(\forall v^{\alpha})\varphi$, where $\varphi|_c^v$ means φ with all variables $v \in (V_{\mathfrak{A}} \cup V_{\Omega})$ appearing in it replaced by constant c of the right sort (action or observation). Quantification over observations is similar to that for actions; the symbol $\exists$ is also available for abbreviation, with the usual meaning.

3.2 Semantics

While presenting our semantics, we show how a POMDP, as defined below, can be represented by a SLAOP structure.

A POMDP [Kaelbling *et al.*, 1998] (for our purposes) is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \Omega, \mathcal{O} \rangle$, where $\mathcal{S}$ is a finite set of states that the agent can be in; $\mathcal{A}$ is a finite set of agent actions; $\mathcal{T}$ is the *state-transition function*, representing, for each action, transition probabilities between states; $\mathcal{R}$ is the *reward function*, giving the expected immediate reward gained by the agent, for any state and agent action; Ω is a finite set of observations the agent can experience of its environment; and $\mathcal{O}$ is the *observation function*, giving, for each action and the resulting state, a probability distribution over observations, representing the agent's 'trust' in its observations.

Our semantics follows that of multi-modal logic **K**. However, SLAOP structures are non-standard in that they are extensions of structures with the form $\langle W, R \rangle$, where W is a *finite* set of worlds such that each world assigns a truth value to each atomic proposition, and R is a binary relation on W.

Intuitively, when talking about some world w, we mean a set of features (*fluents*) that the agent understands and that describes a state of affairs in the world or that describes a possible, alternative world. Let $w : \mathfrak{P} \mapsto \{0, 1\}$ be a total function that assigns a truth value to each fluent. Let C be the set of all possible functions w. We call C the *conceivable worlds*.

Definition 3.2 *A SLAOP structure is a tuple $\mathfrak{S} = \langle W, R, O, N, Q, U \rangle$ such that*

1. *$W \subseteq C$: the set of* possible worlds *(corresponding to $\mathcal{S}$);*
2. *R: a mapping that provides an accessibility relation $R_{\alpha} : W \times W \times \mathbb{Q}^{\cap}$ for each $\alpha \in \mathfrak{A}$ (corresponding to $\mathcal{T}$); Given some $w^- \in W$, we require that $\sum_{(w^-, w^+, pr) \in R_{\alpha}} pr = 1$; If (w^-, w^+, pr), $(w^-, w^+, pr') \in R_{\alpha}$, then $pr = pr'$;*
3. *O: a nonempty finite set of observations (corresponding to Ω);*
4. *$N : \Omega \mapsto O$ is a bijection that associates to each name in Ω, a unique observation in O;*
5. *Q: a mapping that provides a perceivability relation $Q_{\alpha} : O \times W \times \mathbb{Q}^{\cap}$ for each $\alpha \in \mathfrak{A}$ (corresponding to $\mathcal{O}$); Given some $w^+ \in W$, we require that $\sum_{(o, w^+, pr) \in Q_{\alpha}} pr = 1$; If (ς, w^+, pr), $(\varsigma, w^+, pr') \in Q_{\alpha}$, then $pr = pr'$;*
6. *U: a pair $\langle Re, Co \rangle$ (corresponding to $\mathcal{R}$), where $Re : W \mapsto \mathbb{Q}$ is a reward function and Co is a mapping that provides a cost function $Co_{\alpha} : W \mapsto \mathbb{Q}$ for each $\alpha \in \mathfrak{A}$;*
7. *Observation-per-action condition: For all $\alpha \in \mathfrak{A}$, if $(w, w', pr^{\alpha}) \in R_{\alpha}$, then there is an $o \in O$ s.t. $(o, w', pr^{o}) \in Q_{\alpha}$;*
8. *Nothing-for-nothing condition: For all w, if there exists no w' s.t. $(w, w', pr) \in R_{\alpha}$ for some pr, then $Co_{\alpha}(w) = 0$.*

$\mathcal{A}$ corresponds to $\mathfrak{A}$ and Ω to Ω. R_{α} defines which worlds w^+ are accessible via action α performed in world w^- and the transition probability $pr \in \mathbb{Q}^{\cap}$. Q_{α} defines which observations o are perceivable in worlds w^+ accessible via action α and the observation probability $pr \in \mathbb{Q}^{\cap}$. We prefer to exclude relation elements referring to transitions that cannot occur, hence why $pr \in \mathbb{Q}^{\cap}$ and not $pr \in \mathbb{Q} \cap [0, 1]$.

Because N is a bijection, it follows that $|O| = |\Omega|$ (we take $|X|$ to be the cardinality of set X). The value of the reward function $Re(w)$ is a rational number representing the reward an agent gets for being in or getting to the world w. It must be defined for each $w \in W$. The value of the cost function $Co(\alpha, w^-)$ is a rational number representing the cost of executing α in the world w^-. It must be defined for each action $\alpha \in \mathfrak{A}$ and each $w^- \in W$. Item 7 of Definition 3.2 implies that actions and observations always appear in pairs, even if implicitly. And item 8 seems reasonable; it states that any action that is inexecutable in world w incurs no cost for it in the world w.

Definition 3.3 (Truth Conditions) *Let $\mathfrak{S}$ be a SLAOP structure, with $\alpha, \alpha' \in \mathfrak{A}$, $\varsigma, \varsigma' \in \Omega$, $q \in (\mathfrak{Q} \cap (0,1])$ or $\mathbb{Q}^{\cap}$ as applicable, and $r \in \mathfrak{Q}$ or $\mathbb{Q}$ as applicable. Let $p \in \mathfrak{P}$ and let φ be any sentence in $\mathcal{L}_{SLAOP}$. We say φ is* satisfied *at world w in structure $\mathfrak{S}$ (written $\mathfrak{S}, w \models \varphi$) if and only if the following holds:*

1. $\mathfrak{S}, w \models p$ iff $w(p) = 1$ for $w \in W$;
2. $\mathfrak{S}, w \models \top$ for all $w \in W$;
3. $\mathfrak{S}, w \models \neg\varphi$ iff $\mathfrak{S}, w \not\models \varphi$;
4. $\mathfrak{S}, w \models \varphi \wedge \varphi'$ iff $\mathfrak{S}, w \models \varphi$ and $\mathfrak{S}, w \models \varphi'$;
5. $\mathfrak{S}, w \models \alpha = \alpha'$ iff α and α' are identical;
6. $\mathfrak{S}, w \models \varsigma = \varsigma'$ iff ς and ς' are identical;
7. $\mathfrak{S}, w \models [\alpha]_q\varphi$ iff $\big(\sum_{(w,w',pr) \in R_\alpha, \mathcal{S}, w' \models \varphi} pr\big) = q$;
8. $\mathfrak{S}, w \models (\varsigma \mid \alpha)_q$ iff $(N(\varsigma), w, q) \in Q_\alpha$;
9. $\mathfrak{S}, w \models (\varsigma \mid \alpha)^{\diamond}$ iff a q exists s.t. $(N(\varsigma), w, q) \in Q_\alpha$;
10. $\mathfrak{S}, w \models Reward(r)$ iff $Re(w) = r$;
11. $\mathfrak{S}, w \models Cost(\alpha, c)$ iff $Co_\alpha(w) = c$.

The definition of item 7 comes from probability theory, which says that the probability of an event (φ) is simply the sum of the probabilities of the atomic events (worlds) where the event (φ) holds.

A formula φ is valid in a SLAOP structure (denoted $\mathfrak{S} \models \varphi$) if $\mathfrak{S}, w \models \varphi$ for every $w \in W$. We define *global logical entailment* (denoted $\mathcal{K} \models_{GS} \varphi$) as follows: for all $\mathfrak{S}$, if $\mathfrak{S} \models \bigwedge_{\psi \in \mathcal{K}} \psi$ then $\mathfrak{S} \models \varphi$.

4 Some Properties

Remark 4.1 *Item 7 of Definition 3.2, the observation-per-action condition, implies that if $\mathfrak{S}, w \models \langle\alpha\rangle\varphi$ then $\mathfrak{S}, w' \models \varphi \rightarrow (\exists v^{\varsigma})(v^{\varsigma} \mid \alpha)^{\diamond}$, for some $w, w' \in W$.*

Remark 4.2 *Item 8 of Definition 3.2, the nothing-for-nothing condition, implies that $\models_{SLAOP}$ $(\forall v^{\alpha})$ $\neg\langle v^{\alpha}\rangle\top$ $\rightarrow$ $Cost(v^{\alpha}, 0)$.*

In the terminology of probability theory, a single world would be called an *atomic event*. Probability theory says that the probability of an event e is simply the sum of the probabilities of the atomic events (worlds) where e holds. We are interested in noting the interactions of any two sentences of the form $[\alpha]_q\varphi$ being satisfied in the same world. Given the principle of the sum of atomic events, we get the following properties.

Proposition 4.1 *Assume an arbitrary structure $\mathfrak{S}$ and some w in $\mathfrak{S}$. Assume $\mathfrak{S}, w \models [\alpha]_q\theta \wedge [\alpha]_{q'}\psi$. Then*

1. *if $q = q'$ then no deduction can be made;*
2. *if $q \neq q'$ then $\mathfrak{S}, w \models \langle\alpha\rangle\neg(\theta \leftrightarrow \psi)$;*
3. *if $q > q'$ then $\mathfrak{S}, w \models \langle\alpha\rangle\neg(\theta \rightarrow \psi)$;*
4. *if $q + q' > 1$ then $\mathfrak{S}, w \models \langle\alpha\rangle(\theta \leftrightarrow \psi)$;*
5. *$\mathfrak{S}, w \models [\alpha]\neg(\theta \wedge \psi) \rightarrow [\alpha]_{q+q'}(\theta \vee \psi)$;*
6. *if $q = 1$ then $\mathfrak{S}, w \models [\alpha](\psi \rightarrow \theta)$ and $\mathfrak{S}, w \models [\alpha]_{q'}(\theta \wedge \psi)$;*
7. *$\mathfrak{S}, w \models [\alpha]_q\top$ is a contradiction if $q < 1$;*
8. *$\mathfrak{S}, w \models [\alpha]_{1-q}\neg\varphi$ iff $\mathfrak{S}, w \models [\alpha]_q\varphi$ and $q \neq 1$;*
9. *$\mathfrak{S}, w \models \neg[\alpha]_{1-q}\neg\varphi$ iff $\mathfrak{S}, w \models \neg[\alpha]_q\varphi$ and $q \neq 1$.*

Proof:
Please refer to our draft report [Rens and Meyer, 2011]. Q.E.D.

It is worth noting that in the case when $q > q'$ (item 3), $\mathfrak{S}, w \models \langle\alpha\rangle\neg(\theta \wedge \psi)$ is also a consequence. But $\langle\alpha\rangle\neg(\theta \rightarrow \psi)$ logically implies $\langle\alpha\rangle\neg(\theta \wedge \psi)$.

Consider item 8 further: Suppose $[\alpha]_{q^*}\varphi$ where $q^* = 1$ (in some structure at some world). Then, in SLAOP, one could represent $\mathfrak{S}, w \models [\alpha]_{1-q^*}\neg\varphi$ as $\neg\langle\alpha\rangle\neg\varphi$. But this is just $[\alpha]\varphi$ ($\equiv [\alpha]_{q^*}\varphi$). The point is that there is no different way to represent $[\alpha]\varphi$ in SLAOP (other than syntactically). Hence, in item 8, we need not cater for the case when $q = 1$.

Proposition 4.2 $\models_{SLAOP}$ $([\alpha]_q\theta \wedge \neg[\alpha]_q\psi) \rightarrow \neg[\alpha](\theta \leftrightarrow \psi)$.

Proof:
Let $\mathfrak{S}$ be any structure and w a world in $\mathfrak{S}$. Assume $\mathfrak{S}, w \models [\alpha]_q\theta \wedge \neg[\alpha]_q\psi$. Assume $\mathfrak{S}, w \models [\alpha](\theta \leftrightarrow \psi)$. Then because $\mathfrak{S}, w \models [\alpha]_q\theta$, one can deduce $\mathfrak{S}, w \models [\alpha]_q\psi$. This is a contradiction, therefore $\mathfrak{S}, w \not\models [\alpha](\theta \leftrightarrow \psi)$. Hence, $\mathfrak{S}, w \models ([\alpha]_q\theta \wedge \neg[\alpha]_q\psi) \rightarrow \neg[\alpha](\theta \leftrightarrow \psi)$. Q.E.D.

Proposition 4.3 *Assume an arbitrary structure $\mathfrak{S}$ and an arbitrary world w in $\mathfrak{S}$. There exists some constant q such that $\mathfrak{S}, w \models [\alpha]_q\varphi$ if and only if $\mathfrak{S}, w \models \langle\alpha\rangle\varphi$.*

Proof:
Assume an arbitrary structure $\mathfrak{S}$ and an arbitrary world w in it. Then
$\mathfrak{S}, w \models [\alpha]_q\varphi$ for some constant q
$\Leftrightarrow \exists q \,.\, \big(\sum_{(w,w',pr) \in R_\alpha, \mathfrak{S}, w' \models \varphi} pr\big) = q$
$\Leftrightarrow$ Not: $\exists q \,.\, \big(\sum_{(w,w',pr) \in R_\alpha, \mathfrak{S}, w' \models \varphi} pr\big) = 0$
$\Leftrightarrow$ Not: $\exists q \,.\, \big(\sum_{(w,w',pr) \in R_\alpha, \mathfrak{S}, w' \models \neg\varphi} pr\big) = 1$
$\Leftrightarrow$ Not: $\mathfrak{S}, w \models [\alpha]\neg\varphi$
$\Leftrightarrow \mathfrak{S}, w \models \langle\alpha\rangle\varphi$. Q.E.D.

We are also interested in noting the interactions of any two percept events—when sentences of the form $(\varsigma \mid \alpha)_q\varphi$ are satisfied in the same world. Only two consequences could be gleaned, given Definition 3.3, item 8:

Proposition 4.4 *Assume an arbitrary structure $\mathfrak{S}$ and some w in $\mathfrak{S}$.*

1. *If $\mathfrak{S}, w \models (\varsigma \mid \alpha)_q \wedge (\varsigma' \mid \alpha)_{q'}$ and ς is the same observation as ς', then $q = q'$;*

2. *If $\mathfrak{S}, w \models (\varsigma \mid \alpha)_q \wedge (\varsigma' \mid \alpha)_{q'}$ and ς is not the same observation as ς', then $q + q' \leq 1$.*

Proof:
Directly from probability theory and algebra. Q.E.D.

Proposition 4.5 *Assume an arbitrary structure $\mathfrak{S}$ and an arbitrary world w in it. There exists some constant q such that $\mathfrak{S}, w \models (\varsigma \mid \alpha)_q$ if and only if $\mathfrak{S}, w \models (\varsigma \mid \alpha)^\diamond$.*

Proof:
Let $N(\varsigma) = o$. Assume an arbitrary structure $\mathfrak{S}$ and an arbitrary world w in $\mathfrak{S}$. Then
$\mathfrak{S}, w \models (\varsigma \mid \alpha)_q$ for some constant q
$\Leftrightarrow \exists q \,.\, (o, w, q) \in Q_\alpha$
$\Leftrightarrow \mathfrak{S}, w \models (\varsigma \mid \alpha)^\diamond$. Q.E.D.

The following is a direct consequence of Propositions 4.3 and 4.5.

Corollary 4.1 *$\models_{SLAOP} [\alpha]_q\varphi \rightarrow \langle\alpha\rangle\varphi$ and $\models_{SLAOP} (\varsigma \mid \alpha)_q \rightarrow (\varsigma \mid \alpha)^\diamond$.*

Further Properties of Interest
Recall that $R^-_\alpha = \{(w, w') \mid (w, w', pr) \in R_\alpha\}$. We now justify treating $[\alpha]_1$ as $[\alpha]$ of regular multi-modal logic.

Proposition 4.6 *$[\alpha]_1$ is the regular $[\alpha]$. That is, $\mathfrak{S}, w \models [\alpha]_1\varphi$ if and only if for all w', if $wR^-_\alpha w'$, then $\mathfrak{S}, w' \models \varphi$, for any structure $\mathfrak{S}$ and any world w in $\mathfrak{S}$.*

Proof:
$\mathfrak{S}, w \models [\alpha]_1\varphi$
$\Leftrightarrow \left(\sum_{(w,w',pr)\in R_\alpha, \mathfrak{S},w'\models\varphi} pr\right) = 1$
$\Leftrightarrow \forall w' \,.\,$ if $\exists pr \,.\, (w, w', pr) \in R_\alpha$ then $\mathfrak{S}, w' \models \varphi$
$\Leftrightarrow \forall w' \,.\,$ if $wR^-_\alpha w'$ then $\mathfrak{S}, w' \models \varphi$. Q.E.D.

Proposition 4.7 *$\langle\alpha\rangle$ has normal semantics. That is, $\mathfrak{S}, w \models \langle\alpha\rangle\varphi$ if and only if there exist w', pr such that $(w, w', pr) \in R_\alpha$ and $\mathfrak{S}, w' \models \varphi$.*

Proof:
$S, w \models \langle\alpha\rangle\varphi$
$\Leftrightarrow \mathfrak{S}, w \models \neg[\alpha]\neg\varphi$
$\Leftrightarrow \mathfrak{S}, w \models \neg[\alpha]_1\neg\varphi$
$\Leftrightarrow \mathfrak{S}, w \not\models [\alpha]_1\neg\varphi$
$\Leftrightarrow \left(\sum_{(w,w',pr)\in R_\alpha, \mathfrak{S},w'\models\neg\varphi} pr\right) \neq 1$
$\Leftrightarrow \exists w', pr \,.\, (w, w', pr) \in R_\alpha$ and $\mathfrak{S}, w' \not\models \neg\varphi$
$\Leftrightarrow \exists w', pr \,.\, (w, w', pr) \in R_\alpha$ and $\mathfrak{S}, w' \models \varphi$. Q.E.D.

5 Specifying Domains with SLAOP

We briefly describe and illustrate a framework to formally specify—in the language of SLAOP—the domain in which an agent or robot is expected to live. Let BK be an agent's *background knowledge* (including non-static formulae) and let IC be its *initial condition*, a static formula describing the world the agent finds itself in when it becomes active. In the context of SLAOP, we are interested in determining $BK \models_{GS} IC \rightarrow \varphi$, where φ is any sentence.

The agent's background knowledge may include *static law axioms* which are facts about the domain that do not change. They have no predictable form, but by definition, they are not dynamic and thus exclude mention of actions. $drank \rightarrow \neg \mathit{full}$ is one static law axiom for the oil-can scenario. The other kinds of axioms in BK are described below.

5.1 The Action Description

In the following discussion, W^φ is the set of worlds in which static formula φ holds (the 'models' of φ). A formal description for the construction of *conditional effect axioms* follows. For one action, there is a set of axioms that take the form

$$\phi_1 \rightarrow ([\alpha]_{q_{11}}\varphi_{11} \wedge \ldots \wedge [\alpha]_{q_{1n}}\varphi_{1n});$$
$$\phi_2 \rightarrow ([\alpha]_{q_{21}}\varphi_{21} \wedge \ldots \wedge [\alpha]_{q_{2n}}\varphi_{2n}); \quad \cdots;$$
$$\phi_j \rightarrow ([\alpha]_{q_{j1}}\varphi_{j1} \wedge \ldots \wedge [\alpha]_{q_{jn}}\varphi_{jn}),$$

where the ϕ_i and φ_{ik} are static, and where the ϕ_i are conditions for the respective effects to be applicable, and in any one axiom, each φ_{ik} represents a set $W^{\varphi_{ik}}$ of worlds. The number q_{ik} is the probability that the agent will end up in a world in $W^{\varphi_{ik}}$, as the effect of performing α in the right condition ϕ_i. For axioms generated from the effect axioms (later in Sec. 5.1), we shall assume that φ_{ik} is a *minimal disjunctive normal form* characterization of $W^{\varphi_{ik}}$. The following constraints apply.

- There must be a set of effect axioms for each action $\alpha \in \mathfrak{A}$.
- The ϕ_i must be *mutually exclusive*, i.e., the conjunction of any pair of conditions causes a contradiction. However, it is not necessary that $W^{\varphi_{i1}} \cup \ldots \cup W^{\varphi_{in}} = C$.
- A set of effects φ_{i1} to φ_{in} in any axiom i must be *mutually exclusive*.
- The *transition probabilities* $q_{i1}, \ldots, q_{in}$ of any axiom i must sum to 1.

The following sentence is an effect axiom for the $grab$ action: $(\mathit{full} \wedge \neg holding) \rightarrow ([grab]_{0.7}(\mathit{full} \wedge holding) \wedge [grab]_{0.2}(\neg \mathit{full} \wedge \neg holding) \wedge [grab]_{0.1}(\mathit{full} \wedge \neg holding))$.

Executability axioms of the form $\phi_k \rightarrow \langle\alpha\rangle\top$ must be supplied, for each action, where ϕ_k is a precondition conveying physical restrictions in the environment with respect to α. The sentence $\neg holding \rightarrow \langle grab\rangle\top$ states that if the robot is not holding the oil-can, then it is possible to grab the can.

A set of axioms must be *generated* that essentially states that if the effect or executability axioms do not imply executability for some action, then that action is inexecutable. Hence, given α, assume the presence of an *executability closure axiom* of the following form: $\neg(\phi_1 \vee \ldots \vee \phi_j \vee \phi_k) \rightarrow \neg\langle\alpha\rangle\top$. The sentence $holding \rightarrow \neg\langle grab\rangle\top$ states that if the robot is holding the oil-can, then it is not possible to grab it.

Now we show the form of sentences that specify what does *not* change under certain conditions—*conditional frame axioms*. Let $\phi_i \rightarrow ([\alpha]_{qi1}\varphi_{i1} \wedge \ldots \wedge [\alpha]_{qin}\varphi_{in})$ be the i-th effect axiom for α. For each $\alpha \in \mathfrak{A}$, for each effect axiom i, do: For each fluent $p \in \mathfrak{P}$, if p is not mentioned in φ_{i1} to φ_{in}, then $(\phi_i \wedge p) \rightarrow [\alpha]p$ and $(\phi_i \wedge \neg p) \rightarrow [\alpha]\neg p$ are part of the domain specification.

For our scenario, the *conditional frame axioms* of $grab$ are

$$(\mathit{full} \wedge \neg holding \wedge drank) \rightarrow [grab]drank;$$
$$(\mathit{full} \wedge \neg holding \wedge \neg drank) \rightarrow [grab]\neg drank;$$
$$(\neg \mathit{full} \wedge \neg holding \wedge drank) \rightarrow [grab]drank;$$
$$(\neg \mathit{full} \wedge \neg holding \wedge \neg drank) \rightarrow [grab]\neg drank.$$

Given frame and effect axioms, it may still happen that the probability to some worlds cannot be logically deduced. Suppose (for the purpose of illustration only) that the sentence

$$[grab]_{0.7}(\mathit{full} \land \mathit{holding}) \land$$
$$[grab]_{0.3}(\mathit{full} \land \lnot \mathit{holding} \land \mathit{drank}). \qquad (1)$$

can be logically deduced from the frame and effect axioms in BK. Now, according to (1) the following worlds are reachable: $(\mathit{full} \land \mathit{holding} \land \mathit{drank})$, $(\mathit{full} \land \mathit{holding} \land \lnot \mathit{drank})$ and $(\mathit{full} \land \lnot \mathit{holding} \land \mathit{drank})$. The transition probability to $(\mathit{full} \land \lnot \mathit{holding} \land \mathit{drank})$ is 0.3, but what are the transition probabilities to $(\mathit{full} \land \mathit{holding} \land \mathit{drank})$ and $(\mathit{full} \land \mathit{holding} \land \lnot \mathit{drank})$? We have devised a process to determine such hidden probabilities via *uniform axioms* [Rens and Meyer, 2011]. Uniform axioms describes how to distribute probabilities of effects uniformly in the case sufficient information is not available. It is very similar to what [Wang and Schmolze, 2005] do to achieve compact representation. A uniform axiom generated for (1) would be

$$[grab]_{0.35}(\mathit{full} \land \mathit{holding} \land \mathit{drank}) \land$$
$$[grab]_{0.35}(\mathit{full} \land \mathit{holding} \land \lnot \mathit{drank}) \land$$
$$[grab]_{0.3}(\mathit{full} \land \lnot \mathit{holding} \land \mathit{drank}).$$

The following axiom schema represents all the *effect condition closure axioms*. $(\lnot(\phi_1 \lor \ldots \lor \phi_j) \land P) \to [A]P$, where there is a different axiom for each substitution of $\alpha \in \mathfrak{A}$ for A and each literal for P. For example, $(\mathit{holding} \land P) \to [grab]P$, where P is any $p \in \mathfrak{P}$ or its negation.

5.2 The Perception Description

One can classify actions as either *ontic* (physical) or *sensory*. This classification also facilitates specification of perceivability. Ontic actions have intentional ontic effects, that is, effects on the environment that were the main intention of the agent. $grab$, $drink$ and $replace$ are ontic actions. Sensory actions—$weigh$ in our scenario—result in perception, maybe with (unintended) side-effects.

Perceivability axioms specify what conditions must hold after the applicable action is performed, for the observation to be perceivable. Ontic actions each have perceivability axioms of the form $(\mathit{obsNil} \mid \alpha)^{\Box}$. Sensory actions typically have multiple observations and associated conditions for perceiving them. The probabilities for perceiving the various observations associated with sensory actions must be specified. The following set of perceivability axiom schemata does this:

$$\phi_{11} \to (\varsigma_1 \mid \alpha)_{q_{11}}; \quad \phi_{12} \to (\varsigma_1 \mid \alpha)_{q_{12}}; \quad \cdots;$$
$$\phi_{1j} \to (\varsigma_1 \mid \alpha)_{q_{1n}}; \quad \phi_{21} \to (\varsigma_2 \mid \alpha)_{q_{21}}; \quad \cdots;$$
$$\phi_{nk} \to (\varsigma_n \mid \alpha)_{q_{kn}},$$

where $\{\varsigma_1, \varsigma_2, \ldots, \varsigma_n\}$ is the set of first components of all elements in Q_α and the ϕ_i are the conditions expressed as static formulae. The following constraints apply to these axioms.

- There must be a set of perceivability axioms for each action $\alpha \in \mathfrak{A}$.
- In the semantics section, item 7 of the definition of a SLAOP structure states that for every world reachable via some action, there exists an observation associated with the action, perceivable in that world. The perceivability axioms must adhere to this remark.
- For every pair of perceivability axioms $\phi \to (\varsigma \mid \alpha)_q$ and $\phi' \to (\varsigma \mid \alpha)_{q'}$ for the same observation ς, W^{ϕ} must be disjoint from $W^{\phi'}$.
- For every particular condition ϕ, $\sum_{\phi \to (\varsigma \mid \alpha)_q} q = 1$. This is so that $\sum_{N(\varsigma):(N(\varsigma),w^+,pr) \in Q_\alpha} pr = 1$.

Some perceivability axioms for the oil-can scenario might be

$$(\mathit{obsNil} \mid grab)^{\Box};$$
$$(\lnot \mathit{full} \land \mathit{drank} \land \mathit{holding}) \to (\mathit{obsLight} \mid \mathit{weigh})_{0.7};$$
$$(\lnot \mathit{full} \land \mathit{drank} \land \mathit{holding}) \to (\mathit{obsHeavy} \mid \mathit{weigh})_{0.1};$$
$$(\lnot \mathit{full} \land \mathit{drank} \land \mathit{holding}) \to (\mathit{obsMedium} \mid \mathit{weigh})_{0.2}.$$

Perceivability axioms for sensory actions also state *when* the associated observations are *possible*. The following set of axioms states when the associated observations are *impossible* for sensory action $weigh$ of our scenario.

$$((\lnot \mathit{full} \land \mathit{drank} \land \lnot \mathit{holding}) \lor (\mathit{full} \land \lnot \mathit{drank} \land \lnot \mathit{holding})) \to \lnot(l\mathit{obsLight} \mid \mathit{weigh})^{\Diamond};$$
$$((\lnot \mathit{full} \land \mathit{drank} \land \lnot \mathit{holding}) \lor (\mathit{full} \land \lnot \mathit{drank} \land \lnot \mathit{holding})) \to \lnot(\mathit{obsHeavy} \mid \mathit{weigh})^{\Diamond};$$
$$((\lnot \mathit{full} \land \mathit{drank} \land \lnot \mathit{holding}) \lor (\mathit{full} \land \lnot \mathit{drank} \land \lnot \mathit{holding})) \to \lnot(\mathit{obsMedium} \mid \mathit{weigh})^{\Diamond}.$$

The *perceivability condition closure axiom* schema is

$$\lnot(\phi_{11} \lor \cdots \lor \phi_{1j}) \to \lnot(\varsigma_1 \mid \alpha)^{\Diamond};$$
$$\lnot(\phi_{21} \lor \cdots) \to \lnot(\varsigma_2 \mid \alpha)^{\Diamond}; \quad \cdots;$$
$$\lnot(\cdots \lor \phi_{nk}) \to \lnot(\varsigma_n \mid \alpha)^{\Diamond},$$

where the ϕ_i are taken from the perceivability axioms. There are no perceivability closure axioms for ontic actions, because they are always tautologies.

Ontic actions each have *unperceivability axioms* of the form $(\forall v^{\varsigma})((v^{\varsigma} \mid \alpha)^{\Diamond} \leftrightarrow v^{\varsigma} = \mathit{obsNil})$. The axiom says that no other observation is perceivable given the ontic action. That is, for any instantiation of an observation ς' other than *obsNil*, $\lnot(\varsigma' \mid \alpha)^{\Diamond}$ is a logical consequence.

For sensory actions, to state that the observations not associated with action α are always impossible given α was executed, we need an axiom of the form $(\forall v^{\varsigma})(v^{\varsigma} \neq o_1 \land v^{\varsigma} \neq o_2 \land \cdots \land v^{\varsigma} \neq o_n) \to \lnot(v^{\varsigma} \mid \alpha)^{\Diamond}$. For the oil-can scenario, they are

$$(\forall v^{\varsigma})(v^{\varsigma} \mid grab)^{\Diamond} \leftrightarrow v^{\varsigma} = \mathit{obsNil};$$
$$(\forall v^{\varsigma})(v^{\varsigma} \mid \mathit{drink})^{\Diamond} \leftrightarrow v^{\varsigma} = \mathit{obsNil};$$
$$(\forall v^{\varsigma})(v^{\varsigma} \mid \mathit{replace})^{\Diamond} \leftrightarrow v^{\varsigma} = \mathit{obsNil};$$
$$(\forall v^{\varsigma})(v^{\varsigma} \neq \mathit{obsHeavy} \land v^{\varsigma} \neq \mathit{obsLight} \land v^{\varsigma} \neq \mathit{obsMedium}) \to \lnot(v^{\varsigma} \mid \mathit{weigh})^{\Diamond}.$$

5.3 The Utility Function

A sufficient set of axioms concerning 'state rewards' and 'action costs' constitutes a *utility function*.

There must be a means to express the reward an agent will get for performing an action in a world it may find itself—for every action and every possible world. The domain expert must supply a set of *reward axioms* of the form $\phi_i \rightarrow Reward(r_i)$, where ϕ_i is a condition specifying the world in which the rewards can be got (e.g., $holding \rightarrow Reward(5)$ and $drank \rightarrow Reward(10)$).

The conditions of the reward axioms must identify worlds that are pairwise disjoint. This holds for cost axioms too:

The domain expert must also supply a set of *cost axioms* of the form $(\phi_i \wedge \langle\alpha\rangle\top) \rightarrow Cost(\alpha, c_i)$, where ϕ_i is a condition specifying the world in which the cost c_i will be incurred for action α. For example,

$$\begin{aligned}
&(\mathit{full} \wedge \langle \mathit{grab}\rangle\top) \rightarrow Cost(grab, 2);\\
&(\neg \mathit{full} \wedge \langle \mathit{grab}\rangle\top) \rightarrow Cost(grab, 1);\\
&(\mathit{full} \wedge \langle \mathit{drink}\rangle\top) \rightarrow Cost(drink, 2);\\
&(\neg \mathit{full} \wedge \langle \mathit{drink}\rangle\top) \rightarrow Cost(drink, 1);\\
&\langle \mathit{replace}\rangle\top \rightarrow Cost(replace, 0.8).
\end{aligned}$$

5.4 A Frame Solution

The method we propose for avoiding generating all the frame and effect closure axioms, is to write the effect and executability axioms, generate the uniform axioms, and then generate a set of a new kind of axioms representing the frame and effect closure axioms much more compactly. By looking at the effect axioms of a domain, one can define for each fluent $p \in \mathfrak{P}$ a set $Cause^+(p)$ of actions that can (but not necessarily always) cause p (as a positive literal) to flip to $\neg p$, and a set $Cause^-(p)$ of actions that can (but not necessarily always) causes $\neg p$ (as a negative literal) to flip to p.[1] For instance, $grab \in Cause^+(full)$, because in effect axiom

$$\begin{aligned}
&(full \wedge \neg holding) \rightarrow\\
&\quad([\mathit{grab}]_{0.7}(full \wedge holding) \wedge\\
&\quad[\mathit{grab}]_{0.2}(\neg full \wedge \neg holding) \wedge [\mathit{grab}]_{0.1}(full \wedge \neg holding)),
\end{aligned}$$

$grab$ flips $full$ to $\neg full$ (with probability 0.2). The axiom also shows that $grab \in Cause^-(holding)$ because it flips $\neg holding$ to $holding$ (with probability 0.7). The actions mentioned in these sets may have deterministic or stochastic effects on the respective propositions.

Furthermore, by looking at the effects axioms, $Cond$ functions can be defined: For each $\alpha \in Cause^+(p)$, $Cond^+(\alpha, p)$ returns a sentence that represents the disjunction of all ϕ_i under which α caused p to be a negative literal. $Cond^-(\alpha, p)$ is defined similarly.

Suppose that $Cause^+(p) = \{\alpha_1, \ldots, \alpha_m\}$ and $Cause^-(p) = \{\beta_1, \ldots, \beta_n\}$. We propose, for any fluent p, a pair of *compact frame axioms* with schema

$$\begin{aligned}
&(\forall v^\alpha)p \rightarrow\\
&\qquad(v^\alpha = \alpha_1 \wedge \neg Cond^+(\alpha_1, p)) \rightarrow [\alpha_1]p \wedge\\
&\qquad\vdots\\
&\qquad(v^\alpha = \alpha_m \wedge \neg Cond^+(\alpha_m, p)) \rightarrow [\alpha_m]p \wedge\\
&\qquad(v^\alpha \neq \alpha_1 \wedge \cdots \wedge v^\alpha \neq \alpha_m) \rightarrow [v^\alpha]p
\end{aligned}$$

and

$$\begin{aligned}
&(\forall v^\alpha)\neg p \rightarrow\\
&\qquad(v^\alpha = \beta_1 \wedge \neg Cond^-(\beta_1, p)) \rightarrow [\beta_1])\neg p \wedge\\
&\qquad\vdots\\
&\qquad(v^\alpha = \beta_m \wedge \neg Cond^-(\beta_m, p)) \rightarrow [\beta_m])\neg p \wedge\\
&\qquad(v^\alpha \neq \beta_1 \wedge \cdots \wedge v^\alpha \neq \beta_m) \rightarrow [v^\alpha])\neg p.
\end{aligned}$$

Claim 5.1 *The collection of pairs of compact frame axioms for each fluent in $\mathfrak{P}$ is logically equivalent to the collection of all conditional frame axioms and effect closure axioms generated with the processes presented above.*

Proof:
Please refer to our draft report [Rens and Meyer, 2011]. Q.E.D.

There are in the order of $|\mathfrak{A}| \cdot 2|\Omega| \cdot D$ frame axioms, where D is the average number of conditions on effects per action (the ϕ_i). Let N be the average size of $|Cause^+(p)|$ or $|Cause^-(p)|$ for any $p \in \mathfrak{P}$. With the two compact frame axioms (per fluent), no separate frame or effect closure axioms are required in the action description (AD). If we consider each of the most basic conjuncts and disjuncts as a unit length, then the size of each compact frame axiom is $\mathcal{O}(N)$, and the size of all compact frame axioms in AD is in the order of $N \cdot 2|\mathfrak{P}|$. For reasonable domains, N will be much smaller than $|\mathfrak{A}|$, and the size of all compact frame axioms is thus much smaller than the size of all frame and effect closure axioms ($|\mathfrak{A}| \cdot 2|\mathfrak{P}| \cdot (D + 1)$).

5.5 Some Example Entailment Results

The following entailments have been proven concerning the oil-can scenario [Rens and Meyer, 2011]. BK^{oc} is the background knowledge of an agent in the scenario. To save space and for neater presentation, we abbreviate constants and fluents by their initials.

$BK^{oc} \models_{GS} (f \wedge d \wedge \neg h) \rightarrow [g]_{0.7}(f \wedge d \wedge h)$:
If the can is full and the oil has been drunk, the probability of successfully grabbing it without spilling oil is 0.7.

$BK^{oc} \models_{GS} (f \wedge \neg d \wedge h) \rightarrow \neg[d]_{0.2}(f \vee \neg d \vee \neg h)$:
If the robot is in a situation where it is holding the full oil-can (and has not yet attempted drinking), then the probability of having failed to drink the oil is not 0.2.

$BK^{oc} \models_{GS} (\exists v^\varsigma)(v^\varsigma \mid drink)^\Box$:
In any world, there always exists an observation after the robot has drunk.

$BK^{oc} \models_{GS} \langle d\rangle\top \leftrightarrow h$:
In any world, it is possible to drink the oil if and only if the can is being held.

$BK^{oc} \models_{GS} (f \wedge \langle d\rangle\top) \rightarrow \neg Cost(d, 3)$:
Assuming it is possible to drink and the can is full of oil, then the cost of doing the drink action is not 3 units.

6 Concluding Remarks

We introduced a formal language specifically for robots that must deal with uncertainty in affection and perception. It is one step towards a general reasoning system for robots, not the actual system.

[1] Such sets and functions are also employed by Demolombe, Herzig and Varzinczak [Demolombe *et al.*, 2003].

POMDP theory is used as an underlying modeling formalism. The formal language is based on multi-modal logic and accepts basic principals of cognitive robotics. We have also included notions of probability to represent the uncertainty, but we have done so 'minimally', that is, only as far as is necessary to represent POMDPs for the intended application. Beyond the usual elements of logics for reasoning about action and change, the logic presented here adds *observations* as first-class objects, and a means to represent *utility functions*.

In an associated report [Rens and Meyer, 2011], the *frame problem* is addressed, and we provided a *belief network* approach to domain specification for cases when the required information is available.

The computational complexity of SLAOP was not determined, and is left for future work. Due to the nature of SLAOP structures, we conjecture that entailment in SLAOP is decidable. It's worth noting that the three latter frameworks discussed in Section 2 [Wang and Schmolze, 2005; Sanner and Kersting, 2010; Poole, 1998] do not mention decidability results either.

The next step is to prove decidability of SLAOP entailment, and then to develop a logic for decision-making in which SLAOP will be employed. Domains specified in SLAOP will be used to make decisions in the 'meta' logic, with sentences involving sequences of actions and the epistemic knowledge of an agent. This will also show the significance of SLAOP in a more practical context. Please refer to our extended abstract [Rens, 2011] for an overview of our broader research programme.

References

[Bacchus *et al.*, 1999] F. Bacchus, J. Y. Halpern, and H. J. Levesque. Reasoning about noisy sensors and effectors in the situation calculus. *Artificial Intelligence*, 111(1–2):171–208, 1999.

[Bonet and Geffner, 2001] B. Bonet and H. Geffner. Planning and control in artificial intelligence: A unifying perspective. *Applied Intelligence*, 14(3):237–252, 2001.

[Boutilier and Poole, 1996] C. Boutilier and D. Poole. Computing optimal policies for partially observable decision processes using compact representations. In *Proc. of 13th Natl. Conf. on AI*, pages 1168–1175, 1996.

[Boutilier *et al.*, 2000] C. Boutilier, R. Reiter, M. Soutchanski, and S. Thrun. Decision-theoretic, high-level agent programming in the situation calculus. In *Proc. of 17th Natl. Conf. on AI*, pages 355–362. AAAI Press, Menlo Park, CA, 2000.

[Demolombe *et al.*, 2003] R. Demolombe, A. Herzig, and I. Varzinczak. Regression in modal logic. *Journal of Applied Non-Classical Logics*, 13(2):165–185, 2003.

[Fagin and Halpern, 1994] R. Fagin and J. Y. Halpern. Reasoning about knowledge and probability. *J. of ACM*, 41(2):340–367, 1994.

[Gabaldon and Lakemeyer, 2007] A. Gabaldon and G. Lakemeyer. $\mathcal{ESP}$: A logic of only-knowing, noisy sensing and acting. In *Proc. of 22nd Natl. Conf. on AI*, pages 974–979, 2007.

[Halpern, 2003] J. Y. Halpern. *Reasoning about Uncertainty*. The MIT Press, Cambridge, MA, 2003.

[Iocchi *et al.*, 2009] L. Iocchi, T. Lukasiewicz, D. Nardi, and R. Rosati. Reasoning about actions with sensing under qualitative and probabilistic uncertainty. *ACM Transactions on Computational Logic*, 10(1):5:1–5:41, 2009.

[Kaelbling *et al.*, 1998] L. Kaelbling, M. Littman, and A. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1–2):99–134, 1998.

[Levesque and Lakemeyer, 2008] H. Levesque and G. Lakemeyer. Cognitive Robotics. In B. Porter F. Van Harmelen, V. Lifshitz, editor, *Handbook of Knowledge Representation*, pages 869–886. Elsevier Science, 2008.

[Poole, 1998] D. Poole. Decision theory, the situation calculus and conditional plans. *Linköping Electronic Articles in Computer and Information Science*, 8(3), 1998.

[Rens and Meyer, 2011] G. Rens and T. Meyer. Logic and utility based agent planning language, Part II: Specifying stochastic domains. Technical Report KRR-10-01, KRR, CSIR Meraka Institute, Pretoria, South Africa, January 2011. url: http://krr.meraka.org.za/publications/2011.

[Rens *et al.*, 2010] G. Rens, I. Varzinczak, T. Meyer, and A. Ferrein. A logic for reasoning about actions and explicit observations. In Jiuyong Li, editor, *Proc. of 23rd Australasian Joint Conf. on AI*, pages 395–404, 2010.

[Rens, 2010] G. Rens. A belief-desire-intention architecture with a logic-based planner for agents in stochastic domains. Master's thesis, School of Computing, University of South Africa, 2010.

[Rens, 2011] G. Rens. From an agent logic to an agent programming language for partially observable stochastic domains. In *Proc. of 22nd Intl. Joint Conf. on AI*, Menlo Park, CA, 2011. AAAI Press. To appear.

[Sanner and Kersting, 2010] S. Sanner and K. Kersting. Symbolic dynamic programming for first-order POMDPs. In *Proc. of 24th Natl. Conf. on AI*, pages 1140–1146, 2010.

[Shirazi and Amir, 2007] A. Shirazi and E. Amir. Probabilistic modal logic. In *Proc. of 22nd Natl. Conf. on AI*, pages 489–494. AAAI Press, 2007.

[Van Benthem *et al.*, 2009] J. Van Benthem, J. Gerbrandy, and B. Kooi. Dynamic update with probabilities. *Studia Logica*, 93(1):67–96, 2009.

[Van Diggelen, 2002] J. Van Diggelen. Using modal logic in mobile robots. Master's thesis, Cognitive Artificial Intelligence, Utrecht University, 2002.

[Wang and Schmolze, 2005] C. Wang and J. Schmolze. Planning with POMDPs using a compact, logic-based representation. In *Proc. of 17th IEEE Intl. Conf. on Tools with AI*, pages 523–530, 2005.

[Weerdt *et al.*, 1999] M. De Weerdt, F. De Boer, W. Van der Hoek, and J.-J. Meyer. Imprecise observations of mobile robots specified by a modal logic. In *Proc. of ASCI-99*, pages 184–190, 1999.

Agent Supervision in Situation-Determined ConGolog

Giuseppe De Giacomo
Sapienza – Università di Roma
Rome, Italy
degiacomo@dis.uniroma1.it

Yves Lespérance
York University
Toronto, Canada
lesperan@cse.yorku.ca

Christian Muise
University of Toronto
Toronto, Canada
cjmuise@cstoronto.edu

Abstract

We investigate agent supervision, a form of customization, which constrains the actions of an agent so as to enforce certain desired behavioral specifications. This is done in a setting based on the Situation Calculus and a variant of the ConGolog programming language which allows for nondeterminism, but requires the remainder of a program after the execution of an action to be determined by the resulting situation. Such programs can be fully characterized by the set of action sequences that they generate. The main results are a characterization of the maximally permissive supervisor that minimally constrains the agent so as to enforce the desired behavioral constraints when some agent actions are uncontrollable, and a sound and complete technique to execute the agent as constrained by such a supervisor.

1 Introduction

There has been much work on *process customization*, where a generic process for performing a task or achieving a goal is customized to satisfy a client's constraints or preferences [Fritz and McIlraith, 2006; Lin *et al.*, 2008; Sohrabi *et al.*, 2009]. This approach was originally proposed in [McIlraith and Son, 2002] in the context of web service composition [Su, 2008]. The idea is that the generic process provides a wide range of alternative ways to perform the task. During customization, alternatives that violate the constraints are eliminated. Some parameters in the remaining alternatives may be restricted or instantiated so as to ensure that any execution of the customized process will satisfy the client's constraints. Another approach to service composition synthesizes an orchestrator that controls the execution of a set of available services to ensure that they realize a desired service [Sardiña and De Giacomo, 2009; Bertoli *et al.*, 2010].

In this paper, we develop a framework for a similar type of process refinement that we call *supervised execution*. We assume that we have a nondeterministic process that specifies the possible behaviors of an agent, and a second process that specifies the possible behaviors that a supervisor wants to allow (or alternatively, of the behaviors that it wants to rule out). For example, we could have an agent process representing a child and its possible behaviors, and a second process representing a babysitter that specifies the behaviors by the child that can be allowed. If the supervisor can control all the actions of the supervised agent, then it is straightforward to specify the behaviors that may result as a kind of synchronized concurrent execution of the agent and supervisor processes. A more interesting case arises when some agent actions are *uncontrollable*. For example, it may be impossible to prevent the child from getting muddy once he/she is allowed outside. In such circumstances, the supervisor may have to block some agent actions, not because they are undesirable in themselves (e.g. going outside), but because if they are allowed, the supervisor cannot prevent the agent from performing some undesirable actions later on (e.g. getting muddy).

We follow previous work [McIlraith and Son, 2002; Fritz and McIlraith, 2006] in assuming that processes are specified in a high level agent programming language defined in the Situation Calculus [Reiter, 2001].[1] In fact, we define and use a restricted version of the ConGolog agent programming language [De Giacomo *et al.*, 2000] that we call Situation-Determined ConGolog (SDConGolog). In this version, following [De Giacomo *et al.*, 2010] all transitions involve performing an action (i.e. there are no transitions that merely perform a test). Moreover, nondeterminism is restricted so that the remaining program is a function of the action performed, i.e. there is a unique remaining program δ' such that a given program δ can perform a transition $(\delta, s) \rightarrow_a (\delta', do(a, s))$ involving action a in situation s. This means that a run of such a program starting in a given situation can be taken to be simply a sequence of actions, as all the intermediate programs one goes through are functionally determined by the starting program and situation and the actions performed. Thus we can see a program and a starting situation as specifying a language, that of all the sequences of actions that are runs of the program in the situation. This allows us to define language theoretic notions such as union, intersection, and difference/complementation in terms of op-

[1] Clearly, there are applications where a declarative formalism is preferable, e.g. linear temporal logic (LTL), regular expressions over actions, or some type of business rules. However, there has been previous work on compiling such declarative specification languages into ConGolog, for instance [Fritz and McIlraith, 2006], which handles an extended version of LTL interpreted over a finite horizon.

erations on the corresponding programs, which has applications in many areas (e.g. programming by demonstration and programming by instruction [Fritz and Gil, 2010], and plan recognition [Demolombe and Hamon, 2002]).Working with situation-determined programs also greatly facilitates the formalization of supervision/customization. In [De Giacomo *et al.*, 2010], it is in fact shown that any ConGolog program can be made situation-determined by recording nondeterministic choices made in the situation.

Besides a detailed characterization of SDConGolog,[2] the main contributions of the paper are as follows: first, based on previous work in discrete event control [Wonham and Ramadge, 1987], we provide a characterization of the *maximally permissive supervisor* that minimally constrains the actions of the agent so as to enforce the desired behavioral specifications, showing its existence and uniqueness; secondly, we define a program construct for *supervised execution* that takes the agent program and supervisor program, and executes them to obtain only runs allowed by the maximally permissive supervisor, showing its soundness and completeness.

The rest of the paper proceeds as follows. In the next section, we briefly review the Situation Calculus and the ConGolog agent programming language. In Section 3, we define SDConGolog, discuss its properties, and introduce some useful programming constructs and terminology. Then in Section 4, we develop our account of agent supervision, and define the maximal permissive supervisor and supervised execution. Finally in Section 5, we review our contributions and discuss related and future work.

2 Preliminaries

The *situation calculus* is a logical language specifically designed for representing and reasoning about dynamically changing worlds [Reiter, 2001]. All changes to the world are the result of *actions*, which are terms in the language. We denote action variables by lower case letters a, action types by capital letters A, and action terms by α, possibly with subscripts. A possible world history is represented by a term called a *situation*. The constant S_0 is used to denote the initial situation where no actions have yet been performed. Sequences of actions are built using the function symbol do, such that $do(a, s)$ denotes the successor situation resulting from performing action a in situation s. Predicates and functions whose value varies from situation to situation are called *fluents*, and are denoted by symbols taking a situation term as their last argument (e.g., $Holding(x, s)$). Within the language, one can formulate action theories that describe how the world changes as the result of actions [Reiter, 2001].

To represent and reason about complex actions or processes obtained by suitably executing atomic actions, various so-called *high-level programming languages* have been defined. Here we concentrate on (a fragment of) ConGolog that includes the following constructs:

α	atomic action
$\varphi?$	test for a condition
$\delta_1;\delta_2$	sequence
if φ **then** δ_1 **else** δ_2	conditional
while φ **do** δ	while loop
$\delta_1 \vert \delta_2$	nondeterministic branch
$\pi x.\delta$	nondeterministic choice of argument
δ^*	nondeterministic iteration
$\delta_1 \Vert \delta_2$	concurrency

In the above, α is an action term, possibly with parameters, and φ is situation-suppressed formula, that is, a formula in the language with all situation arguments in fluents suppressed. As usual, we denote by $\varphi[s]$ the situation calculus formula obtained from φ by restoring the situation argument s into all fluents in φ. Program $\delta_1|\delta_2$ allows for the nondeterministic choice between programs δ_1 and δ_2, while $\pi x.\delta$ executes program δ for *some* nondeterministic choice of a legal binding for variable x (observe that such a choice is, in general, unbounded). δ^* performs δ zero or more times. Program $\delta_1\|\delta_2$ expresses the concurrent execution (interpreted as interleaving) of programs δ_1 and δ_2.

Formally, the semantics of ConGolog is specified in terms of single-step transitions, using the following two predicates [De Giacomo *et al.*, 2000]: *(i)* $Trans(\delta, s, \delta', s')$, which holds if one step of program δ in situation s may lead to situation s' with δ' remaining to be executed; and *(ii)* $Final(\delta, s)$, which holds if program δ may legally terminate in situation s. The definitions of $Trans$ and $Final$ we use are as in [De Giacomo *et al.*, 2010]; these are in fact the usual ones [De Giacomo *et al.*, 2000], except that the test construct $\varphi?$ does not yield any transition, but is final when satisfied. Thus, it is a *synchronous* version of the original test construct (it does not allow interleaving). A consequence of this is that in the version of ConGolog that we use, every transition involves the execution an action (tests do not make transitions), i.e.,

$$\Sigma \cup \mathcal{C} \models \mathit{Trans}(\delta, s, \delta', s') \supset \exists a. s' = \mathit{do}(a, s).$$

Here and in the remainder, we use Σ to denote the foundational axioms of the situation calculus from [Reiter, 2001] and $\mathcal{C}$ to denote the axioms defining the ConGolog language.

3 Situation-Determined Programs

As mentioned earlier, we are interested in process customization. For technical reasons, we will focus on a restricted class of ConGolog programs for describing processes, namely "situation-determined programs". A program δ is *situation-determined* in a situation s if for every sequence of transitions, the remaining program is determined by the resulting situation, i.e.,

$$\begin{array}{l} \mathit{SituationDetermined}(\delta, s) \doteq \forall s', \delta', \delta''. \\ \quad \mathit{Trans}^*(\delta, s, \delta', s') \wedge \mathit{Trans}^*(\delta, s, \delta'', s') \supset \delta' = \delta'', \end{array}$$

where *Trans** denotes the reflexive transitive closure of *Trans*. Thus, a (partial) execution of a situation-determined program is uniquely determined by the sequence of actions it has produced. This is a key point. In general, the possible executions of a ConGolog program are characterized by sequences of configurations formed by the remaining program and the current situation. In contrast, the *execution of situation-determined programs can be characterized in terms of sequences of actions only*, those sequences that correspond to the situations reached from where the program started.

[2] In [De Giacomo *et al.*, 2010], situation-determined programs were only dealt with incidentally.

For example, the ConGolog program $(a;b) \mid (a;c)$ is not situation-determined in situation S_0 as it can make a transition to a configuration $(b, do(a, S_0))$, where the situation is $do(a, S_0)$ and the remaining program is b, and it can also make a transition to a configuration $(c, do(a, S_0))$, where the situation is also $do(a, S_0)$ and the remaining program is instead c. It is impossible to determine what the remaining program is given only a situation, e.g. $do(a, S_0)$, reached along an execution. In contrast, the program $a; (b \mid c)$ is situation-determined in situation S_0. There is a unique remaining program $(b \mid c)$ in situation $do(a, S_0)$ (and similarly for the other reachable situations).

When we restrict our attention to situation-determined programs, we can use a simpler semantic specification for the language; instead of *Trans* we can use a $next$ (partial) function, where $next(\delta, a, s)$ returns the program that remains after δ does a transition involving action a in situation s (if δ is situation determined, such a remaining program must be unique). We will axiomatize the $next$ function so that it satisfies the following properties:

$$next(\delta, a, s) = \delta' \wedge \delta' \neq \bot \supset \mathit{Trans}(\delta, s, \delta', do(a, s)) \quad \text{(N1)}$$

$$\exists! \delta'.\mathit{Trans}(\delta, s, \delta', do(a, s)) \supset \forall \delta'.(\mathit{Trans}(\delta, s, \delta', do(a, s)) \supset next(\delta, a, s) = \delta') \quad \text{(N2)}$$

$$\neg \exists! \delta'.\mathit{Trans}(\delta, s, \delta', do(a, s)) \supset next(\delta, a, s) = \bot \quad \text{(N3)}$$

Here $\exists! x.\phi(x)$ means that there exists a unique x such that $\phi(x)$; this is defined in the usual way. $\bot$ is a special value that stands for "undefined". The function $next(\delta, a, s)$ is only defined when there is a unique remaining program after program δ does a transition involving the action a; if there is such a unique remaining program, then $next(\delta, a, s)$ denotes it.

We define the function $next$ inductively on the structure of programs using the following axioms:

Atomic action:

$$next(\alpha, a, s) = \begin{cases} nil & \text{if } Poss(a, s) \text{ and } \alpha = a \\ \bot & \text{otherwise} \end{cases}$$

Sequence: $next(\delta_1; \delta_2, a, s) =$

$$\begin{cases} next(\delta_1, a, s); \delta_2 & \text{if } next(\delta_1, a, s) \neq \bot \text{ and} \\ & (\neg Final(\delta_1, s) \text{ or } next(\delta_2, a, s) = \bot) \\ next(\delta_2, a, s) & \text{if } \mathit{Final}(\delta_1, s) \text{ and } next(\delta_1, a, s) = \bot \\ \bot & \text{otherwise} \end{cases}$$

Conditional:

$$next(\textbf{if } \varphi \textbf{ then } \delta_1 \textbf{ else } \delta_2, a, s) = \begin{cases} next(\delta_1, a, s) & \text{if } \varphi[s] \\ next(\delta_2, a, s) & \text{if } \neg\varphi[s] \end{cases}$$

Loop:

$$next(\textbf{while } \varphi \textbf{ do } \delta, a, s) = \begin{cases} next(\delta, a, s); \textbf{while } \varphi \textbf{ do } \delta \\ \quad \text{if } \varphi[s] \text{ and } next(\delta, a, s) \neq \bot \\ \bot \text{ otherwise} \end{cases}$$

Nondeterministic branch:

$$next(\delta_1 | \delta_2, a, s) = \begin{cases} next(\delta_1, a, s) \text{ if } next(\delta_2, a, s) = \bot \text{ or} \\ \quad next(\delta_2, a, s) = next(\delta_1, a, s) \\ next(\delta_2, a, s) \text{ if } next(\delta_1, a, s) = \bot \\ \bot \text{ otherwise} \end{cases}$$

Nondeterministic choice of argument:

$$next(\pi x.\delta, a, s) = \begin{cases} next(\delta^x_d, a, s) & \text{if } \exists! d.next(\delta^x_d, a, s) \neq \bot \\ \bot \text{ otherwise} \end{cases}$$

Nondeterministic iteration:

$$next(\delta^*, a, s) = \begin{cases} next(\delta, a, s); \delta^* & \text{if } next(\delta, a, s) \neq \bot \\ \bot \text{ otherwise} \end{cases}$$

Interleaving concurrency: $next(\delta_1 \| \delta_2, a, s) =$

$$\begin{cases} next(\delta_1, a, s) \| \delta_2 \\ \quad \text{if } next(\delta_1, a, s) \neq \bot \text{ and } next(\delta_2, a, s) = \bot \\ \delta_1 \| next(\delta_2, a, s) \\ \quad \text{if } next(\delta_2, a, s) \neq \bot \text{ and } next(\delta_1, a, s) = \bot \\ \bot \text{ otherwise} \end{cases}$$

Test, empty program, undefined:

$next(\varphi?, a, s) = \bot \quad next(nil, a, s) = \bot \quad next(\bot, a, s) = \bot$

Moreover the undefined program is never *Final*: $\mathit{Final}(\bot, s) \equiv \texttt{false}$.

Let $\mathcal{C}^n$ be the set of ConGolog axioms extended with the above axioms specifying $next$ and $\mathit{Final}(\bot, s)$. It is easy to show that:

Proposition 1 *Properties N1, N2, and N3 are entailed by* $\Sigma \cup \mathcal{C}^n$.

Note in particular that as per N3, if the remaining program is not uniquely determined, then $next(\delta, a, s)$ is undefined. Notice that for situation-determined programs this will never happen, and if $next(\delta, a, s)$ returns $\bot$ it is because δ cannot make any transition using a in s:

Corollary 2

$$\Sigma \cup \mathcal{C}^n \models \forall \delta, s.SituationDetermined(\delta, s) \supset \forall a\, [(next(\delta, a, s) = \bot) \;\equiv\; (\neg \exists \delta'.\mathit{Trans}(\delta, s, \delta', do(a, s)))].$$

Let's look at an example. Imagine an agent specified by δ_{B1} below that can repeatedly pick an available object and repeatedly use it and then discard it, with the proviso that if during use the object breaks, the agent must repair it:

$$\delta_{B1} = [\pi\, x.Available(x)?; [use(x); (nil \mid [break(x); repair(x)])]^*; discard(x)]^*$$

We assume that there is a countably infinite number of available unbroken objects initially, that objects remain available until they are discarded, that available objects can be used if they are unbroken, and that objects are unbroken unless they break and are not repaired (this is straightforwardly axiomatized in the situation calculus). Notice that this program is situation-determined, though very nondeterministic.

Language theoretic operations on programs. We can extend the SDConGolog language so as to close it with respect to language theoretic operations, such as *union*, *intersection* and *difference/complementation*. We can already see the nondeterministic branch construct as a union operator, and intersection and difference can be defined as follows:

Intersection/synchronous concurrency:

$$next(\delta_1 \;\&\; \delta_2, a, s) = \begin{cases} next(\delta_1, a, s) \;\&\; next(\delta_2, a, s) \\ \quad \text{if both are different from } \bot \\ \bot \text{ otherwise} \end{cases}$$

Difference: $next(\delta_1 - \delta_2, a, s) =$

$$\begin{cases} next(\delta_1, a, s) - next(\delta_2, a, s) \text{ if both are different from } \bot \\ next(\delta_1, a, s) \quad \text{if } next(\delta_2, a, s) = \bot \\ \bot \text{ if } next(\delta_1, a, s) = \bot \end{cases}$$

For these new constructs, *Final* is defined as follows:

$$Final(\delta_1 \mathbin{\&} \delta_2, s) \equiv Final(\delta_1, s) \wedge Final(\delta_2, s)$$
$$Final(\delta_1 - \delta_2, s) \equiv Final(\delta_1, s) \wedge \neg Final(\delta_2, s)$$

We can express the *complement* of a program δ using difference as follows: $(\pi a.a)^* - \delta$.

It is easy to check that Proposition 1 and Corollary 2 also hold for programs involving these new constructs.

As we will see later, synchronous concurrency can be used to constrain/customize a process. Difference can be used to prohibit certain process behaviors: $\delta_1 - \delta_2$ is the process where δ_1 is executed but δ_2 is not.

To illustrate, consider an agent specified by program δ_{S1} that repeatedly picks an available object and does anything to it provided it is broken at most once before it is discarded:

$$\begin{aligned} \delta_{S1} = [&\pi\, x.Available(x)?; \\ &[\pi\, a.(a-(break(x) \mid discard(x)))]^*; \\ &(nil \mid (break(x)); [\pi\, a.(a-(break(x) \mid discard(x)))]^*); \\ &discard(x)]^* \end{aligned}$$

Sequences of actions generated by programs. We can extend the function $next$ to the function $next^*(\delta, \vec{a}, s)$ that takes a program δ, a finite sequence of actions $\vec{a}$,[3] and a situation s, and returns the remaining program δ' after executing δ in s producing the sequence of actions $\vec{a}$, defined by induction on the length of the sequence of actions as follows:

$$next^*(\delta, \epsilon, s) = \delta$$
$$next^*(\delta, a\vec{a}, s) = next^*(next(\delta, a, s), \vec{a}, do(a, s))$$

where ϵ denotes the empty sequence. Note that if along $\vec{a}$ the program becomes $\bot$ then $next^*$ returns $\bot$ as well.

We define the set $\mathcal{RR}(\delta, s)$ of (partial) *runs* of a program δ in a situation s as the sequences of actions that can be produced by executing δ from s:[4]

$$\mathcal{RR}(\delta, s) = \{\vec{a} \mid next^*(\delta, \vec{a}, s) \neq \bot\}$$

Note that if $\vec{a} \in \mathcal{RR}(\delta, s)$, then all prefixes of $\vec{a}$ are in $\mathcal{RR}(\delta, s)$ as well.

We define the set $\mathcal{CR}(\delta, s)$ of *complete runs* of a program δ in a situation s as the sequences of actions that can be produced by executing δ from s until a *Final* configuration is reached:

$$\mathcal{CR}(\delta, s) = \{\vec{a} \mid Final(next^*(\delta, \vec{a}, s), do(\vec{a}, s))\}$$

We define the set $\mathcal{GR}(\delta, s)$ of *good runs* of a program δ in a situation s as the sequences of actions that can be produced by executing δ from s which can be *extended* until a *Final* configuration is reached:

$$\mathcal{GR}(\delta, s) = \{\vec{a} \mid \exists \vec{b}.Final(next^*(\delta, \vec{a}\vec{b}, s), do(\vec{a}\vec{b}, s))\}$$

It is easy to see that $\mathcal{CR}(\delta, s) \subseteq \mathcal{GR}(\delta, s) \subseteq \mathcal{RR}(\delta, s)$, i.e., complete runs are good runs, and good runs are indeed runs. Moreover, $\mathcal{CR}(\delta, s) = \mathcal{CR}(\delta', s)$ implies $\mathcal{GR}(\delta, s) = \mathcal{GR}(\delta', s)$, i.e., if two programs in a situation have the same complete runs, then they also have the same good runs; however they may still differ in their sets of non-good runs, since $\mathcal{CR}(\delta, s) = \mathcal{CR}(\delta', s)$ does not imply $\mathcal{RR}(\delta, s) = \mathcal{RR}(\delta', s)$. We say that a program δ in s is *non-blocking* iff $\mathcal{RR}(\delta, s) = \mathcal{GR}(\delta, s)$, i.e., if all runs of the program δ in s can be extended to runs that reach a *Final* configuration.

The search construct. We can add to the language a search construct Σ, as in [De Giacomo *et al.*, 1998]:

$$next(\Sigma(\delta), a, s) = \begin{cases} \Sigma(next(\delta, a, s)) \text{ if there exists } \vec{a} \text{ s.t.} \\ \quad Final(next^*(\delta, a\vec{a}, s)) \\ \bot \text{ otherwise} \end{cases}$$

$$Final(\Sigma(\delta), s) \equiv Final(\delta, s).$$

Intuitively, $next(\Sigma(\delta), a, s)$ does lookahead to ensure that action a is in a good run of δ in s, otherwise it returns $\bot$.

Notice that: *(i)* $\mathcal{RR}(\Sigma(\delta), s) = \mathcal{GR}(\Sigma(\delta), s)$, i.e., under the search construct all programs are non-blocking; *(ii)* $\mathcal{RR}(\Sigma(\delta), s) = \mathcal{GR}(\delta, s)$, i.e., $\Sigma(\delta)$ produces exactly the good runs of δ; *(iii)* $\mathcal{CR}(\Sigma(\delta), s) = \mathcal{CR}(\delta, s)$, i.e., $\Sigma(\delta)$ and δ produce exactly the same set of complete runs. Thus $\Sigma(\delta)$ *trims* the behavior of δ by eliminating all those runs that do not lead to a *Final* configuration.

Note also that if a program is *non-blocking* in s, then $\mathcal{RR}(\Sigma(\delta), s) = \mathcal{RR}(\delta, s)$, in which case there is no point in using the search construct. Finally, we have that: $\mathcal{CR}(\delta, s) = \mathcal{CR}(\delta', s)$ implies $\mathcal{RR}(\Sigma(\delta), s) = \mathcal{RR}(\Sigma(\delta'), s)$, i.e., if two programs have the same complete runs, then under the search construct they have exactly the same runs.

4 Supervision

Let us assume that we have two agents: an agent B with behavior represented by the program δ_B and a supervisor S with behavior represented by δ_S. While both are represented by programs, the roles of the two agents are quite distinct. The first is an agent B that acts freely within its space of deliberation represented by δ_B. The second, S, is supervising B so that as B acts, it remains within the behavior permitted by S. This role makes the program δ_S act as a specification of allowed behaviors for agent B.

Note that, because of these different roles, one may want to assume that all configurations generated by (δ_S, s) are $Final$, so that we leave B unconstrained on when it may terminate. This amounts to requiring the following property to hold: $\mathcal{CR}(\delta_S, s) = \mathcal{GR}(\delta_S, s) = \mathcal{RR}(\delta_S, s)$. While reasonable, for the technical development below, we do not need to rely on this assumption.

The behavior of B under the supervision of S is constrained so that at any point B can execute an action in its original behavior, only if such an action is also permitted in

[3] Notice that such sequences of actions have to be axiomatized in second-order logic, similarly to situations (with UNA and domain closure). As a short cut they could also be characterized directly in terms of "difference" between situations.

[4] Here and in what follows, we use set notation for readability; if we wanted to be very formal, we could introduce $\mathcal{RR}$ as a defined predicate, and similarly for $\mathcal{CR}$, etc.

S's behavior. Using the synchronous concurrency operator, this can be expressed simply as:

$$\delta_B \ \& \ \delta_S.$$

Note that unless $\delta_B \ \& \ \delta_S$ happens to be non-blocking, it may get stuck in dead end configurations. To avoid this, we need to apply the search construct, getting $\Sigma(\delta_B \ \& \ \delta_S)$. In general, the use of the search construct to avoid blocking, is always needed in the development below.

We can use the example programs presented earlier to illustrate. The execution of δ_{B1} under the supervision of δ_{S1} is simply $\delta_{B1} \ \& \ \delta_{S1}$ (assuming all actions are controllable). It is straightforward to show that the resulting behavior is to repeatedly pick an available object and use it as long as one likes, breaking it at most once, and repairing it whenever it breaks, before discarding it. It can be shown that the set of partial/complete runs of $\delta_{B1} \ \& \ \delta_{S1}$ is exactly that of:

$$\begin{array}{l} [\pi\, x.\mathit{Available}(x)?; \\ \quad \mathit{use}(x)^*; \\ \quad [\mathit{nil} \mid (\mathit{break}(x); \mathit{repair}(x); \mathit{use}(x)^*)]; \\ \quad \mathit{discard}(x)]^* \end{array}$$

Uncontrollable actions. In the above, we implicitly assumed that all actions of agent B could be controlled by the supervisor S. This is often too strong an assumption, e.g. once we let a child out in a garden after rain, there is nothing we can do to prevent her/him from getting muddy. We now want to deal with such cases.

Following [Wonham and Ramadge, 1987], we distinguish between actions that are *uncontrollable* by the supervisor and actions that are *controllable*. The supervisor can block the execution of the controllable actions but cannot prevent the supervised agent from executing the uncontrollable ones.

To characterize the uncontrollable actions in the situation calculus, we use a special fluent $A_u(a_u, s)$, which we call an *action filter*, that expresses that action a_u is uncontrollable in situation s. Notice that, differently from the Wonham and Ramadge work, we allow controllability to be *context dependent* by allowing an arbitrary specification of the fluent $A_u(a_u, s)$ in the situation calculus.

While we would like the supervisor S to constrain agent B so that $\delta_B \ \& \ \delta_S$ is executed, in reality, since S cannot prevent uncontrollable actions, S can only constrain B on the controllable actions. When this is sufficient, we say that the supervisor is "effective". Technically, following again Wonham and Ramadge's ideas, this can be captured by saying that the *supervision by δ_S is effective for δ_B in situation s* iff:

$$\begin{array}{l} \forall \vec{a} a_u.\vec{a} \in \mathcal{GR}(\delta_B \ \& \ \delta_S, s) \text{ and } A_u(a_u, do(\vec{a}, s)) \text{ implies} \\ \quad \text{if } \vec{a}a_u \in \mathcal{GR}(\delta_B, s) \text{ then } \vec{a}a_u \in \mathcal{GR}(\delta_S, s). \end{array}$$

What this says is that if we postfix a good run $\vec{a}$ for both B and S with an uncontrollable action a_u that is good for B, then this uncontrollable action a_u must also be good for S. By the way, notice that $\vec{a}a_u \in \mathcal{GR}(\delta_B, s)$ and $\vec{a}a_u \in \mathcal{GR}(\delta_S, s)$ together imply that $\vec{a}a_u \in \mathcal{GR}(\delta_B \ \& \ \delta_S, s)$.

What about if such a property does not hold? We can take two orthogonal approaches: *(i)* relax δ_S so that it places no constraints on the uncontrollable actions; *(ii)* require that δ_S be indeed enforced, but disallow all those runs that prevent δ_S from being effective. We look at both approaches below.

Relaxed supervision. To define relaxed supervision we first need to introduce two operations on programs: *projection* and, based on it, *relaxation*. The *projection operation* takes a program and an *action filter* A_u, and projects all the actions that satisfy the action filter (e.g., are uncontrollable), out of the execution. To do this, projection substitutes each occurrence of an atomic action term α_i by a conditional statement that replaces it with the trivial test `true`? when $A_u(\alpha_i)$ holds in the current situation, that is:

$$\begin{array}{l} pj(\delta, A_u) = \delta^{\alpha_i}_{\textbf{if } A_u(\alpha_i) \textbf{ then } \texttt{true?} \textbf{ else } \alpha_i} \\ \text{for every occurrence of an action term } \alpha_i \text{ in } \delta. \end{array}$$

(Recall that such a test does not perform any transition in our variant of ConGolog.)

The *relaxation operation* on δ wrt $A_u(a, s)$ is as follows:

$$\mathit{rl}(\delta, A_u) = pj(\delta, A_u) \| (\pi a.A_u(a)?; a)^*.$$

In other words, we project out the actions in A_u from δ and run the resulting program concurrently with one that picks (uncontrollable) actions filtered by A_u and executes them. The resulting program no longer constrains the occurrence of actions from A_u in any way. In fact, notice that the remaining program of $(\pi a.A_u(a)?; a)^*$ after the execution of an (uncontrollable) filtered action is $(\pi a.A_u(a)?; a)^*$ itself, and that such a program is always *Final*.

Now we are ready to define *relaxed supervision*. Let us consider a supervisor S with behavior δ_S for agent B with behavior δ_B. Let the action filter $A_u(a_u, s)$ specify the uncontrollable actions. Then the *relaxed supervision* of S (for $A_u(a_u, s)$) in s is the relaxation of δ_S so as that it allows every uncontrollable action, namely: $\mathit{rl}(\delta_S, A_u)$. So we can characterize the behavior of B under the relaxed supervision of S as:

$$\delta_B \ \& \ \mathit{rl}(\delta_S, A_u).$$

The following properties are immediate consequences of the definitions:

Proposition 3 *The relaxed supervision $\mathit{rl}(\delta_S, A_u)$ is effective for δ_B in situation s.*

Proposition 4 $\mathcal{CR}(\delta_B \ \& \ \delta_S, s) \subseteq \mathcal{CR}(\delta_B \ \& \ \mathit{rl}(\delta_S, A_u), s)$.

Proposition 5 *If $\mathcal{CR}(\delta_B \ \& \ \mathit{rl}(\delta_S, A_u), s) \subseteq \mathcal{CR}(\delta_B \ \& \ \delta_S, s)$, then δ_S is effective for δ_B in situation s.*

Notice that, the first one is what we wanted. But the second one says that $\mathit{rl}(\delta_S, A_u)$ may indeed by more permissive than δ_S: some complete runs that are disallowed in δ_S may be permitted by its relaxation $\mathit{rl}(\delta_S, A_u)$. This is not always acceptable. The last one, says that when the converse of Proposition 4 holds, we have that the original supervision δ_S is indeed effective for δ_B in situation s. Notice however that even if δ_S effective for δ_B in situation s, it may still be the case that $\mathcal{CR}(\delta_B \ \& \ \mathit{rl}(\delta_S, A_u), s) \subset \mathcal{CR}(\delta_B \ \& \ \delta_S, s)$.

Maximal permissive supervisor. Next we study a more conservative approach: we require the supervision δ_S to be fulfilled, and for getting effectiveness we restrict it further. Interestingly, we show that there is a single maximal way of restricting the supervisor S so that it both fulfills δ_S and becomes effective. We call the resulting supervisor the *maximal permissive supervisor*.

We start by introducing a new abstract program construct $\mathbf{set}(E)$ taking as argument a possibly infinite set E of sequences of actions, with $next$ and *Final* defined as follows:

$$next(\mathbf{set}(E), a, s) = \begin{cases} \mathbf{set}(E') \text{ with } E' = \{\vec{a} \mid a\vec{a} \in E\} \\ \qquad \text{if } E' \neq \emptyset \\ \bot \text{ if } E' = \emptyset \end{cases}$$

$$\mathit{Final}(\mathbf{set}(E), s) \equiv (\epsilon \in E)$$

Thus $\mathbf{set}(E)$ can be executed to produce any of the sequences of actions in E.

Notice that for every program δ and situation s, we can define $E_\delta = \mathcal{CR}(\delta, s)$ such that $\mathcal{CR}(\mathbf{set}(E_\delta), s) = \mathcal{CR}(\delta, s)$. The converse does not hold in general, i.e., there are abstract programs $\mathbf{set}(E)$ such that for all programs δ, not involving the $\mathbf{set}(\cdot)$ construct, $\mathcal{CR}(\mathbf{set}(E_\delta), s) \neq \mathcal{CR}(\delta, s)$. That is, the syntactic restrictions in ConGolog may not allow us to represent some possible sets of sequences of actions.

With the $\mathbf{set}(E)$ construct at hand, following [Wonham and Ramadge, 1987], we may define the *maximal permissive supervisor* $mps(\delta_B, \delta_S, s)$ of B with behavior δ_B by S with behavior δ_S in situation s, as:

$$\begin{array}{l} mps(\delta_B, \delta_S, s) = \mathbf{set}(\bigcup_{E \in \mathcal{E}} E) \text{ where} \\ \qquad \mathcal{E} = \{E \mid E \subseteq \mathcal{CR}(\delta_B \;\&\; \delta_S, s) \\ \qquad\qquad \text{and } \mathbf{set}(E) \textit{ is effective for } \delta_B \textit{ in } s\} \end{array}$$

Intuitively mps denotes the maximal set of runs that are effectively allowable by a supervisor that fulfills the specification δ_S, and which can be left to the arbitrary decisions of the agent δ_B on the non-controllable actions. A quite interesting result is that, even in the general setting we are presenting, such a maximally permissive supervisor always *exists* and is *unique*. Indeed, we can show:

Theorem 6 *For the maximal permissive supervisor $mps(\delta_B, \delta_S, s)$ the following properties hold:*

1. *$mps(\delta_B, \delta_S, s)$ always exists and is unique;*
2. *$mps(\delta_B, \delta_S, s)$ is an effective supervisor for δ_B in s;*
3. *For every possible effective supervisor $\hat{\delta}_S$ for δ_B in s such that $\mathcal{CR}(\delta_B \;\&\; \hat{\delta}_S, s) \subseteq \mathcal{CR}(\delta_B \;\&\; \delta_S, s)$, we have that $\mathcal{CR}(\delta_B \;\&\; \hat{\delta}_S, s) \subseteq \mathcal{CR}(\delta_B \;\&\; mps(\delta_B, \delta_S, s), s)$.*

Proof: We prove the three claims separately.

Claim 1 follows directly from the fact $\mathbf{set}(\emptyset)$ satisfies the conditions to be included in $mps(\delta_B, \delta_S, s)$.

Claim 3 also follows immediately from the definition of $mps(\delta_B, \delta_S, s)$, by recalling that $\mathcal{CR}(\delta_B \;\&\; \hat{\delta}_S, s) = \mathcal{CR}(\delta_B \;\&\; \mathbf{set}(E_{\hat{\delta}_S}), s)$.

For *Claim 2*, it suffices to show that $\forall \vec{a} a_u . \vec{a} \in \mathcal{GR}(\delta_B \;\&\; mps(\delta_B, \delta_S, s), s)$ and $A_u(a_u, do(\vec{a}, s))$ we have that if $\vec{a} a_u \in \mathcal{GR}(\delta_B, s)$ then $\vec{a} a_u \in \mathcal{GR}(mps(\delta_B, \delta_S, s), s)$. Indeed, if $\vec{a} \in \mathcal{GR}(\delta_B \;\&\; mps(\delta_B, \delta_S, s), s)$ then there is an effective supervisor $\mathbf{set}(E)$ such that $\vec{a} \in \mathcal{GR}(\delta_B \;\&\; \mathbf{set}(E), \delta_S, s), s)$. $\mathbf{set}(E)$ being effective for δ_B in s, if $\vec{a} a_u \in \mathcal{GR}(\delta_B, s)$ then $\vec{a} a_u \in \mathcal{GR}(\mathbf{set}(E), s)$, but then $\vec{a} a_u \in \mathcal{GR}(mps(\delta_B, \delta_S, s), s)$. □

We can illustrate using our example programs. If we assume that the $break$ action is uncontrollable (and the others are controllable), the supervisor $S1$ can only ensure that its constraints are satisfied if it forces $B1$ to discard an object as soon as it is broken and repaired. This is what we get as maximal permissive supervisor $mps(\delta_{B1}, \delta_{S1}, S_0)$, whose set of partial/complete runs can be shown to be exactly that of:

$$\begin{array}{l} [\pi\, x.Available(x)?; \\ \qquad use(x)^*; \\ \qquad [nil \mid (break(x); repair(x))]; \\ \qquad discard(x)]^* \end{array}$$

By the way, notice that $(\delta_{B1} \;\&\; rl(\delta_{S1}, A_u))$ instead is completely ineffective since it has exactly the runs as δ_{B1}.

Unfortunately, in general, $mps(\delta_B, \delta_S, s)$ requires the use of the abstract program construct $\mathbf{set}(E)$, which can be expressed directly in ConGolog only if E is finite.[5] For this reason the above characterization remains essentially mathematical. So next, we develop a new construct for execution of programs under maximal permissive supervision, which is indeed realizable.

Maximal permissive supervised execution. To capture the notion of maximal permissive execution of agent B with behavior δ_B under the supervision of S with behavior δ_S in situation s, we introduce a special version of the synchronous concurrency construct that takes into account the fact the some actions are uncontrollable. Without loss of generality, we assume that δ_B and δ_S both start with a common controllable action (if not, it is trivial to add a dummy action in front of both so as to fullfil the requirement). Then, we characterize the construct through $next$ and *Final* as follows:

$$next(\delta_B \;\&_{A_u}\; \delta_S, a, s) =$$

$$\begin{cases} \bot \text{ if } \neg A_u(a, s) \text{ and } \exists \vec{a_u} . A_u(\vec{a_u}, do(a, s)) \text{ s.t.} \\ next^*(\Sigma(\delta_B), a\vec{a_u}, s) \neq \bot \text{ and } next^*(\Sigma(\delta_S), a\vec{a_u}, s) = \bot \\ \bot \text{ if } next(\delta_B, a, s) = \bot \text{ or } next(\delta_S, a, s) = \bot \\ next(\delta_B, a, s) \;\&_{A_u}\; next(\delta_S, a, s) \quad \text{otherwise} \end{cases}$$

Here $A_u(\vec{a_u}, s)$ is inductively defined on the length of $\vec{a_u}$ as the smallest predicate such that: *(i)* $A_u(\epsilon, s) \equiv \mathtt{true}$; *(ii)* $A_u(a_u \vec{a_u}, s) \equiv A_u(a_u, s) \wedge A_u(\vec{a_u}, do(a_u, s))$.

Final for the new construct is as follows:

$$\mathit{Final}(\delta_B \;\&_{A_u}\; \delta_S, s) \equiv \mathit{Final}(\delta_B, s) \wedge \mathit{Final}(\delta_S, s).$$

This new construct captures exactly the maximal permissive supervisor; indeed the theorem below shows the *correctness of maximal permissive supervised execution*:

Theorem 7

$$\mathcal{CR}(\delta_B \;\&_{A_u}\; \delta_S, s) = \mathcal{CR}(\delta_B \;\&\; mps(\delta_B, \delta_S, s), s).$$

Proof: We start by showing:

$$\mathcal{CR}(\delta_B \;\&_{A_u}\; \delta_S, s) \subseteq \mathcal{CR}(\delta_B \;\&\; mps(\delta_B, \delta_S, s), s).$$

It suffices to show that $\delta_B \;\&_{A_u}\; \delta_S$ is effective for δ_B in s. Indeed, if this is the case, by considering that $\delta_B \;\&\; mps(\delta_B, \delta_S, s)$ is the largest effective supervisor for δ_B in s, and that $\mathcal{RR}(\delta_B \;\&\; (\delta_B \;\&_{A_u}\; \delta_S), s) = \mathcal{RR}(\delta_B \;\&_{A_u}\; \delta_S, s)$, we get the thesis.

[5] Note that the object domain may be uncountable in general, hence not even an infinitary ConGolog program could capture $\mathbf{set}(E)$ in general.

So we have to show that: $\forall \vec{a} a_u . \vec{a} \in \mathcal{GR}(\delta_B \mathbin{\&_{A_u}} \delta_S, s)$ and $A_u(a_u, do(\vec{a}, s))$ we have that if $\vec{a} a_u \in \mathcal{GR}(\delta_B, s)$ then $\vec{a} a_u \in \mathcal{GR}(\delta_B \mathbin{\&_{A_u}} \delta_S, s)$.

Since, wlog we assume that δ_B and δ_S started with a common controllable action, we can write $\vec{a} = \vec{a'} a_c \vec{a_u}$, where $\neg A_u(a_c, do(\vec{a'}, s))$ and $A_u(\vec{a_u}, do(\vec{a'} a_c, s))$ holds. Let $\delta'_B = next^*(\delta_B, \vec{a'}, s)$, $\delta'_S = next^*(\delta_S, \vec{a'}, s)$, and $s' = do(\vec{a'}, s)$. By the fact that $\vec{a'} a_c \vec{a_u} \in \mathcal{GR}(\delta_B \mathbin{\&_{A_u}} \delta_S, s)$ we know that $next(\delta'_B \mathbin{\&_{A_u}} \delta'_S, do(a_c, s')) \neq \bot$. But then, by de definition of $next$, we have that for all $\vec{b_u}$ such that $A_u(\vec{b_u}, s')$ if $\vec{b_u} \in \mathcal{GR}(\delta'_B, do(a_c, s'))$ then $\vec{b_u} \in \mathcal{GR}(\delta'_S, do(a_c, s'))$. In particular this holds for $\vec{b_u} = \vec{a_u} a_u$. Hence we have that if $\vec{a} a_u \in \mathcal{GR}(\delta_B, s)$ then $\vec{a} a_u \in \mathcal{GR}(\delta_S, s)$.

Next we prove:

$$\mathcal{CR}(\delta_B \mathbin{\&} mps(\delta_B, \delta_S, s), s) \subseteq \mathcal{CR}(\delta_B \mathbin{\&_{A_u}} \delta_S, s).$$

Suppose not. Then there exist a complete run $\vec{a}$ such that $\vec{a} \in \mathcal{CR}(\delta_B \mathbin{\&} mps(\delta_B, \delta_S, s), s)$ but $\vec{a} \notin \mathcal{CR}(\delta_B \mathbin{\&_{A_u}} \delta_S, s)$.

As an aside, notice that $\vec{a} \in \mathcal{CR}(\delta, s)$ then $\vec{a} \in \mathcal{GR}(\delta, s)$ and for all prefixes $\vec{a'}$ such that $\vec{a'}\vec{b} = \vec{a}$ we have $\vec{a'} \in \mathcal{GR}(\delta, s)$.

Hence, let $\vec{a'} = \vec{a''} a$ such that $\vec{a'} \in \mathcal{GR}(\delta_B \mathbin{\&_{A_u}} \delta_S, s)$ but $\vec{a''} a \notin \mathcal{GR}(\delta_B \mathbin{\&_{A_u}} \delta_S, s)$, and let $\delta''_B = next^*(\delta''_B, \vec{a''}, s)$, $\delta''_S = next^*(\delta_S, \vec{a''}, s)$, and $s' = do(\vec{a''}, s)$.

Since $\vec{a''} a \notin \mathcal{GR}(\delta_B \mathbin{\&_{A_u}} \delta_S, s)$, it must be the case that $next(\delta''_B \mathbin{\&_{A_u}} \delta''_S, a, s'') = \bot$. But then, considering that both $next(\delta''_B, a, s'') \neq \bot$ and $next(\delta''_S, a, s'') \neq \bot$, it must be the case that $\neg A_u(a, s'')$ and exists $\vec{b_u}$ such that $A_u(\vec{b_u}, do(a, s''))$, and $a\vec{b_u} \in \mathcal{GR}(\delta''_B, s'')$ but $a\vec{b_u} \notin \mathcal{GR}(\delta''_S, s'')$.

Notice that $\vec{b_u} \neq \epsilon$, since we have that $a \in \mathcal{GR}(\delta''_S, s'')$. So $\vec{b_u} = \vec{c_u} b_u \vec{d_u}$ with $a\vec{c_u} \in \mathcal{GR}(\delta''_S, s'')$ but $a\vec{c_u} b_u \notin \mathcal{GR}(\delta''_S, s'')$.

Now $\vec{a'} \in \mathcal{GR}(\delta_B \mathbin{\&} mps(\delta_B, \delta_S, s), s)$ and since $A_u(\vec{c_u} b_u, do(\vec{a'}, s))$, we have that $\vec{a'}\vec{c_u} b_u \in \mathcal{GR}(\delta_B \mathbin{\&} mps(\delta_B, \delta_S, s), s)$. Since, $mps(\delta_B, \delta_S, s)$ is effective for δ_B in s, we have that, if $\vec{a'}\vec{c'_u} b_u \in \mathcal{GR}(\delta_B, s)$ then $\vec{a'}\vec{c_u} b_u \in \mathcal{GR}(mps(\delta_B, \delta_S, s), s)$. This, by definition of $mps(\delta_B, \delta_S, s)$, implies $\vec{a'}\vec{c_u} b_u \in \mathcal{GR}(\delta_B \mathbin{\&} \delta_S, s)$, and hence, in turn, $\vec{a'}\vec{c_u} b_u \in \mathcal{GR}(\delta_S, s)$. Hence, we can conclude that $a\vec{c'_u} b_u \in \mathcal{GR}(\delta''_S, s'')$, getting a contradiction. $\square$

5 Conclusion

In this paper, we have investigated agent supervision in situation-determined ConGolog programs. Our account of maximal permissive supervisor builds on [Wonham and Ramadge, 1987]. However, Wonham and Ramage's work deals with finite state automata, while we handle infinite state systems in the context of the rich agent framework provided by the situation calculus and ConGolog. We used ConGolog as a representative of an unbounded-states process specification language, and it should be possible to adapt our account of supervision to other related languages. We considered a form of supervision that focuses on complete runs, i.e., runs that lead to *Final* configurations. We can ensure that an agent finds such executions by having it do lookahead/search. Also of interest is the case in which agents act boldly without necessarily performing search to get to *Final* configurations. In this case, we need to consider all partial runs, not just good ones. Note that this would actually yield the same result if we engineered the agent behavior such that all of its runs are good runs, i.e. if $\mathcal{RR}(\delta_B, s) = \mathcal{GR}(\delta_B, s)$, i.e., all configurations are final. In fact, one could define a closure construct $cl(\delta)$ that would make all configurations of δ final. Using this, one can apply our specification of the maximal permissive supervisor to this case as well if we replace $\delta_B \mathbin{\&} \delta_S$ by $cl(\delta_B \mathbin{\&} \delta_S)$ in the definition. Observe also, that under the assumption $\mathcal{RR}(\delta_B, s) = \mathcal{GR}(\delta_B, s)$, in $next(\delta_B \mathbin{\&_{A_u}} \delta_S, a, s)$ we no longer need to do the search $\Sigma(\delta_B)$ and $\Sigma(\delta_S)$ and can directly use δ_B and δ_S.

We conclude by mentioning that if the object domain is finite, then ConGolog programs assume only a finite number of possible configurations. In this case, we can take advantage of the finite state machinery that was originally proposed by Wonham and Ramage (generalizing it to deal with situation-dependent sets of controllable actions), and the recent work on translating ConGolog into finite state machines and back [Fritz *et al.*, 2008], to obtain a program that actually characterizes the maximally permissive supervisor. In this way, we can completely avoid doing search during execution. We leave an exploration of this notable case for future work.

Acknowledgments

We thank Murray Wonham for inspiring discussions on supremal controllable languages in finite state discrete event control, which actually made us look into agent supervision from a different and very fruitful point of view. We also thank the anonymous referees for their comments. We acknowledge the support of EU Project FP7-ICT ACSI (257593).

References

[Bertoli *et al.*, 2010] Piergiorgio Bertoli, Marco Pistore, and Paolo Traverso. Automated composition of web services via planning in asynchronous domains. *Artif. Intell.*, 174(3-4):316–361, 2010.

[De Giacomo *et al.*, 1998] Giuseppe De Giacomo, Raymond Reiter, and Mikhail Soutchanski. Execution monitoring of high-level robot programs. In *KR*, pages 453–465, 1998.

[De Giacomo *et al.*, 2000] Giuseppe De Giacomo, Yves Lespérance, and Hector J. Levesque. ConGolog, a concurrent programming language based on the situation calculus. *Artificial Intelligence*, 121(1–2):109–169, 2000.

[De Giacomo *et al.*, 2010] Giuseppe De Giacomo, Yves Lespérance, and Adrian R. Pearce. Situation calculus based programs for representing and reasoning about game structures. In *KR*, 2010.

[Demolombe and Hamon, 2002] Robert Demolombe and Erwan Hamon. What does it mean that an agent is performing a typical procedure? a formal definition in the situation calculus. In *AAMAS*, pages 905–911, 2002.

[Fritz and Gil, 2010] Christian Fritz and Yolanda Gil. Towards the integration of programming by demonstration and programming by instruction using Golog. In *PAIR*, 2010.

[Fritz and McIlraith, 2006] Christian Fritz and Sheila McIlraith. Decision-theoretic Golog with qualitative preferences. In *KR*, pages 153–163, June 2–5 2006.

[Fritz *et al.*, 2008] Christian Fritz, Jorge A. Baier, and Sheila A. McIlraith. ConGolog, sin trans: Compiling ConGolog into basic action theories for planning and beyond. In *KR*, pages 600–610, 2008.

[Lin *et al.*, 2008] Naiwen Lin, Ugur Kuter, and Evren Sirin. Web service composition with user preferences. In *ESWC*, pages 629–643, 2008.

[McIlraith and Son, 2002] S. McIlraith and T. Son. Adapting Golog for composition of semantic web services. In *KR*, pages 482–493, 2002.

[Reiter, 2001] Ray Reiter. *Knowledge in Action. Logical Foundations for Specifying and Implementing Dynamical Systems*. The MIT Press, 2001.

[Sardiña and De Giacomo, 2009] Sebastian Sardiña and Giuseppe De Giacomo. Composition of ConGolog programs. In *IJCAI*, pages 904–910, 2009.

[Sohrabi *et al.*, 2009] Shirin Sohrabi, Nataliya Prokoshyna, and Sheila A. McIlraith. Web service composition via the customization of Golog programs with user preferences. In *Conceptual Modeling: Foundations and Applications*, pages 319–334. Springer, 2009.

[Su, 2008] Jianwen Su. Special issue on semantic web services: Composition and analysis. *IEEE Data Eng. Bull.*, 31(3), 2008.

[Wonham and Ramadge, 1987] WM Wonham and PJ Ramadge. On the supremal controllable sub-language of a given language. *SIAM J Contr Optim*, 25(3):637659, 1987.

On the Use of Epistemic Ordering Functions as Decision Criteria for Automated and Assisted Belief Revision in SNePS
(Preliminary Report)

Ari I. Fogel and Stuart C. Shapiro
University at Buffalo, The State University of New York, Buffalo, NY
{arifogel,shapiro}@buffalo.edu

Abstract

We implement belief revision in SNePS based on a user-supplied epistemic ordering of propositions. We provide a decision procedure that performs revision completely automatically when given a well preorder. We also provide a decision procedure for revision that, when given a total preorder, simulates a well preorder by making a minimal number of queries to the user when multiple propositions within a minimally-inconsistent set are minimally-epistemically-entrenched. The first procedure uses $O(|\Sigma|)$ units of space, and completes within $O(|\Sigma|^2 \cdot s_{max})$ units of time, where Σ is the set of distinct minimally-inconsistent sets, and s_{max} is the number of propositions in the largest minimally-inconsistent set. The second procedure uses $O(|\Sigma|^2 \cdot s_{max}^2)$ space and $O(|\Sigma|^2 \cdot s_{max}^2)$ time. We demonstrate how our changes generalize previous techniques employed in SNePS.

1 Introduction

1.1 Belief Revision

Several varieties of belief revision have appeared in the literature over the years. AGM revision typically refers to the addition of a belief to a belief set, at the expense of its negation and any other beliefs supporting its negation [Alchourron *et al.*, 1985]. Removal of a belief and beliefs that support it is called *contraction*. Alternatively, revision can refer to the process of resolving inconsistencies in a contradictory knowledge base, or one *known to be inconsistent* [Martins and Shapiro, 1988]. This is accomplished by removing one or more beliefs responsible for the inconsistency, or *culprits*. This is the task with which we are concerned. In particular, we have devised a means of automatically resolving inconsistencies by discarding the least-preferred beliefs in a belief base, according to some *epistemic ordering* [Gärdenfors, 1988; Williams, 1994; Gärdenfors and Rott, 1995].

> The problem of belief revision is that logical considerations alone do not tell you which beliefs to give up, but this has to be decided by some other means. What makes things more complicated is that beliefs in a database have logical consequences. So when giving up a belief you have to decide as well which of its *consequences* to retain and which to retract... [Gärdenfors and Rott, 1995]

In later sections we will discuss in detail how to make a choice of belief(s) to retract when presented with an inconsistent belief set.

AGM Paradigm

In [Gärdenfors, 1982; Alchourron *et al.*, 1985], operators and rationality postulates for *theory change* are discussed. In general, any operator that satisfies those postulates may be thought of as an AGM operation.

Let $Cn(A)$ refer to the closure under logical consequence of a set of propositions A. A *theory* is defined to be a set of propositions closed under logical consequence. Thus for any set of propositions A, $Cn(A)$ is a theory. It is worth noting that theories are infinite sets. [Alchourron *et al.*, 1985] discusses operations that may be performed on theories. *Partial meet contraction and revision*. defined in [Alchourron *et al.*, 1985], satisfy all of the postulates for a rational contraction and revision operator, respectively.

Theory Change on Finite Bases

> It is widely accepted that agents, because of their limited resources, believe some but by no means all of the logical consequences of their beliefs. [Lakemeyer, 1991]

A major issue with the AGM paradigm it *tends to* operate on and produce infinite sets (theories). A more practical model would include operations to be performed on finite belief sets, or *belief bases*. Such operators would be useful in supporting computer-based implementations of revision systems [Williams, 1994].

It has been argued that The AGM paradigm uses a *coherentist* approach[1] [Gärdenfors, 1989], in that all beliefs require some sort of external justification. On the other hand, finite-base systems are said to use a foundationalist approach, wherein some beliefs indeed have their own epistemic standing, and others can be derived from them. SNePS, as we shall see, uses the finite-base foundationalist approach.

[1] It has also been argued otherwise [Hansson and Olsson, 1999]

Epistemic Entrenchment

Let us assume that the decision on which beliefs to retract from a belief base is made is based on the relative importance of each belief, which is called its degree of *epistemic entrenchment* [Gärdenfors, 1988]. Then we need an ordering $\leqslant$ with which to compare the entrenchment of individual beliefs. Beliefs that are less entrenched are preferentially discarded during revision over beliefs that are more entrenched. An epistemic entrenchment ordering is used to uniquely determine the result of AGM contraction. Such an ordering is a noncircular total preorder (that satisfies certain other postulates) on *all propositions*.

Ensconcements

Ensconcements, introduced in [Williams, 1994], consist of a set of forumlae together with a total preorder on that set. They can be used to construct epistemic entrenchment orderings, and determine theory base change operators.

Safe Contraction

In [Alchourron and Makinson, 1985], the operation *safe contraction* is introduced. Let $<$ be a non-circular relation over a belief set A. An element a of A is *safe* with respect to x iff a is not a minimal element of any minimal subset B of A such that $x \in Cn(B)$. Let A/x be the set of all elements of A that are safe with respect to x. Then the safe contraction of A by x, denoted $A \dot{-}_s x$, is defined to be $A \cap Cn(A/x)$.

Assumption-based Truth Maintenance Systems

In an assumption-based truth maintenance system (ATMS), the system keeps track of the assumptions (base beliefs) underlying each belief [de Kleer, 1986]. One of the roles of an conventional TMS is to keep the database contradiction-free. In an assumption-based ATMS, contradictions are removed as they are discovered. When a contradiction is detected in an ATMS, then there will be one or more minimally-inconsistent sets of assumptions underlying the contradiction. Such sets are called *no-goods*. [Martins and Shapiro, 1988] presented SNeBR, an early implementation of an ATMS that uses the logic of SNePS. In that paper, sets of assumptions supporting a belief are called *origin sets*. They correspond to *antecedents* of a *justification* from [de Kleer, 1986]. The focus of this paper is modifications to the modern version of SNeBR.

Kernel Contraction

In [Hansson, 1994], the operation *kernel contraction* is introduced. A *kernel set* $A \perp\!\!\!\perp \alpha$ is defined to be the set of all minimal subsets of A that imply α. A kernel set is like a set of *origin sets* from [Martins and Shapiro, 1988]. Let σ be an *incision function* for A. Then for all α, $\sigma(A \perp\!\!\!\perp \alpha) \subseteq \cup(A \perp\!\!\!\perp \alpha)$, and if $\emptyset \neq X \in A \perp\!\!\!\perp \alpha$, then $X \cap \sigma(A \perp\!\!\!\perp \alpha) \neq \emptyset$. The *kernel contraction* of A by α based on σ, denoted $A \sim_\sigma \alpha$, is equal to $A \backslash \sigma(A \perp\!\!\!\perp \alpha)$.

Prioritized Versus Non-Prioritized Belief Revision

> In the AGM model of belief revision [Alchourron *et al.*, 1985] …the input sentence is always accepted. This is clearly an unrealistic feature, and …several models of belief change have been proposed in which no absolute priority is assigned to the new information due to its novelty. …One way to construct non-prioritized belief revision is to base it on the following two-step process: First we decide whether to accept or reject the input. After that, if the input was accepted, it is incorporated into the belief state [Hansson, 1999].

Hansson goes on to describe several other models of nonprioritized belief revision, but they all have one unifiying feature distinguishing them from prioritized belief revision: the input, i.e. the RHS argument to the revision operator, is not always accepted. To reiterate: *Prioritized belief revision* is revision in which the proposition by which the set is revised is always present in the result (as long as it is not a contradiction). *Non-prioritized belief revision* is revision in which the RHS argument to the revision operator is not always present in the result (even if it is not a contradiction).

The closest approximation from Hansson's work to our work is the operation of *semi-revision* [Hansson, 1997]. Semi-revision is a type of non-prioritized belief revision that may be applied to belief bases.

1.2 SNePS

Description of the System

"SNePS is a logic-, frame-, and network- based knowledge representation, reasoning, and acting system. Its logic is based on Relevance Logic [Shapiro, 1992], a paraconsistent logic (in which a contradiction does not imply anything whatsoever) [Shapiro and Johnson, 2000]."

SNeRE, the SNePS Rational Engine, provides an acting system for SNePS-based agents, whose beliefs must change to keep up with a changing world. Of particular interest is the *believe* action, which is used to introduce beliefs that take priority over all other beliefs at the time of their introduction.

Belief Change in SNePS

Every belief in a SNePS knowledge base (which consists of a belief base and all currently-known derived propostions therefrom) has one or more *support sets*, each of which consists of an *origin tag* and an *origin set*. The origin tag will identify a belief as either being introduced as a hypothesis, or derived (note that it is possible for a belief to be both introduced as a hypothesis and derived from other beliefs). The origin set contains those *hypotheses* that were used to derive the belief. In the case of the origin tag denoting a hypothesis, the corresponding origin set would be a singleton set containing only the belief itself. The contents of the origin set of a derived belief are computed by the implemented rules of inference at the time the inference is drawn [Martins and Shapiro, 1988; Shapiro, 1992].

The representation of beliefs in SNePS lends itself well to the creation of processes for contraction and revision. Specifically, in order to contract a belief, one must merely remove at least one hypothesis from each of its origin sets. Similarly, prioritized revision by a belief b (where $\neg b$ is already believed) is accomplished by removing at least one belief from each origin set of $\neg b$. Non-prioritized belief revision under this paradigm is a bit more complicated. We discuss both types of revision in more detail in §2.

SNeBR

SNeBR, The SNePS Belief Revision subsystem, is responsible for resolving inconsistencies in the knowledge base as they are discovered. In the current release of SNePS (version 2.7.1), SNeBR is able to *automatically* resolve contradictions under a limited variety of circumstances [Shapiro and The SNePS Implementation Group, 2010, 76]. Otherwise "assisted culprit choosing" is performed, where the user must manually select culprits for removal. After belief revision is performed, the knowledge base might still be inconsistent, but every *known* derivation of an inconsistency has been eliminated.

2 New Belief Revision Algorithms

2.1 Problem Statement

Nonprioritized Belief Revision

Suppose we have a knowledge base that is not known to be inconsistent, and suppose that at some point we add a contradictory belief to that knowledge base. Either that new belief directly contradicts an existing belief, or we derive a belief that directly contradicts an existing one as a result of performing forward and/or backward inference on the new belief. Now the knowledge base is known to be inconsistent. We will refer to the contradictory beliefs as p and $\neg p$

Since SNePS tags each belief with one or more origin sets, or sets of supporting hypotheses, we can identify the underlying beliefs that support each of the two contradictory beliefs. In the case where p and $\neg p$ each have one origin set, OS_p and $OS_{\neg p}$ respectively, we may resolve the contradiction by removing at least one hypothesis from $OS_p \cup OS_{\neg p}$. We shall refer to such a union as a *no-good*. If there are m origin sets for p, and n origin sets for $\neg p$, then there will be at most $m \times n$ distinct no-goods (some unions may be duplicates of others). To resolve a contradiction in this case, we must retract at least one hypothesis from each no-good (Sufficiency).

We wish to devise an algorithm that will select the hypotheses for removal from the set of no-goods. The first priority will be that the hypotheses selected should be minimally-epistemically-entrenched (Minimal Entrenchment) according to some total preorder $\leqslant$. Note that we are not referring strictly to an AGM entrenchment order, but to a total preorder on the set of hypotheses, without regard to the AGM postulates. The second priority will be not to remove any more hypotheses than are necessary in order to resolve the contradiction (Information Preservation), while still satisfying priority one.

Prioritized Belief Revision

The process of Prioritized Belief Revision in SNePS occurs when a contradiction is discovered after a belief is asserted explicitly using the *believe* act of SNeRE. The major difference here is that a subtle change is made to the entrenchment ordering $\leqslant$. If $\leqslant_{nonpri}$ is the ordering used for nonprioritized belief revision, then for prioritized belief revision we use an ordering $\leqslant_{pri}$ as follows:

Let P be the set of beliefs asserted by a *believe* action. Then
$\forall e_1, e_2[e_1 \in P \land e_2 \notin P \rightarrow \neg(e_1 \leqslant_{pri} e_2) \land e_2 \leqslant_{pri} e_1]$
$\forall e_1, e_2[e_1 \notin P \land e_2 \notin P \rightarrow (e_1 \leqslant_{pri} e_2 \leftrightarrow e_1 \leqslant_{nonpri} e_2)]$
$\forall e_1, e_2[e_1 \in P \land e_2 \in P \rightarrow (e_1 \leqslant_{pri} e_2 \leftrightarrow e_1 \leqslant_{nonpri} e_2)]$

That is, a proposition asserted by a *believe* action takes priority over any other proposition. When either both or neither propositions being compared have been assserted by the *believe* action, then we use the same ordering as we would for nonprioritized revision.

2.2 Common Requirements for a Rational Belief Revision Algorithm

Primary Requirements

The inputs to the algorithm are:

- A set of formulae Φ: the current belief base, which is known to be inconsistent
- A total preorder $\leqslant$ on Φ: an epistemic entrenchment ordering that can be used to compare the relative desirability of each belief in the current belief base
- Minimally-inconsistent sets of formulae $\sigma_1, \ldots, \sigma_n$, each of which is a subset of Φ: the no-goods
- A set $\Sigma = \{\sigma_1, \ldots, \sigma_n\}$: the set of all the no-goods

The algorithm should produce a set T that satisfies the following conditions:

$(EE_{SNePS}1)$ $\forall \sigma[\sigma \in \Sigma \rightarrow \exists \tau[\tau \in (T \cap \sigma)]$ (Sufficiency)

$(EE_{SNePS}2)$ $\forall \tau[\tau \in T \rightarrow \exists \sigma[\sigma \in \Sigma \land \tau \in \sigma \land \forall w[w \in \sigma \rightarrow \tau \leqslant w]]]$ (Minimal Entrenchment)

$(EE_{SNePS}3)$ $\forall T'[T' \subset T \rightarrow \neg \forall \sigma[\sigma \in \Sigma \rightarrow \exists \tau[\tau \in (T' \cap \sigma)]]]$ (Information Preservation)

Condition $(EE_{SNePS}1)$ states that T contains at least one formula from each set in Σ. Condition $(EE_{SNePS}2)$ states that every formula in T is a minimally-entrenched formula of some set in Σ. Condition $(EE_{SNePS}3)$ states that if any formula is removed from T, then Condition $(EE_{SNePS}1)$ will no longer hold. In addition to the above conditions, our algorithm must terminate on all possible inputs, i.e. it must be a decision procedure.

Supplementary Requirement

In any case where queries must be made of the user in order to determine the relative epistemic ordering of propositions, the number of such queries must be kept to a minimum.

2.3 Implementation

We present algorithms to solve the problem as stated:
Where we refer to $\leqslant$ below, we are using the *prioritized* entrenchment ordering from §2.1. In the case of nonprioritized revision we may assume that $P = \varnothing$

Using a well preorder

Let $\leq_{\leqslant}$ be the output of a function f whose input is a total preorder $\leqslant$, such that $\leq_{\leqslant} \subseteq \leqslant$ The idea is that f creates the well preorder $\leq_{\leqslant}$ from $\leqslant$ by removing some pairs from the total preorder $\leqslant$. Note that in the case where $\leqslant$ is already a well preorder, $\leq_{\leqslant} = \leqslant$. Then we may use Algorithm 1 to solve the problem.

Algorithm 1 Algorithm to compute T given a well preorder

```
Input: $\Sigma, \leq_{\leqslant}$
Output: $T$
 1: $T \Leftarrow \emptyset$
 2: for all $(\sigma \in \Sigma)$ do
 3:     Move minimally entrenched belief in $\sigma$ to first position
        in $\sigma$, using $\leq_{\leqslant}$ as a comparator
 4: end for
 5: Sort elements of $\Sigma$ into descending order of the values of
    the first element in each $\sigma$ using $\leq_{\leqslant}$ as a comparator
 6: $AddLoop$ :
 7: while $(\Sigma \neq \emptyset)$ do
 8:     $currentCulprit \Leftarrow \sigma_{1_1}$
 9:     $T \Leftarrow T \cup \{currentCulprit\}$
10:     $DeleteLoop$ :
11:     for all $(\sigma_{current} \in \Sigma)$ do
12:         if $(currentCulprit \in \sigma_{current})$ then
13:             $\Sigma \Leftarrow \Sigma \setminus \sigma_{current}$
14:         end if
15:     end for
16: end while
17: return $T$
```

Using a total preorder

Unfortunately it is easy to conceive of a situation in which the supplied entrenchment ordering is a total preorder, but not a well preorder. For instance, let us say that, when reasoning about a changing world, propositional fluents (propositions that are only true of a specific time or situation) are abandoned over non-fluent propositions. It is not clear then how we should rank two distinct propositional fluents, nor how to rank two distinct non-fluent propositions. If we can arbitrarily specify a well preorder that is a subset of the total preorder we are given, then algorithm 1 will be suitable. Otherwise, we can simulate a well order $\leq$ through an iterative construction by querying the user for the unique minimally-entrenched proposition of a particular set of propositions at appropriate times in the belief-revision process. Algorithm 2 accomplishes just this.

Algorithm 2 Algorithm to compute T given a total preorder

```
Input: $\Sigma, \leqslant$
Output: $T$
$T \Leftarrow \emptyset$
$MainLoop$:
loop
    $ListLoop$:
    for all $(\sigma_i \in \Sigma, 1 \leqslant i \leqslant |\Sigma|)$ do
        Make a list $l_{\sigma_i}$ of all minimally-entrenched propo-
        sitions, i.e. propositions that are not strictly more
        entrenched than any other, among those in $\sigma_i$, using
        $\leqslant$ as a comparator.
    end for
    $RemoveLoop$:
    for all $(\sigma_i \in \Sigma, 1 \leqslant i \leqslant |\Sigma|)$ do
        if (According to $l_{\sigma_i}$, $\sigma$ has exactly one minimally-
        entrenched proposition $p$ AND the other proposi-
        tions in $\sigma_i$ are not minimally-entrenched in any
        other no-good via an $l_{\sigma_j}$,$(1 \leqslant j \leqslant |\Sigma|, i \neq j)$) then
            $T \Leftarrow T \cup \{p\}$
            for all $(\sigma_{current} \in \Sigma)$ do
                if $(p \in \sigma_{current})$ then
                    $\Sigma \Leftarrow \Sigma \setminus \sigma_{current}$
                end if
            end for
            if $(\Sigma = \emptyset)$ then
                return $T$
            end if
        end if
    end for
    $ModifyLoop$:
    for all $(\sigma \in \Sigma)$ do
        if ($\sigma$ has multiple minimally-entrenched proposi-
        tions) then
            query which proposition $l$ of the minimally-
            entrenched propostions is least desired.
            Modify $\leqslant$ so that $l$ is strictly less entrenched than
            those other propositions.
            break out of $ModifyLoop$
        end if
    end for
end loop
```

Characterization

These algorithms perform an operation similar to *incision functions* [Hansson, 1994], since they select one or more propositions to be removed from each minimally-inconsistent set. Their output seems analogous to $\sigma(\Phi \perp\!\!\!\perp (p \wedge \neg p))$, where σ is the incision function, $\perp\!\!\!\perp$ is the kernel-set operator from [Hansson, 1994], and p is a proposition. But we are actually incising Σ, the set of *known* no-goods. The known no-goods are of course a subset of all no-goods, i.e. $\Sigma \subseteq \Phi \perp\!\!\!\perp (p \wedge \neg p)$. This happens because SNeBR resolves contradictions as soon as they are discovered, rather than performing inference first to discover all possible sources of contradictions.

The type of contraction eventually performed is similar to safe contraction [Alchourron and Makinson, 1985], except that there are fewer restrictions on our epistemic ordering.

3 Analysis of Algorithm 1

3.1 Proofs of Satisfaction of Requirements by Algorithm 1

We show that Algorithm 1 satisfies the requirements established in section 2:

$(EE_{SNePS}1)$ **(Sufficiency)**

During each iteration of *AddLoop* an element τ is added to T from some $\sigma \in \Sigma$. Then each set $\sigma \in \Sigma$ containing τ is removed from Σ. The process is repeated until Σ is empty. Therefore each removed set σ in Σ contains some τ in T (Note that each σ will be removed from Σ by the end of the process). So $\forall\sigma[\sigma \in \Sigma \rightarrow \exists\tau[\tau \in (T \cap \sigma)]$. Q.E.D.

$(EE_{SNePS}2)$ **(Minimal Entrenchment)**

From lines 8-9, we see that T is comprised solely of first elements of sets in Σ. And from lines 2-4, we see that those first elements are all minimal under $\leq_{\leqslant}$ relative to the other elements in each set. Since $\forall e_1, e_2, \leqslant [e_1 \dot{\leq}_{\leqslant} e_2 \rightarrow e_1 \leqslant e_2]$, those first elements are minimal under $\leqslant$ as well. That is, $\forall \tau[\tau \in T \rightarrow \exists \sigma[\sigma \in \Sigma \wedge \tau \in \sigma \wedge \forall w[w \in \sigma \rightarrow \tau \leqslant w]]]$. Q.E.D.

$(EE_{SNePS}3)$ **(Information Preservation)**
From the previous proof we see that during each iteration of *AddLoop*, we are guaranteed that at least one set σ containing the current culprit is removed from Σ. And we know that the current culprit for that iteration is minimally-entrenched in σ. We also know from $(EE_{SNePS}2)$ that each subsequently chosen culprit will be minimally entrenched in some set. From lines 2-5 and *AddLoop*, we know that subsequently chosen culprits will be less entrenched than the current culprit. From lines 2-5, we also see that all the other elements in σ have higher entrenchment than the current culprit. Therefore subsequent culprits cannot be elements in σ. So, they cannot be used to eliminate σ. Obviously, previous culprits were also not members of σ. Therefore, if we exclude the current culprit from T, then there will be a set in Σ that does not contain any element of T. That is,
$\forall T'[T' \subset T \rightarrow \exists \sigma[\sigma \in \Sigma \wedge \neg \exists \tau[\tau \in (T' \cap \sigma)]]]$
$\therefore \forall T'[T' \subset T \rightarrow \exists \sigma[\neg\neg(\sigma \in \Sigma \wedge \neg \exists \tau[\tau \in (T' \cap \sigma))]]]$
$\therefore \forall T'[T' \subset T \rightarrow \exists \sigma[\neg(\neg(\sigma \in \Sigma) \vee \exists \tau[\tau \in (T' \cap \sigma))]]]$
$\therefore \forall T'[T' \subset T \rightarrow \exists \sigma[\neg(\sigma \in \Sigma \rightarrow \exists \tau[\tau \in (T' \cap \sigma))]]$
$\therefore \forall T'[T' \subset T \rightarrow \neg \forall \sigma[\sigma \in \Sigma \rightarrow \exists \tau[\tau \in (T' \cap \sigma)]]]$ Q.E.D.

Decidability
We see that *DeleteLoop* is executed once for each element in Σ, which is a finite set. So it always terminates. We see that *AddLoop* terminates when Σ is empty. And from lines 8 and 13 we see that at least one set is removed from Σ during each iteration of *AddLoop*. So *AddLoop* always terminates. Lines 2-4 involve finding a minimum element, which is a decision procedure. Line 5 performs sorting, which is also a decision procedure. Since every portion of Algorithm 1 always terminates, it is a decision procedure. Q.E.D.

Supplementary Requirement
Algorithm 1 is a fully-automated procedure that makes no queries of the user. Q.E.D.

3.2 Complexity of Algorithm 1

Space Complexity
Algorithm 1 can be run completely in-place, i.e. it can use only the memory allocated to the input, with the exception of the production of the set of culprits T. Let us assume that the space needed to store a single proposition is $O(1)$ memory units. Since we only need to remove one proposition from each no-good to restore consistency, algorithm 1 uses $O(|\Sigma|)$ memory units.

Time Complexity
The analysis for time complexity is based on a sequential-procesing system. Let us assume that we implement lists as array structures. Let us assume that we may determine the size of an array in $O(1)$ time. Let us also assume that performing a comparison using $\dot{\leq}_{\leqslant}$ takes $O(1)$ time. Then in lines 2-4, for each array $\sigma \in \Sigma$ we find the minimum element σ and perform a swap on two elements at most once for each element in σ. If we let s_{max} be the cardinality of the largest σ in Σ, then lines 2-4 will take $O(|\Sigma| \cdot s_{max})$ time. In line 5, we sort the no-goods' positions in Σ using their first elements as keys. This takes $O(|\Sigma| \cdot log(|\Sigma|))$ time. Lines 7-16 iterate through the elements of Σ at most once for each element in Σ. During each such iteration, a search is performed for an element within a no-good. Also, during each iteration through all the no-goods, at least one σ is removed, though this does not help asymptotically. Since the no-goods are not sorted, the search takes linear time in s_{max}. So lines 7-16 take $O(|\Sigma|^2 \cdot s_{max})$ time. Therefore, the running time is $O(|\Sigma|^2 \cdot s_{max})$ time.

Note that the situation changes slightly if we sort the no-goods instead of just placing the minimally-entrenched proposition at the front, as in lines 2-4. In this case, each search through a no-good will take $O(log(s_{max}))$ time, yielding a new total time of $O(|\Sigma| \cdot s_{max} \cdot log(s_{max}) + |\Sigma|^2 \cdot log(s_{max}))$.

4 Analysis of Algorithm 2

4.1 Proofs of Satisfaction of Requirements by Algorithm 2

We show that Algorithm 2 satisfies the requirements established in section 2:

$(EE_{SNePS}1)$ **(Sufficiency)**
Since every set of propositions must contain at least one proposition that is minimally entrenched, at least one proposition is added to the list in each iteration of *ListLoop*. In the worst case, assume that for each iteration of *MainLoop*, only either *RemoveLoop* or *ModifyLoop* do any work. We know that at least this much work is done for the following reasons: if *ModifyLoop* cannot operate on any no-good during an iteration of *MainLoop*, then all no-goods have only one minimally-entrenched proposition. So either *RemoveLoop*'s condition at line 10 would hold, or:
1. A no-good has multiple minimally-entrenched propositions, causing *ModifyLoop* to do work. This contradicts our assumption that *ModifyLoop* could not do any work during this iteration of *MainLoop*, so we set this possibility aside.
2. Some proposition p_1 is a non-minimally-entrenched proposition in some no-good σ_n, and a minimally-entrenched one in another no-good σ_m. In this case, either p_1 is removed during the iteration of *RemoveLoop* where σ_m is considered, or there is another proposition p_2 in σ_m that is not minimally-entrenched in σ_m, but is in $\sigma_{m'}$. This chaining must eventually terminate at a no-good $\sigma_{m_{final}}$ since $\leqslant$ is transitive. And the final proposition in the chain p_{final} must be the sole minimally-entrenched proposition in σ_{final}, since otherwise *ModifyLoop* would have been able to do work for this iteration of *MainLoop*, which is a contradiction. *ModifyLoop* can only do work once for each no-good, so eventually its work is finished. If *ModifyLoop* has no more work left to do, then *RemoveLoop* must do work at least once for each iteration of *MainLoop*. And in doing so, it will create a list of culprits of which each no-good contains at least one. Q.E.D.

$(EE_{SNePS}2)$ **(Minimal Entrenchment)**
Since propositions are only added to T when the condition in line 10 is satisfied, it is guaranteed that every proposition in

T is a minimally-entrenched proposition in some no-good σ.

$(EE_{SNePS}3)$ **(Information Preservation)**
From line 10, we see that when a proposition p is removed, none of the other propositions in its no-good are minimally-entrenched in any other no-good. That means none of the other propositions could be a candidate for removal. So, the only way to remove the no-good in which p appears is by removing p. So if p were not removed, then $(EE_{SNePS}1)$ would not be satisfied. Q.E.D.

Decidability
ListLoop creates lists of minimal elements of lists. This is a decision procedure since the comparator is a total preorder. From the proof of $(EE_{SNePS}1)$ above, we see that either *RemoveLoop* or *ModifyLoop* must do work for each iteration of *MainLoop*. *ModifyLoop* cannot operate more than once on the same no-good, because there are no longer multiple minimally-entrenched propositions in the no-good after it does its work. Nor can *RemoveLoop* operate twice on the same no-good, since the no-good is removed when *ModifyLoop* does work. So, eventually *ModifyLoop* has no more work to do, and at that point *RemoveLoop* will remove at least one no-good for each iteration of *MainLoop*. By lines 17-18, when the last no-good is removed, the procedure terminates. So it always terminates. Q.E.D.

Supplementary Requirement
RemoveLoop attempts to compute T each time it is run from *MainLoop*. If the procedure does not terminate within *RemoveLoop*, then we run *ModifyLoop* on *at most one* no-good. Afterwards, we run RemoveLoop again. Since the user is only queried when the procedure cannot automatically determine any propositions to remove, we argue that this means minimal queries are made of the user. Q.E.D.

4.2 Complexity of Algorithm 2

Space Complexity
As before, let s_{max} be the cardinality of the largest no-good in Σ. In the worst case all propositions are minimally entrenched, so *ListLoop* will recreate Σ. So *ListLoop* will use $O(|\Sigma| \cdot s_{max})$ space. RemoveLoop creates a culprit list, which we stated before takes $O(|\Sigma|)$ space. ModifyLoop may be implemented in a variety of ways. We will assume that it creates a list of pairs, of which the first and second elements range over propositions in the no-goods. In this case *ModifyLoop* uses $O(|\Sigma|^2 \cdot s_{max}^2)$ space. So the total space requirement is $O(|\Sigma|^2 \cdot s_{max}^2)$ memory units.

Time Complexity
The analysis for time complexity is based on a sequential-procesing system. For each no-good σ, in the worst case, *ListLoop* will have to compare each proposition in σ agains every other. So, for each iteration of *MainLoop*, *ListLoop* takes $O(|\Sigma| \cdot s_{max}^2)$ time. There are at most $O(s_{max})$ elements in each list created by *ListLoop*. So, checking the condition in line 10 takes $O(|\Sigma| \cdot s_{max}^2)$ time. Lines 12-16 can be executed in $O(|\Sigma| \cdot s_{max})$ time. Therefore, *RemoveLoop* takes $O(|\Sigma| \cdot s_{max}^2)$ time. We assume that all the work in lines 24-27 can be done in constant time. So, *ModifyLoop* takes $O(|\Sigma|)$ time. We noted earlier that during each iteration of *MainLoop*, *RemoveLoop* or *ModifyLoop* will do work. In the worst case, only one will do work each time. And they each may do work at most $|\Sigma|$ times. So the total running time for the procedure is $O(|\Sigma|^2 \cdot s_{max}^2)$.

5 Annotated Demonstrations

A significant feature of our work is that it generalizes previous published work on belief revision in SNePS [Johnson and Shapiro, 1999; Shapiro and Johnson, 2000; Shapiro and Kandefer, 2005]. The following demonstrations showcase the new features we have introduced to SNeBR, and capture the essence of belief revision as seen in the papers mentioned above by using well-specified epistemic ordering functions. The demos have been edited for formatting and clarity.

The commands *br-tie-mode auto* and *br-tie-mode manual* indicate that Algorithm 1 and Algorithm 2 should be used respectively. A *wff* is a well-formed formula. A wff followed by a period (.) indicates that the wff should be asserted, i.e. added to the knowledge base. A wff followed by an exclamation point (!) indicates that the wff should be asserted, and that forward inference should be performed on it.

Says Who?
We present a demonstration on how the source-credibility-based revision behavior from [Shapiro and Johnson, 2000] is generalized by our changes to SNeBR. The knowledge base in the demo is taken from [Johnson and Shapiro, 1999]. In the following example, the command *set-order source* sets the epistemic ordering used by SNeBR to be a lisp function that compares two propositions based on the relative credibility of their sources. Unsourced propositions are assumed to have maximal credibility. The sources, as well as their relative credibility are represented as meta-knowledge in the SNePS knowledge base. This was also done in [Johnson and Shapiro, 1999] and [Shapiro and Johnson, 2000]. The *source* function makes SNePSLOG queries to determine sources of propositions and credibility of sources, using the *askwh* and *ask* commands [Shapiro and The SNePS Implementation Group, 2010]. This allows it to perform inference in making determinations about sources.

Here we see that the nerd and the sexist make the generalizations that all jocks are not smart and all females are not smart respectively, while the holy book and the professor state that all old people are smart, and all grad students are smart respectively. Since Fran is an old female jock graduate student, there are two sources that would claim she is smart, and two that would claim she is not, which is a contradiction.

```
;;; Show origin sets
: expert
: br—mode auto
Automatic belief revision will now be automatically selected.
: br—tie—mode manual
The user will be consulted when an entrenchment tie occurs
;;; Use source credibilities as epistemic ordering criteria.
set—order source
;;; The holy book is a better source than the professor.
IsBetterSource(holybook, prof).
;;; The professor is a better source than the nerd.
IsBetterSource(prof, nerd).
;;; The nerd is a better source than the sexist.
IsBetterSource(nerd, sexist).
```

```
;;; Fran is a better source than the nerd.
IsBetterSource(fran, nerd).
;;; Better-Source is a transitive relation
all(x,y,z)({IsBetterSource(x,y), IsBetterSource(y,z)} &=>
      IsBetterSource(x,z))!
;;; All jocks are not smart.
all(x)(jock(x)=>~smart(x)). ;wff10
;;; The source of the statement 'All jocks are not smart' is the nerd
HasSource(wff10, nerd).
;;; All females are not smart.
all(x)(female(x)=>~smart(x)). ;wff12
;;; The source of the statement 'All females are not smart' is the
      sexist.
HasSource(wff12, sexist).
;;; All graduate students are smart.
all(x)(grad(x)=>smart(x)). ;wff14
;;; The source of the statement 'All graduate students are smart' is
      the professor.
HasSource(wff14, prof).
;;; All old people are smart.
all(x)(old(x)=>smart(x)). ;wff16
;;; The source of the statement 'All old people are smart' is the
      holy book.
HasSource(wff16, holybook).
;;; The source of the statement 'Fran is an old female jock who is a
      graduate student' is fran.
HasSource(and{jock(fran),grad(fran),female(fran),old(fran)},fran).
;;; The KB thus far list-asserted-wffs
wff23!: HasSource(old(fran) and female(fran) and grad(fran) and
             jock(fran),fran)  {<hyp,{wff23}>}
wff17!: HasSource(all(x)(old(x) => smart(x)),holybook){<hyp,{wff17}>}
wff16!: all(x)(old(x) => smart(x))  {<hyp,{wff16}>}
wff15!: HasSource(all(x)(grad(x) => smart(x)),prof)  {<hyp,{wff15}>}
wff14!: all(x)(grad(x) => smart(x))  {<hyp,{wff14}>}
wff13!: HasSource(all(x)(female(x) => (~smart(x))),sexist)
             {<hyp,{wff13}>}
wff12!: all(x)(female(x) => (~smart(x)))  {<hyp,{wff12}>}
wff11!: HasSource(all(x)(jock(x) => (~smart(x))),nerd) <hyp,{wff11}>}
wff10!: all(x)(jock(x) => (~smart(x)))  {<hyp,{wff10}>}
wff9!:  IsBetterSource(fran,sexist)  {<der,{wff3,wff4,wff5}>}
wff8!:  IsBetterSource(prof,sexist)  {<der,{wff2,wff3,wff5}>}
wff7!:  IsBetterSource(holybook,sexist) {<der,{wff1,wff2,wff3,wff5}>}
wff6!:  IsBetterSource(holybook,nerd)  {<der,{wff1,wff2,wff5}>}
wff5!:  all(z,y,x)({IsBetterSource(y,z),IsBetterSource(x,y)} &=>
             {IsBetterSource(x,z)})  {<hyp,{wff5}>}
wff4!:  IsBetterSource(fran,nerd)  {<hyp,{wff4}>}
wff3!:  IsBetterSource(nerd,sexist)  {<hyp,{wff3}>}
wff2!:  IsBetterSource(prof,nerd)  {<hyp,{wff2}>}
wff1!:  IsBetterSource(holybook,prof)  {<hyp,{wff1}>}
;;; Fran is an old female jock who is a graduate student (asserted
      with forward inference).
and{jock(fran),grad(fran),female(fran),old(fran)}!
wff50!: ~(all(x)(jock(x) => (~smart(x))))
             {<ext,{wff16,wff22}>,<ext,{wff14,wff22}>}
wff24!: smart(fran)  {<der,{wff16,wff22}>,<der,{wff14,wff22}>}
;;; The resulting knowledge base (HasSource and IsBetterSource omited
       for clarity)
list-asserted-wffs
wff50!: ~(all(x)(jock(x) => (~smart(x))))
             {<ext,{wff16,wff22}>, <ext,{wff14,wff22}>}
wff37!: ~(all(x)(female(x) => (~smart(x))))  {<ext,{wff16,wff22}>}
wff24!: smart(fran)  {<der,{wff16,wff22}>,<der,{wff14,wff22}>}
wff22!: old(fran) and female(fran) and grad(fran) and jock(fran)
             {<hyp,{wff22}>}
wff21!: old(fran)  {<der,{wff22}>}
wff20!: female(fran)  {<der,{wff22}>}
wff19!: grad(fran)  {<der,{wff22}>}
wff18!: jock(fran)  {<der,{wff22}>}
wff16!: all(x)(old(x) => smart(x))  {<hyp,{wff16}>}
wff14!: all(x)(grad(x) => smart(x))  {<hyp,{wff14}>}
```

We see that the statements that all jocks are not smart and that all females are not smart are no longer asserted at the end. These statements supported the statement that Fran *is not* smart. The statements that all old people are smart and that all grad students are smart supported the statement that Fran *is* smart. The contradiction was resolved by contracting "Fran *is not* smart," since the sources for its supports were less credible than the sources for "Fran *is* smart."

Wumpus World

We present a demonstration on how the state-constraint-based revision behavior from [Shapiro and Kandefer, 2005] is generalized by our changes to SNeBR. The command *set-order fluent* says that propositional fluents are strictly less entrenched than non-fluent propositions. The *fluent* order was created specifically to replace the original belief revision behavior of the SNeRE *believe* act. In the version of SNeBR used in [Shapiro and Kandefer, 2005], propositions of the form $andor(<0|1>,1)(p_1,p_2,\ldots)$ were assumed to be state contraints, while the inner propositions, p_1, p_2, etc., were assumed to be fluents. The fluents were less entrenched than the state constraints. We see that the ordering was heavily syntax-dependent.

In our new version, the determination of which propositions are fluents is made by checking for membership of the predicate symbol of an atomic proposition in a list called `*fluents*`, which is defined by the user to include the predicate symbols of all propositional fluents. So the entrenchment ordering defined here uses metaknowledge about the knowledge base that is not represented in the SNePS knowledge base. The command *br-tie-mode manual* indicates that Algorithm 2 should be used. Note that the *xor* connective [Shapiro, 2010] used below replaces instances of *andor(1,1)(...)* from [Shapiro and Kandefer, 2005]. The command `perform believe(wff)` is identical to the command `wff!`, except that the former causes `wff` to be strictly more entrenched than every other proposition during belief revision. That is, `wff` is guaranteed to be *safe* (unless `wff` is itself a contradiction). So we would be using *prioritized* belief revision.

```
;;; Show origin sets
: expert
;;; Always use automatic belief revision
: br-mode auto
Automatic belief revision will now be automatically selected.
;;; Use algorithm 2
: br-tie-mode manual
The user will be consulted when an entrenchment tie occurs.
;;; Use an entrenchment ordering that favors non-fluents over
;;; fluents
set-order fluent
;;; Establish what kinds of propositions are fluents; specifically,
      that the agent is facing some direction is a fact that may
      change over time.
^(setf *fluents* '(Facing))
;;; The agent is Facing west
Facing(west).
;;; At any given time, the agent is facing either north, south, east,
       or west (asserted with forward inference).
xor{Facing(north),Facing(south),Facing(east), Facing(west)}!
;;; The knowledge base as it stands
list-asserted-wffs
wff8!:  ~Facing(north)  {<der,{wff1,wff5}>}
wff7!:  ~Facing(south)  {<der,{wff1,wff5}>}
wff6!:  ~Facing(east)  {<der,{wff1,wff5}>}
wff5!:  xor{Facing(east),Facing(south),Facing(north), Facing(west)}
      {<hyp,{wff5}>}
wff1!:  Facing(west)  {<hyp,{wff1}>}
;;; Tell the agent to believe it is now facing east.
perform believe(Facing(east))
;;; The resulting knowledge base
list-asserted-wffs
wff10!:  ~Facing(west)  {<ext,{wff4,wff5}>}
wff8!:  ~Facing(north)  {<der,{wff1,wff5}>,<der,{wff4,wff5}>}
wff7!:  ~Facing(south)  {<der,{wff1,wff5}>,<der,{wff4,wff5}>}
```

```
wff5!:  xor{Facing(east),Facing(south),Facing(north),Facing(west)} {<
        hyp,{wff5}>}
wff4!:  Facing(east)  {<hyp,{wff4}>}
```

There are three propositions in the no-good when revision is performed: `Facing(west)`, `Facing,east`, and `xor(1,1){Facing(...}`. `Facing(east)` is not considered for removal since it was prioritized by the believe action. The state-constraint `xor(1,1){Facing...}` remains in the knowledge base at the end, because it is more entrenched than `Facing(west)`, a propositional fluent, which is ultimately removed.

6 Conclusions

Our modified version of SNeBR provides decision procedures for belief revision in SNePS. By providing a single resulting knowledge base, these procedures essentially perform maxichoice revision for SNePS. Using a well preorder, belief revision can be performed completely automatically. Given a total preorder, it may be necessary to consult the user in order to simulate a well preorder. The simulated well preorder need only be partially specified; it is only necessary to query the user when multiple beliefs are minimally-epistemically-entrenched within a no-good, and even then only in the case where no other belief in the no-good is already being removed. In any event, the epistemic ordering itself is *user-supplied*. Our algorithm for revision given a well preorder uses asymptotically less time and space than the other algorithm, which uses a total preorder. Our work generalize previous belief revision techniques employed in SNePS.

Acknowledgments

We would like to thank Prof. William Rapaport for providing editorial review, and Prof. Russ Miller for his advice concerning the analysis portion of this paper.

References

[Alchourron and Makinson, 1985] C.E. Alchourron and D. Makinson. On the logic of theory change: Safe contraction. *Studia Logica*, (44):405–422, 1985.

[Alchourron *et al.*, 1985] C. E. Alchourron, P. Gärdenfors, and D. Makinson. On the logic of theory change: Partial meet contraction and revision functions. *Journal of Symbolic Logic*, 20:510–530, 1985.

[de Kleer, 1986] J. de Kleer. An Assumption-Based TMS. *Artificial Intelligence*, 28:127–162, 1986.

[Gärdenfors and Rott, 1995] P. Gärdenfors and H. Rott. Belief revision. In Gabbay, Hogger, and Robinson, editors, *Epistemic and Temporal Reasoning*, volume 4 of *Handbook of Logic in Artificial Intelligence and Logic Programming*, pages 35–131. Clarendon Press, Oxford, 1995.

[Gärdenfors, 1982] P. Gärdenfors. Rules for rational changes of belief. In T. Pauli, editor, *Philosophical Essays Dedicated to Lennart Åqvist on His Fiftieth Birthday*, number 34 in Philosophical Studies, pages 88–101, Uppsala, Sweden, 1982. The Philosophical Society and the Department of Philosophy, University at Uppsala.

[Gärdenfors, 1988] P. Gärdenfors. *Knowledge in Flux: Modeling the Dynamics of Epistemic States*. The MIT Press, Cambridge, Massachusetts, 1988.

[Gärdenfors, 1989] P. Gärdenfors. The dynamics of belief systems: Foundations vs. coherence. *Revue Internationale de Philosophie*, 1989.

[Hansson and Olsson, 1999] S. O. Hansson and E. J. Olsson. Providing foundations for coherentism. *Erkenntnis*, 51(2–3):243–265, 1999.

[Hansson, 1994] S. O. Hansson. Kernel contraction. *The Journal of Symbolic Logic*, 59(3):845–859, 1994.

[Hansson, 1997] S. O. Hansson. Semi-revision. *Journal of Applied Non-Classical Logics*, 7(2):151–175, 1997.

[Hansson, 1999] S. O. Hansson. A survey of non-prioritized belief revision. *Erkenntnis*, 50:413–427, 1999.

[Johnson and Shapiro, 1999] F. L. Johnson and S. C. Shapiro. Says Who? - Incorporating Source Credibility Issues into Belief Revision. Technical Report 99-08, Department of Computer Science and Engineering, SUNY Buffalo, Buffalo, NY, 1999.

[Lakemeyer, 1991] Lakemeyer. On the relation between explicit and implicit beliefs. In *Proc. KR-1991*, pages 368–375. Morgan Kaufmann, 1991.

[Martins and Shapiro, 1988] J. P. Martins and S. C. Shapiro. A model for belief revision. *Artificial Intelligence*, 35(1):25–79, 1988.

[Shapiro and Johnson, 2000] S. C. Shapiro and F. L. Johnson. Automatic belief revision in SNePS. In C. Baral and M. Truszczynski, editors, *Proc. NMR-2000*, 2000. unpaginated, 5 pages.

[Shapiro and Kandefer, 2005] S. C. Shapiro and M. Kandefer. A SNePS Approach to the Wumpus World Agent or Cassie Meets the Wumpus. In L. Morgenstern and M. Pagnucco, editors, *NRAC-2005*, pages 96–103, 2005.

[Shapiro and The SNePS Implementation Group, 2010] Stuart C. Shapiro and The SNePS Implementation Group. *SNePS 2.7.1 USER'S MANUAL*. Department of Computer Science and Engineering, SUNY Buffalo, December 2010.

[Shapiro, 1992] Stuart C. Shapiro. Relevance logic in computer science. Section 83 of A. R. Anderson and N. D. Belnap, Jr. and J. M/ Dunn *et al. Entailment, Volume II*, pages 553–563. Princeton University Press, Princeton, NJ, 1992.

[Shapiro, 2010] S. C. Shapiro. Set-oriented logical connectives: Syntax and semantics. In F. Lin, U. Sattler, and M. Truszczynski, editors, *KR-2010*, pages 593–595. AAAI Press, 2010.

[Williams, 1994] M.-A. Williams. On the logic of theory base change. In C. MacNish, D. Pearce, and L. Pereira, editors, *Logics in Artificial Intelligence*, volume 838 of *Lecture Notes in Computer Science*, pages 86–105. Springer Berlin / Heidelberg, 1994.

Decision-Theoretic Planning for Golog Programs with Action Abstraction

Daniel Beck and Gerhard Lakemeyer
Knowledge Based Systems Group
RWTH Aachen University, Aachen, Germany
{dbeck,gerhard}@cs.rwth-aachen.de

Abstract

DTGolog combines the ability to specify an MDP in a first-order language with the possibility to restrict the search for an optimal policy by means of programs. In particular, it employs decision-theoretic planning to resolve the nondeterminism in the programs in an optimal fashion (wrt an underlying optimization theory). One of the nondeterministic constructs DTGolog offers is the nondeterministic choice of action arguments. The possible choices, though, are restricted to a finite, predefined list. We present an extension to DTGolog that overcomes this restriction but still retains the optimality property of DTGolog. That is, in our extended version of DTGolog we can formulate programs that allow for an unrestricted choice of action arguments even in domains where there are infinitely many possible choices. The key to this is that we compute the optimal execution strategy for a program on the basis of abstract value functions. We present experiments which show that these extensions may lead to a speed-up in the computation time in comparison to the original DTGolog.

1 Introduction

Markov decision processes (MDPs) [Puterman, 1994] have proved to be a conceptually adequate model for decision-theoretic planning. Their solution, though, is often intractable. DTGolog [Boutilier *et al.*, 2000], a decision-theoretic extension of the high-level agent programming language Golog [Levesque *et al.*, 1997], tackles this problem by constraining the search space with a Golog program. In particular, only the policies which comply with the program are considered during the search. The agent programming language Golog is based on the situation calculus, has a clearly defined semantics and offers programming constructs known from other programming languages (e.g., conditionals, nondeterministic choice, etc.). Thus, DTGolog programs can be understood as an advice to the decision-theoretic planner. Their semantics is understood as the optimal execution of the program.

There is one particular nondeterministic construct, namely the nondeterministic choice of arguments, which is a little troublesome in DTGolog. Whereas the semantics of Golog allow the agent to freely choose those arguments, DTGolog needs to restrict the choice to a finite, pre-defined list. The reason being that DTGolog performs a forward search and branches over the possible continuations of the remaining program (and also over the outcomes of stochastic actions) which requires that the number of successor states in the search tree is finite. Generally, what the possible choices are and how many there are in any domain instance is unknown a-priori and thus the approach of DTGolog is not directly extensible to handle an unconstrained nondeterministic choice of arguments. In [Boutilier *et al.*, 2001] an approach that allows to solve an MDP using dynamic programming methods on a purely symbolic level was presented. The key idea was that from the first-order description of the MDP a first-order representation of the value function can be derived. This representation of the value function allows not only abstraction over the state space but also it allows to abstract over action instances. We show how these ideas extend in the presence of programs that constrain the search for the optimal policy. Finding the optimal execution of a DTGolog program (or, more precisely, the optimal policy compatible with the program) is understood as a multi-objective optimization problem where the objectives are the expected cumulative reward and the probability of successfully executing the program. The latter refers to the probability of not ending up in a configuration in which the program cannot be executed any further. We show how symbolic representations of the functions representing these quantities can be derived. With the help of these functions we then can extend the semantics of DTGolog to programs containing an unrestricted choice of arguments. In fact, we show that for DTGolog programs the original DTGolog interpreter and our extended version compute the same policies.

We provide a short overview of the situation calculus, Golog and DTGolog in Section 1.1. In Section 2 we introduce the case notation which we use to represent the abstract value functions for Golog programs presented in Section 3. Using these abstract value functions we provide the semantics of our DTGolog extension in Section 4. We discuss the advantages and disadvantages of our extension over the original DTGolog in Section 5.

1.1 The Situation Calculus and Golog

The situation calculus is a first-order logic (with second-order elements which are of no concern to us, here) with sorts for situations and actions. The binary predicate symbol $do(a, s)$ denotes the situation resulting from executing action a in situation s; the constant S_0 denotes the initial situation. Fluents are regular function- or predicate-symbols that take a term of sort situation as their last argument. According to Reiter's solution of the frame problem (cf. [Reiter, 1991]) the value of a fluent in a particular situation can be determined with the help of so-called successor-state axioms (SSAs) of which one has to exists for every fluent. For instance for the fluent $F(\vec{x}, s)$:

$$F(\vec{x}, do(a, s)) \equiv \Phi_F(\vec{x}, a, s)$$

where, intuitively, $\Phi_F(\vec{x}, a, s)$ describes the conditions which have to hold in situation s such that in the successor situation $do(a, s)$, after executing the action a, the fluent F holds for the parameters $\vec{x}$.

The preconditions for actions are specified by axioms of the form

$$Poss(A(\vec{x}), s) \equiv \Pi_A(\vec{x}, s)$$

where $\Pi_A(\vec{x}, s)$ describes the preconditions that have to hold before the action $A(\vec{x})$ can be executed.

A basic action theory (BAT) $\mathcal{D}$ then consists of the foundational axioms Σ constraining the form of situation terms, the successor-state axiom $\mathcal{D}_{ssa}$, the action preconditions $\mathcal{D}_{ap}$, unique names assumptions for actions $\mathcal{D}_{una}$, and a description of the initial situation $\mathcal{D}_{S_0}$.

By means of regression a regressable formula, basically a formula where all terms of sort situation are rooted in S_0, can be transformed into an equivalent formula which only mentions the initial situation S_0. Thereby reasoning is restricted to reasoning about formulas in the initial situation. In particular, every occurrence of a fluent having a non-initial situation as its last argument is replaced with the right-hand side of its SSA. The regression operator $\mathcal{R}$ for a formula whose situation arguments are of the form $do(a, s)$ is defined as follows:

$$\begin{aligned}
&\mathcal{R}(F(\vec{x}, do(a, s))) = \Phi_F(\vec{x}, a, s)\\
&\mathcal{R}(\neg\phi) = \neg\mathcal{R}(\phi)\\
&\mathcal{R}(\phi \wedge \psi) = \mathcal{R}(\phi) \wedge \mathcal{R}(\psi)\\
&\mathcal{R}(\exists x.\, \phi) = \exists x.\, \mathcal{R}(\phi)
\end{aligned}$$

In order to model stochastic domains, that is, domains where the effect of performing an action is not deterministic, but different effects might occur with certain probabilities, some kind of stochastic actions are necessary. In DTGolog those stochastic actions are modelled with the help of a number of associated, deterministic actions that describe the possible outcomes when executing the stochastic action. The understanding is that Nature chooses between the associated, deterministic actions when executing the stochastic action. For instance, in a Blocks World domain the (stochastic) $move(b_1, b_2)$ action whose outcomes are described by the deterministic actions $moveS(b_1, b_2)$ and $moveF(b_1, b_2)$, respectively. Notationally, this is captured by

$$\begin{aligned}
&choice(move(b_1, b_2), a) \overset{def.}{=}\\
&\qquad a = moveS(b_1, b_2) \vee a = moveF(b_1, b_2)
\end{aligned}$$

For every combination of a stochastic action and one of its associated, deterministic actions the probability with which Nature chooses the deterministic action needs to be specified. To continue the example from above we might have

$$\begin{aligned}
&prob_0(moveS(b_1, b_2), move(b_1, b_2), s) = p \overset{def.}{=}\\
&\qquad \neg heavy(b_1) \wedge p = 0.9 \vee heavy(b_1) \wedge p = 0.1\\
&prob_0(moveF(b_1, b_2), move(b_1, b_2), s) = p \overset{def.}{=}\\
&\qquad \neg heave(b_1) \wedge p = 0.1 \vee heavy(b_1) \wedge p = 0.9
\end{aligned}$$

which says that if the block to be moved is heavy the $move$-action succeeds with a probability of 0.1 and fails with a probability of 0.9. Note that $prob_0$ neglects the preconditions of the associated, deterministic actions. Also, the probability of Nature choosing another than one of the associated, deterministic actions has to be 0:

$$\begin{aligned}
&prob(a, \alpha, s) = p \overset{def.}{=}\\
&\qquad choice(\alpha, a) \wedge Poss(a, s) \wedge p = prob_0(a, \alpha, s)\\
&\qquad\qquad \vee \neg(choice(\alpha, a) \wedge Poss(a, s)) \wedge p = 0
\end{aligned}$$

It is crucial that the axiomatizer ensures that the probability distribution is well-defined, that is, the probabilities over the deterministic outcome actions always sum up to 1.

When actually executing a program containing stochastic actions, it is necessary to determine which of the associated, deterministic actions has been selected by Nature during the execution. Consequently, some kind of sensing is required. In particular, we assume that for every stochastic action there is a unique *associated sense action* (which itself is a stochastic action) and *sense outcome conditions* which discriminate Nature's choices. The intention behind this is that when the agent actually executes a stochastic action, it can execute the associated sense action to acquire the necessary information from its sensors afterwards to unambiguously determine the action chosen by Nature with the help of the sense outcome conditions. Since we assume full observability, we can assume that the sensing is accurate and consequently the associated sense action is a noop-action in the theory.

Besides the BAT DTGolog also requires an optimization theory in order to determine the optimal execution strategy for a program. This theory includes axioms defining the reward function $reward(s)$ which assesses the current situation. For instance:

$$reward(do(moveS(B_1, B_2), s)) = 10$$

The kind of programs we consider are similar to regular Golog programs with the only exception that the primitives in the programs are not deterministic but stochastic actions. In particular, the following program constructs are available:

$\delta_1; \delta_2$	sequences
$\vartheta?$	test actions
if ϑ **then** δ_1 **else** δ_2 **end**	conditionals
while ϑ **do** δ **end**	loops
$(\delta_1 \mid \delta_2)$	nondeterministic branching
$\pi\, v.\, (\gamma)$	nondeterministic choice of argument
proc $P(\vec{x})\ \delta_P$ **end**	procedures (including recursion)

In DTGolog only a restricted version of the nondeterministic choice of argument is supported which is semantically equivalent to a nondeterministic branching over the same program but with different arguments. In our extension of DTGolog we support the unrestricted nondeterministic choice of arguments.

2 The Case Notation

We use a case notation similar to that introduced in [Boutilier *et al.*, 2001]. This notation is convenient for the representation of finite case distinctions, that is, piecewise constant functions which have a finite number of different values. We write $case[\phi_1, v_1; \dots; \phi_n, v_n]$ (or $case[\phi_i, v_i : i \leq n]$ for short) as an abbreviation for

$$\bigvee_{i=1}^{n} \phi_i \wedge \mu = v_i$$

The v_i are numerical expressions, that is, expressions that evaluate to numbers. The variable μ is a special variable that is reserved for the use in case statements and must not be used anywhere else. In order to use case statements in formulas without explicitly referring to μ we define the following macro:

$$v = case[\phi_i, v_i : i \leq n] \stackrel{def.}{=} case[\phi_i, v_i : i \leq n]^{\mu}_{v}$$

Furthermore, we slightly extend the case notation to allow the representation of a two-valued function: $case[\phi_1, (v_1, p_1); \dots; \phi_n, (v_n, p_n)]$ is used as an abbreviation for:

$$\bigvee_{i=1}^{n} \phi_i \wedge \mu_1 = v_i \wedge \mu_2 = p_i \ .$$

The v_i and p_i are numerical expressions and μ_1 and μ_2 are reserved variables. In a similar fashion as in the single-valued case we define the macro:

$$(v, p) = case[\phi_i, (v_i, p_i) : i \leq n] \stackrel{def.}{=} case[\phi_i, (v_i, p_i) : i \leq n]^{\mu_1}_{v}{}^{\mu_2}_{p} \ .$$

By means of the $\circ$-operator two single-valued case statements can be combined to a two-valued case statement:

$$case[\phi_i, v_i : i \leq n] \circ case[\psi_j, v'_j : j \leq m] \stackrel{def.}{=} case[\phi_i \wedge \psi_j, (v_i, v'_j) : i \leq n, j \leq m]$$

Further operators we use (for single-valued case statements) are the binary operators $\oplus$, $\otimes$, and $\cup$ and the unary *casemax*-operator for symbolic maximization (cf. [Sanner and Boutilier, 2009]).

$$\begin{aligned}
&case[\phi_i, v_i, : i \leq n] \otimes case[\psi_j, v'_j : j \leq m] = \\
&\qquad case[\phi_i \wedge \psi_j, v_i \cdot v'_j : i \leq n, j \leq m] \\
&case[\phi_i, v_i : i \leq n] \oplus case[\psi_j, v'_j : j \leq m] = \\
&\qquad case[\phi_i \wedge \psi_j, v_i + v'_j : i \leq n, j \leq m] \\
&case[\phi_i, v_i : i \leq n] \cup case[\psi_j, v'_j : j \leq m] = \\
&\qquad case[\phi_1, v_i; \dots; \phi_n, v_n; \psi_1, v'_1; \dots; \psi_m, v'_m]
\end{aligned}$$

For the casemax-operator we assume that the formulas in the input case statement are sorted in a descending order, that is, $v_i > v_{i+1}$. For this it is necessary that the v_i are numerical constants which is what we assume for the remainder of this paper. Then, the operator is defined as follows:

$$\mathrm{casemax}\, case[\phi_i, v_i : i \leq n] \stackrel{def.}{=} case[\phi_i \wedge \bigwedge_{j<i} \neg\phi_j, v_i : i \leq n] \ .$$

Generally, a formula $v = case[\phi_i, v_i : i \leq n]$ might be ambiguous wrt the value of v, i.e., the ϕ_i are not required to hold mutually exclusively. Applying the casemax-operator to $case[\phi_i, v_i : i \leq n]$ remedies this problems. In the resulting case statement the formulas hold mutual exclusively and, furthermore, the value of v is maximized. Given an ordering over two-valued tuples that allow to pre-sort the formulas in the input case statement the casemax-operator can be applied on two-value case statements as well.

The expressions $\psi \wedge case[\phi_i, v_i : i \leq n]$ and $\exists x.\, case[\phi_i, v_i : i \leq n]$ are case statements as well. Due to the disjunctive nature of the case statements the conjunction can be distributed into the disjunction and the existential quantifier can be moved into the disjunction. The resulting case statements then are $case[\psi \wedge \phi_i, v_i : i \leq n]$ and $case[\exists x.\, \phi_i, v_i : i \leq n]$, respectively.

We assume that the reward function $reward(s)$ and the probability distribution over the deterministic actions associated with a stochastic action are specified using case statements:

$$reward(s) = case[\phi^{rew}_1(s), r_1; \cdots; \phi^{rew}_m(s), r_m]$$

and

$$prob(N_j(\vec{x}), A(\vec{x}), s) = case[\phi^A_{j,1}(\vec{x}, s), p_1; \cdots; \phi^A_{j,n}(\vec{x}, s), p_n]$$

We denote them by $rCase(s)$ and $pCase^A_j(\vec{x}, s)$, respectively. Since these case statements define functions it is necessary that each of the sets $\{\phi^{rew}_i(s)\}$ and the $\{\phi^A_{j,i}(\vec{x}, s)\}$ *partitions* the state space. That is, for every $\vec{x}$ and s a unique value can be determined. Formally, a set of formulas $\{\psi_i(\vec{x}, s)\}$ is said to partition the state space iff $\models \forall \vec{x}, s.\, \bigvee_i \psi_i(\vec{x}, s)$ and $\models \forall \vec{x}, s.\, \psi_i(\vec{x}, s) \supset \neg\psi_j(\vec{x}, s)$ for all $i, i \neq j$.

3 Abstract Value Functions

The type of programs we consider cannot be directly executed, the nondeterminism in the program needs to be resolved first. Of course, the agent executing the program strives for an optimal execution strategy for the program. Optimality, in this case, is defined with respect to the expected reward accumulated during the first h steps of the program and the probability that these first h steps can be executed successfully (i.e., the probability of not running into a situation in which the program cannot be executed any further). Those two quantities are measured by the value functions $V^{\delta}_{h}(s)$ and

$P_h^\delta(s)$, respectively. Our intention and the key to abstracting from the actual situation is to identify regions of the state space in which these functions are constant. The advantages of such an abstract function are twofold. First, these functions can be pre-computed since they are independent of the actual situation (and the initial situation). This allows to apply some simplification in order to lower the time necessary to evaluate the formula. Second, these abstract value functions allow to asses the values of a program containing nondeterministic choices of action arguments without explicitly enumerating all possible (ground) choices for these arguments. Rather the abstract value functions abstract from the actual choices by identifying properties for the action arguments that lead to a certain value for the expected reward and the probability of successfully executing the program, respectively. For instance, if a high reward is given to situations where there is a green block on top of a non-green block and the program tells the agent to pick a block and move it onto another nondeterministically chosen block, then the value function for the expected reward distinguishes the cases where a green block is moved on top of a non-green block from the other constellations. What it does not do is to explicitly refer to the green and the non-green blocks in the domain instance. Thus, given these abstract value functions, the nondeterminism in the program can be resolved by settling on the choice maximizing the abstract value functions when evaluated in the current situation.

For a program δ and a horizon h we compute case statements $V_h^\delta(s)$ and $P_h^\delta(s)$ representing the abstract value functions. As can be seen in the definition below the computation of these case statements is independent of the situation s. $V_h^\delta(s)$ and $P_h^\delta(s)$ are inductively defined on the structure of δ. Since the definition is recursive we first need to assume that the horizon h is finite and that the programs are nil-terminated which can be achieved easily by sequentially combining a program with the empty program nil.

1. Zero horizon:

   $$V_0^\delta(s) \overset{def.}{=} rCase(s) \text{ and } P_0^\delta(s) \overset{def.}{=} case[true, 1]$$

   For the remaining cases we assume $h > 0$.

2. The empty program nil:

   $$V_h^{nil}(s) \overset{def.}{=} rCase(s) \text{ and } P_h^{nil}(s) \overset{def.}{=} case[true, 1]$$

3. The program begins with a stochastic action $A(\vec{x})$ with outcomes $N_1(\vec{x}), \ldots, N_k(\vec{x})$:

   $$V_h^{A(\vec{x});\delta}(s) \overset{def.}{=} rCase(s) \oplus \bigoplus_{j=1}^{k} \left[pCase_j^A(\vec{x}, s) \otimes \mathcal{R}(V_{h-1}^\delta(do(N_j(\vec{x}), s)))\right]$$

   That is, the expected value is determined as the sum of the immediate reward and the sum over the expected values executing the remaining program in the possible successor situations $do(N_j(\vec{x}), s)$ each weighted by the probability of seeing the deterministic actions $N_j(\vec{x})$ as the outcome. Due to regression the formulas only refer to the situation s and not to any of the successor situations.

   For the probability of successfully executing the program the definition is quite similar only that the immediate reward is ignored:

   $$P_h^{A(\vec{x});\delta}(s) \overset{def.}{=} \bigoplus_{j=1}^{k} \left[pCase_j^A(\vec{x}, s) \otimes \mathcal{R}(P_{h-1}^\delta(do(N_j(\vec{x}), s)))\right]$$

4. The program begins with a test action:

   $$V_h^{\vartheta?;\delta}(s) \overset{def.}{=} (\vartheta[s] \land V_h^\delta(s)) \cup (\neg\vartheta[s] \land rCase(s))$$

   In case the test does not hold the execution of the program has to be aborted and consequently no further rewards are obtained.

   $$P_h^{\vartheta?;\delta}(s) \overset{def.}{=} (\vartheta[s] \land P_h^\delta(s)) \cup case[\neg\vartheta[s], 0]$$

5. The program begins with a conditional:

   $$V_h^{\textbf{if } \vartheta \textbf{ then } \delta_1 \textbf{else } \delta_2 \textbf{end};\delta}(s) \overset{def.}{=} (\vartheta[s] \land V_h^{\delta_1;\delta}(s)) \cup (\neg\vartheta[s] \land V_h^{\delta_2;\delta}(s))$$

   Analogous for $P_h^{\textbf{if } \vartheta \textbf{ then } \delta_1 \textbf{else } \delta_2 \textbf{end};\delta}(s)$.

6. The program begins with a nondeterministic branching:

   $$V_h^{(\delta_1 \mid \delta_2);\delta}(s) \overset{def.}{=} \text{casemax}\,(V_h^{\delta_1;\delta}(s) \cup_\geq V_h^{\delta_2;\delta}(s))$$

   where $\cup_\geq$ is an extended version of the $\cup$-operator that additionally sorts the formulas according to their values such that $v_i \geq v_{i+1}$ holds in the resulting case statement. Another minor modification of the $\cup$-operator is necessary to keep track of from where the formulas originate. The resulting case statement then looks like this:

   $$case[\phi_i, v_i \rightarrow idx_i]$$

   where $idx_i = 1$ if ϕ_i stems from $V_h^{\delta_1;\delta}(s)$ and $idx_i = 2$ if ϕ_i stems from $V_h^{\delta_2;\delta}(s)$. This allows the agent to reconstruct what branch has to be chosen when ϕ_i holds in the current situation. For all further operations on the case statement those mappings can be ignored. $P_h^{(\delta_1 \mid \delta_2);\delta}(s)$ is defined analogously.

7. The program begins with a nondeterministic choice of arguments:

   $$V_h^{\pi\, x.\, (\gamma);\delta}(s) \overset{def.}{=} \text{casemax}\, \exists x.\, V_h^{\gamma;\delta}(s)$$

   Note that the resulting case statement is independent of the actually available choices for x. The formulas $\phi_i(x, s)$ in $V_h^{\gamma;\delta}(s)$ (which mention x as a free variable) describe how the choice for x influences the expected reward for the remaining program $\gamma; \delta$. To obtain $V_h^{\pi\, v.\, (\gamma);\delta}(s)$ it is then maximized over the existentially quantified case statement $V_h^{\gamma;\delta}(s)$. Again, $P_h^{\pi\, x.\, (\gamma);\delta}(s)$ is defined analogously.

8. The program begins with a sequence:

$$V_h^{[\delta_1;\delta_2];\delta_3}(s) \overset{def.}{=} V_h^{\delta_1;[\delta_2;\delta_3]}(s)$$

   that is, we associate the sequential composition to the right. By possibly repetitive application of this rule the program is transformed into a form such that one of the cases above can be applied.

9. Procedure calls:

   The problem with procedures is that it is not clear how to macro expand a procedure's body when it includes a recursive procedure call. Similar to how procedures are handled by Golog's Do-macro we define an auxiliary macro:

$$V_h^{P(t_1,\ldots,t_n);\delta}(s) \overset{def.}{=} P(t_1[s],\ldots,t_n[s],\delta,s,h,v)$$

   We consider programs including procedure definitions to have the following form:

$$\{\mathbf{proc}\; P_1(\vec{v}_1)\; \delta_1\; \mathbf{end}; \cdots; \mathbf{proc}\; P_n(\vec{v}_n)\; \delta_n\; \mathbf{end}; \delta_0\}$$

   Then, we define the optimal expected value obtainable for executing the first h steps of such a program as:

$$\begin{aligned}
&V_h^{\{\mathbf{proc}\; P_1(\vec{v}_1)\; \delta_1\; \mathbf{end};\cdots;\mathbf{proc}\; P_n(\vec{v}_n)\; \delta_n\; \mathbf{end};\delta_0\}}(s) \overset{def.}{=} \\
&\quad \forall P_i.\,[\bigwedge_{i=1}^{n} \forall \vec{v}_i, s', h', \delta', v.\, v = V_{h'}^{\delta_i;\delta'}(s') \\
&\qquad\qquad \supset P(\vec{v}_i, \delta', s, h, v)] \supset V_h^{\delta_0}(s)
\end{aligned}$$

Lemma 1. *For every δ and h, the formulas in $V_h^\delta(s)$ and $P_h^\delta(s)$ partition the state space.*

Proof. (Sketch) By definition the formulas in $rCase(s)$ and $pCase_j^A(\vec{x},s)$ partition the state space. The operations on case statements used in the definition of $V_h^\delta(s)$ $P_h^\delta(s)$ retain this property. □

4 Semantics

Informally speaking, the semantics for the kind of programs we consider is given by the optimal h-step execution of the program. Formally, it is defined by means of the macro $BestDo^+(\delta,s,h,\rho)$ where δ is the program for which a h-step policy ρ in situation s shall be computed. A policy is a special kind of program that is intended to be directly handed over to the execution system of the agent and executed without further deliberation. A policy for a program δ "implements" a (h-step) execution strategy for δ: it resolves the nondeterminism in δ and considers the possible outcomes of stochastic actions. In particular, it may proceed differently depending on what outcome actually has been chosen by Nature.

The macro $BestDo^+(\delta,s,h,\rho)$ is defined inductively on the structure of δ. Its definition is in parts quite similar to that of DTGolog's $BestDo$ which is why we do not present all cases here, but focus on those where the definitions differ. Clearly, if h equals zero the horizon has been reached and the execution of δ is terminated. This is denoted by the special action $Stop$.

$$BestDo^+(\delta,0,s,\rho) \overset{def.}{=} \rho = Stop$$

If the program begins with a stochastic action α a policy for every possibly outcome $n_1,\ldots,n_k$ is determined by means of the auxiliary macro $BestDoAux^+$ which expects a list of deterministic outcome actions as its first argument. $senseEffect_\alpha$ is the sense action associated with α.

$$\begin{aligned}
&BestDo^+([\alpha;\delta],h,s,\rho) \overset{def.}{=} \\
&\quad \exists\rho'.\, \rho = [\alpha; senseEffect_a; \rho'] \\
&\qquad\qquad \wedge BestDoAux^+(\{n_1,\ldots,n_k\},h,s,\rho')
\end{aligned}$$

If the first argument, the list of outcome actions, is empty then $BestDoAux^+$ expands to

$$BestDoAux^+(\{\},h,s,\rho) \overset{def.}{=} \rho = Stop.$$

Otherwise, the first action n_1 of the list is extracted and (if it is possible) a policy for the remaining program starting in the situation $do(n_1,s)$ is computed by $BestDo^+$. Then, the policy is assembled by branching over the sense outcome condition θ_1 for outcome n_1. The if-branch is determined by $BestDo^+(\delta,do(n_i,s),h-1),\rho_1)$; the else branch by the $BestDoAux^+$ macro for the remaining outcome actions.

$$\begin{aligned}
&BestDoAux^+(\{n_1,\ldots,n_k\},h,s,\rho) \overset{def.}{=} \\
&\quad \neg Poss(n_1,s) \wedge BestDoAux^+(\{n_2,\ldots,n_k\},h,s,\rho) \\
&\quad \vee Poss(n_1,s) \\
&\qquad \wedge \exists\rho'.\, BestDoAux^+(\{n_2,\ldots,n_k\}),h,s,\rho') \\
&\qquad \wedge \exists\rho_1.\, BestDo(\delta,h-1,do(n_1,s),\rho_1) \\
&\qquad \wedge \rho = \textbf{if } \theta_1 \textbf{ then } \rho_1 \textbf{ else } \rho'
\end{aligned}$$

The cases where the program begins with a test-action or a conditional are handled in a quite similar manner by DTGolog's $BestDo$ which is why we omit them here. The cases where the program begins with a nondeterministic statement are handled quite differently, though. Whereas $BestDo$ computes the expected reward as well as the probability of successfully executing the remaining program for the current situation, $BestDo^+(\delta,s,h,\rho)$ relies on $V_h^\delta(s)$ and $P_h^\delta(s)$ for that. If the program begins with a nondeterministic branching another auxiliary macro is necessary:

$$\begin{aligned}
&BestDo^+((\delta_1 \mid \delta_2);\delta,s,h,\rho) \overset{def.}{=} \\
&\qquad BestDoNDet((\delta_1 \mid \delta_2);\delta,s,h, \\
&\qquad\qquad case[\phi_i(s),(v_i,p_i) \rightarrow idx_i],\rho)
\end{aligned}$$

where the forth argument of $BestDoNDet$, the case statement, is the result obtained from applying the casemax-operator on $\left(V_h^{\delta_1;\delta}(s) \circ P_h^{\delta_1;\delta}(s)\right) \cup_{\geq} \left(V_h^{\delta_2;\delta}(s) \circ P_h^{\delta_2;\delta}(s)\right)$ where $\geq$ implies an ordering over tuples (v_i,p_i) and implements the trade-off between the expected reward and the probability of successfully executing the program. The

$BestDoNDet$-macro then is defined as:

$$\begin{aligned}&BestDoNDet((\delta_1 \mid \delta_2); \delta, s, h,\\ &\qquad case[\phi_i(s), (v_i, p_i) \rightarrow idx_i], \rho) \overset{def.}{=}\\ &\qquad\qquad \bigvee_i \phi_i(s) \wedge BestDo^+(\delta_{idx_i}; \delta, s, h, \rho)\end{aligned}$$

According to Lemma 1 exactly one of the $\phi_i(s)$ holds and thus the decision of whether to continue with the policy computed for $\delta_1; \delta$ or for $\delta_2; \delta$ is unambiguous.

If the remaining program begins with a nondeterministic choice of arguments the definition of $BestDo^+$ again relies on an auxiliary macro $BestDoPick$:

$$\begin{aligned}&BestDo^+(\pi\, x.\, (\gamma); \delta, s, h, \rho) \overset{def.}{=}\\ &\qquad BestDoPick(\pi\, x.\, (\gamma); \delta, s, h, V_h^{\gamma;\delta}(s) \circ P_h^{\gamma;\delta}(s), \rho)\end{aligned}$$

The definition of $BestDoPick(\pi\, x.\, (\gamma); \delta, s, h, case[\phi_i(x, s), (v_i, p_i)], \rho)$ resembles the operation method of the casemax-operator. We assume that the $\phi_i(x, s)$ are sorted such that $(v_i, p_i) \geq (v_{i+1}, p_{i+1})$. Then:

$$\begin{aligned}&BestDoPick(\pi x.(\gamma); \delta, s, h, case[\phi_i(x, s), (v_i, p_i)], \rho) \overset{def.}{=}\\ &\qquad \bigvee_i \bigwedge_{j<i} \neg\exists x.\, \phi_j(x, s)\\ &\qquad\qquad \wedge \exists x.\, [\phi_i(x, s) \wedge BestDo^+(\gamma; \delta, s, h, \rho)]\\ &\qquad \vee \bigwedge_i \neg\exists x.\, \phi_i(x, s) \wedge \rho = Stop\end{aligned}$$

Note that the existential quantifier over the the ϕ_i also ranges over the macro $BestDo^+$ and thus the x which occurs as a free variable in the policy returned by $BestDo^+(\gamma; \delta, s, h, \rho)$ is bound by the existential such that $\phi_i(x, s)$ holds.

Theorem 1. *For any DTGolog program δ,*

$$\begin{aligned}&\mathcal{D} \models \forall\rho.\, \exists p, v.\, BestDo(\delta, h, S_0, p, v, \rho)\\ &\qquad\qquad \equiv BestDo^+(\delta, h, S_0, \rho)\end{aligned}$$

(We assume that all restricted nondeterministic choices of arguments in δ have been rewritten as nondeterministic branchings.)

There seems to be an anomaly in the definition of DTGolog's $BestDo$-macro. Whereas for primitive actions the reward obtained in the situation before the primitive action is executed is considered this is not the case for stochastic actions. For instance, let A be a primitive, deterministic action and B a stochastic action with A being its sole outcome action (which is chosen by Nature with a probability of 1). Then the expected rewards for executing A and B may be different which seems to be strange. This anomaly can easily be "fixed" by considering the reward obtained in a situation before a stochastic action is executed:

$$\begin{aligned}&BestDo([\alpha; \delta], s, h, \rho, v, pr) \overset{def.}{=}\\ &\qquad \exists\rho', v'.\, BestDoAux(\{n_1, \ldots; n_k\}, \delta, s, h, \rho', v', pr)\\ &\qquad \wedge v = reward(s) + v' \wedge \rho = [\alpha; senseEffect_\alpha; \rho']\end{aligned}$$

For the proof of Theoream 1 we assumed the definition of $BestDo$ as shown above.

5 Comparison with DTGolog

The major difference between the original DTGolog and our extended version of DTGolog is that the latter allows for an unrestricted nondeterministic choice of arguments. DTGolog, on the other hand, only allows the agent to choose from a finite, pre-defined list of possibilities. The practical implications of this are that the programs are tailored to specific domain instantiations—allowing the agent to choose between the blocks B_1, B_2, and B_3, for instance, only makes sense if there are blocks with those names. On the other hand there might be other blocks than those three and and in that case limiting the choice to B_1, B_2, and B_3 is not always what is intended. In our extension of DTGolog the choice is, in general, unrestricted. Any intended restriction on the choice can be implemented by including a corresponding test in the program. For instance, if the programmer wants the agent to choose a green block and do something with it she would write

$$\pi\, x.\, (?(green(x)); \cdots); \cdots$$

Our approach can handle the unrestricted choice of arguments due to the (first-order) state- as well as action-abstraction that is achieved by means of the case-statements $V_h^\delta(s)$ and $P_h^\delta(s)$. The consequence thereof is that the branching factor of the search tree spanned by $BestDo^+$ depends on the number of cases in $V_h^\delta(s)$ and $P_h^\delta(s)$ and not on the number of (ground) choices given to the agent as it is the case for $BestDo$. Although DTGolog is not limited to finite domains it is not capable of incorporating infinitely many ground choices into its decisions. Our extension can do so due to its abstraction mechanisms. On the other hand the formulas in the case statements can get large, actually even unmanageable large, quite quickly. But one can argue that the complexity of the formulas resulting form expanding the $BestDo$-macro is comparable.

To see how this turns out in practise and whether we can even achieve a speed-up in the computation of a policy in comparison to DTGolog we performed tests in two different domains. The first domain is a Blocks World domain. The purpose of this domain is to test how the number of available blocks affects the time it takes to compute a policy. The second domain is the logistics domain which consists of several cities, trucks, and boxes which can be transported from one city to another. Here the goal is to see how DTGolog with and without action abstraction compare with each other in a domain that is a little bit more complex than the Blocks World domain.

Both interpreters, the standard DTGolog interpreter and our extended version, have been implemented in Prolog in a straightforward manner. The only optimization we employed is to represent the case statements $V_h^\delta(s)$ and $P_h^\delta(s)$ using first-order arithmetic decision diagrams (FOADD) as it has been suggested in [Sanner and Boutilier, 2009]. All experiments were carried out on a machine with a 2.6 GHz Core 2 duo and 2 GB of RAM.

5.1 Blocks World

In our instances of the Blocks World there are coloured blocks. The fluent $On(b_1, b_2, s)$ denotes that block b_1 is on

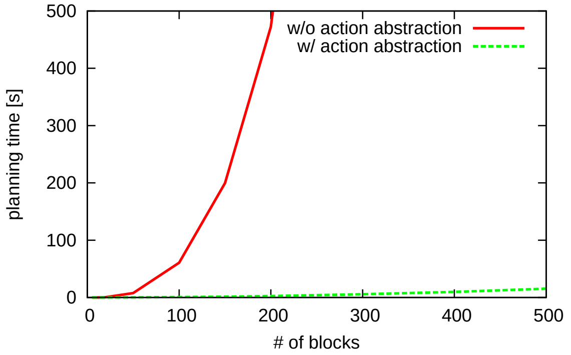

Figure 1: Influence of the number of possible action arguments on the planning time of DTGolog (w/o action abstraction) and our extended version featuring action abstraction.

top of block b_2 in situation s. Predicates like $green(b)$ or $blue(b)$ encode the colouring of the blocks. The stochastic action $move(b_1, b_2)$ has two outcomes: if it succeeds block b_1 is moved on top of block b_2 or if it fails block b_1 remains in its current location. The probability with which the $move$-action succeeds or fails depends on whether the block to be moved is heavy or not; the probability that the action fails is higher if the block is heavy. The reward function assigns a reward of 10 to situations where there exists a green block on top of a non-green block. In the experiments the number of blocks varied between 10 and 500 and we computed policies with $BestDo$ and $BestDo^+$ for a program that nondeterministically picks two blocks and moves one of them on top of the other:

$$\pi\, x.\, (\pi\, y.\, (move(x, y)))$$

In the DTGolog variant of this program the nondeterministic choices of action arguments ranges over all the blocks in the domain instance. Consequently, the search tree branches over all possible combinations for the two blocks. With action abstraction the branching factor of the search tree is constant and independent of the number of objects. This reflects in the time it takes to compute a policy (cf. Figure 1). With an increasing number of blocks the computation time for DTGolog rises exponentially whereas the computation time for our extended version of DTGolog with action abstraction remains nearly constant. The slow increase of computation time can be explained with the fact that the evaluation of quantified formulas as they appear in the case statements for the program above takes longer with an increasing number of objects. Nevertheless in comparison to the computation time of DTGolog this increase is negligible.

5.2 Logistics Domain

In a second experiment we compared our version of DTGolog with action abstraction to the original DTGolog version in the logistics domain. In that domain trucks are supposed to transport boxes from one city to another. The world is described by the fluents $boxIn(b, c)$ meaning that box b is in city c, $boxOn(b, t)$ meaning that box b is on truck t, and $truckIn(t, c)$ meaning that the truck t is in city c. In our setting there are five cities and the intended goal is to have a box in city C_1. The program we computed policies for is:

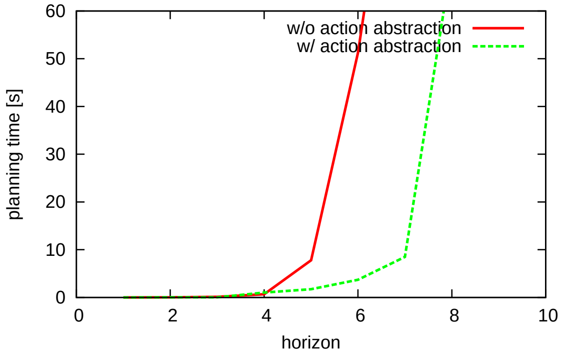

Figure 2: Planning times for different horizons.

```
**while** $\neg\exists b.\, boxIn(b, C_1)$ **do**
        $\pi\, c.\, (drive(T_1, c))$;
        $\pi\, b.\, (load(b, T_1))$;
        $drive(T_1, C_1)$;
        $\pi\, b.\, (unload(b, T_1))$
**end**
```

This time we intend to examine whether we gain any advantage in terms of being able to compute longer policies in the same amount of time from using our extended version of DTGolog. Thus we recorded the computation time for planning policies of different lengths. The results are shown in Figure 2. It can be seen that with action abstraction policies that are two steps longer can be computed in the same time. It has to be noted though that there are also domains in which no advantage in terms of computation time can be gained from the abstraction we apply in our extended version of DTGolog. One such example is a slight variation of the aforementioned logistics domain. In the version above every city is directly reachable from every other city. If we restrict this it is necessary to explicitly encode the reachability in the domain description. This not only increases the complexity of the formulas in $V_h^\delta(s)$ and $P_h^\delta(s)$ but in particular it leads to formulas with more deeply nested quantifiers. This in turn increases the time it takes to evaluate those formulas (at least with our rather unsophisticated implementation) by such an amount that in the end DTGolog is a faster by a little bit. Additionally, we did not precompute the case statements but computed them on-the-fly since the required computation time was marginal.

To sum it up, these experiments show that our extension of DTGolog can lead to a tremendous speed-up in planning. Such a speed-up should be observable in domains/domain instances where the branching factor of the search tree can be drastically reduced due to state and action abstraction. But then, there are also domains where this is not possible and there the speed-up is modest or our extended version of DTGolog is even slower than the original version.

6 Related Work

There are numerous approaches that aim for a compact representation of MDPs by using representation languages of varying expressiveness (e.g., a probabilistic variant of STRIPS [Dearden and Boutilier, 1997], relational logic [Kersting *et al.*, 2004], or first-order logic as in DTGolog [Boutilier *et al.*, 2000]). These representations adhere to abstract representations of the states and the state transitions and transition probabilities, respectively. The next step then is to exploit those compact representations when solving the MDP which is exactly what we did here for DTGolog's first-order MDP representation. The technique we use for that was first presented in [Boutilier *et al.*, 2001] where it was shown how an abstract representation of the value function can be derived from a first-order description of an MDP. The main difference to our approach is that they do not consider programs to restrict the search for the optimal policy. A first approach that combines abstract value functions and Golog programs was presented in [Finzi and Lukasiewicz, 2007]. Contrary to our approach they assume an incompletely specified model (in particular the probabilistic distribution of Nature's choice is unspecified) and apply Q-learning techniques to obtain an optimal policy. The update of the Q-values and the assembling of the policy, though, is not handled within the language. Furthermore, their approach does not incorporate action abstraction.

Restricting the search space for the optimal policy by means of partial programs has been explored extensively in [Parr and Russell, 1997], [Andre and Russell, 2000], and [Andre and Russell, 2002], for instance. Some of these approaches include abstraction mechanisms, too, but these rely manual intervention since the partial programs used by them have no properly defined semantics.

7 Conclusion

In this paper we presented an extension to DTGolog that allows to decision-theoretically determine an execution strategy for a given program with abstracting over action instances. This not only allows to heighten the expressiveness of the programs since we are no longer limited to the restricted nondeterministic choice as in DTGolog. Additionally, we have shown that also in practise this can lead to a speed-up in the computation of the policy despite the complexity of the formulas that has to be dealt with to achieve action abstraction. Nevertheless, for more complex domains or larger horizons the formulas still become unmanageable large, even with the ADD representation. Consequently, subject of future research will be to explore how the complexity of the formulas can be minimized. One possible approach is to approximate the value function by a linear combination of weighted basis functions. The problem with this is that it is not clear how to find basis functions that allow for a good approximation in the context of programs.

References

[Andre and Russell, 2000] D. Andre and S. Russell. Programmable reinforcement learning agents. In *Advances in Neural Information Processing Systems 13 (NIPS 2000)*, pages 1019–1025, 2000.

[Andre and Russell, 2002] D. Andre and S. J. Russell. State abstraction for programmable reinforcement learning agents. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence (AAAI-02)*, pages 119–125, 2002.

[Boutilier *et al.*, 2000] C. Boutilier, R. Reiter, M. Soutchanski, and S. Thrun. Decision-theoretic, high-level agent programming in the situation calculus. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI-00)*, pages 355–362, 2000.

[Boutilier *et al.*, 2001] C. Boutilier, R. Reiter, and B. Price. Symbolic dynamic programming for first-order MDPs. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-01)*, volume 17, pages 690–700, 2001.

[Dearden and Boutilier, 1997] R. Dearden and C. Boutilier. Abstraction and approximate decision-theoretic planning* 1. *Artificial Intelligence*, 89(1-2):219–283, 1997.

[Finzi and Lukasiewicz, 2007] A. Finzi and T. Lukasiewicz. Adaptive multi-agent programming in GTGolog. In *KI 2006: Advances in Artificial Intelligence, 29th Annual German Conference on AI*, pages 389–403. Springer, 2007.

[Kersting *et al.*, 2004] K. Kersting, M.V. Otterlo, and L. De Raedt. Bellman goes relational. In *Proceedings of the twenty-first international conference on Machine learning*, page 59. ACM, 2004.

[Levesque *et al.*, 1997] H. J. Levesque, R. Reiter, Y. Lespérance, F. Lin, and R. B. Scherl. GOLOG: A logic programming language for dynamic domains. *The Journal of Logic Programming*, 31(1-3):59–83, 1997.

[Marthi *et al.*, 2005] B. Marthi, S. Russell, D. Latham, and C. Guestrin. Concurrent hierarchical reinforcement learning. In *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI-05)*, pages 1652–1653, 2005.

[Parr and Russell, 1997] R. Parr and S. Russell. Reinforcement learning with hierarchies of machines. In *Advances in Neural Information Processing Systems 10 (NIPS 1997)*, pages 1043–1049, 1997.

[Puterman, 1994] M. L. Puterman. *Markov decision processes: Discrete stochastic dynamic programming*. John Wiley & Sons, Inc. New York, NY, USA, 1994.

[Reiter, 1991] R. Reiter. The frame problem in situation the calculus: a simple solution (sometimes) and a completeness result for goal regression. *Artificial intelligence and mathematical theory of computation: papers in honor of John McCarthy*, pages 359–380, 1991.

[Sanner and Boutilier, 2009] S. Sanner and C. Boutilier. Practical solution techniques for first-order MDPs. *Artificial Intelligence*, 173(5-6):748–788, 2009.

Verifying properties of action theories by bounded model checking

Laura Giordano
Dipartimento di Informatica
Università del Piemonte
Orientale, Italy

Alberto Martelli
Dipartimento di Informatica
Università di Torino
Italy

Daniele Theseider Dupré
Dipartimento di Informatica
Università del Piemonte
Orientale, Italy

Abstract

Temporal logics are well suited for reasoning about actions, as they allow for the specification of domain descriptions including temporal constraints as well as for the verification of temporal properties of the domain. In this paper, we exploit bounded model checking (BMC) techniques in the verification of properties of an action theory formulated in a temporal extension of *answer set programming*. To achieve completeness, in this paper, we follow an approach to BMC which exploits the Büchi automaton construction. The paper provides an encoding in ASP of the temporal action domain and of bounded model checking of LTL formulas.

1 Introduction

Temporal logics are well suited for reasoning about actions, as they allow for the specification of domain descriptions including temporal constraints as well as for the verification of temporal properties of the domain. In this paper, we exploit bounded model checking (BMC) techniques in the verification of properties of an action theory formulated in a temporal extension of *answer set programming* (ASP [10]).

Given a system model (a transition system) and a property to be checked, bounded model checking (BMC) [4] searches for a counterexample of the property by looking for a path in the model. BMC does not require a tableau or automaton construction. It searches for a counterexample as a path of length k and generates a propositional formula that is satisfiable iff such a counterexample exists. The bound k is iteratively increased and if no model exists, the iterative procedure will never stop. As a consequence, bounded model checking (as defined in [4]) provides a partial decision procedure for checking validity. Techniques for achieving completeness have been described in [4], where upper bounds for k are defined for some classes of properties, namely unnested properties. To solve this problem [5] proposes a *semantic* translation scheme, based on Büchi automata.

Helianko and Niemelä [18] developed a compact encoding of bounded model checking of LTL formulas as the problem of finding stable models of logic programs. In this paper, we propose an alternative encoding of BMC of LTL formulas in ASP, with the aim of achieving completeness. As a difference with [18], the computed path is built by exploiting the Büchi automaton construction [14]: it is an accepting path of the product Büchi automaton which can be finitely represented as a k-loop, i.e., a finite path of length k terminating in a loop back in which the states are all distinct from each other. In the verification of a given property, the iterative procedure looks for a k-loop which provides a counterexample to the property by increasing k until either a counterexample is found, or no k-loop of length greater or equal to k can be found. The second condition can be verified by checking that there is no path of length k whose states are all distinct from each other.

In the paper, the transition system defining the system model on which the property is to be checked is provided by defining a domain description in a temporal action theory. The action theory is given in a temporal extension of ASP and the extensions of a domain description are defined by generalizing the standard notion of *answer set* [10] to temporal answer sets. The encoding of BMC in ASP is based on the definition of the Büchi automaton in [19] and exploits the tableau-based procedure in [15] to provide the construction on-the-fly of the automaton. The tableau procedure is directly encoded in ASP to build a path of the product automaton. The encoding in ASP uses a number of ground atoms which is linear in the size of the formula and quadratic in k.

2 Linear Time Temporal Logic

In this paper we refer to a formulation of LTL (linear time temporal logic), introduced in [19], where the next state modality is indexed by actions.

Let Σ be a finite non-empty alphabet. The members of Σ are actions. Let Σ^* and Σ^ω be the set of finite and infinite words on Σ. Let Σ^∞ $=\Sigma^* \cup \Sigma^\omega$. We denote by σ, σ' the words over Σ^ω and by τ, τ' the words over Σ^*. Moreover, we denote by $\leq$ the usual prefix ordering over Σ^* and, for $u \in \Sigma^\infty$, we denote by *prf(u)* the set of finite prefixes of u.

Let $\mathcal{P} = \{p_1, p_2, \ldots\}$ be a countable set of atomic propositions. The set of formulas of LTL(Σ) is defined as follows:

$$\text{LTL}(\Sigma) ::= p \mid \neg\alpha \mid \alpha \vee \beta \mid \langle a \rangle \alpha \mid \alpha \mathcal{U} \beta$$

where $p \in \mathcal{P}$ and α, β range over LTL(Σ).

A model of LTL(Σ) is a pair $M = (\sigma, V)$ where $\sigma \in \Sigma^\omega$ and $V : prf(\sigma) \rightarrow 2^{\mathcal{P}}$ is a valuation function. Given a model $M = (\sigma, V)$, a finite word $\tau \in prf(\sigma)$ and a formula α, the satisfiability of a formula α at τ in M, written $M, \tau \models \alpha$, is defined as follows:

- $M, \tau \models p$ iff $p \in V(\tau)$;
- $M, \tau \models \neg\alpha$ iff $M, \tau \not\models \alpha$;
- $M, \tau \models \alpha \vee \beta$ iff $M, \tau \models \alpha$ or $M, \tau \models \beta$;
- $M, \tau \models \langle a \rangle \alpha$ iff $\tau a \in \mathit{prf}(\sigma)$ and $M, \tau a \models \alpha$.
- $M, \tau \models \alpha \mathcal{U} \beta$ iff there exists τ' such that $\tau\tau' \in \mathit{prf}(\sigma)$ and $M, \tau\tau' \models \beta$. Moreover, for every τ'' such that $\varepsilon \leq \tau'' < \tau'$[1], $M, \tau\tau'' \models \alpha$.

A formula α is satisfiable iff there is a model $M = (\sigma, V)$ and a finite word $\tau \in \mathit{prf}(\sigma)$ such that $M, \tau \models \alpha$.

The symbols $\top$ and $\bot$ can be defined as: $\top \equiv p \vee \neg p$ and $\bot \equiv \neg\top$. The derived modalities $[a]\alpha$, $\bigcirc$ (next), $\Diamond$ and $\Box$ can be defined as follows: $[a]\alpha \equiv \neg\langle a \rangle \neg\alpha$, $\bigcirc\alpha \equiv \bigvee_{a \in \Sigma} \langle a \rangle \alpha$, $\Diamond\alpha \equiv \top\mathcal{U}\alpha$, $\Box\alpha \equiv \neg\Diamond\neg\alpha$.

3 Temporal action language

A *domain description* Π is defined as a set of laws describing the effects of actions and their executability preconditions. Atomic propositions describing the state of the domain are called *fluents*. Actions may have direct effects, that are described by action laws, and indirect effects, described by causal laws capturing the causal dependencies among fluents.

Let $\mathcal{L}$ be a first order language which includes a finite number of constants and variables, but no function symbol. Let $\mathcal{P}$ be a set of atomic literals $p(t_1, \ldots, t_n)$, that we call *fluent names*. A *simple fluent literal* l is a fluent name f or its negation $\neg f$. We denote by Lit_S the set of all simple fluent literals. Lit_T is the set of *temporal fluent literals*: if $l \in Lit_S$, then $[a]l, \bigcirc l \in Lit_T$, where a is an action name (an atomic proposition, possibly containing variables), and $[a]$ and $\bigcirc$ are the temporal operators introduced in the previous section. Let $Lit = Lit_S \cup Lit_T \cup \{\bot\}$, where $\bot$ represents inconsistency. Given a (simple or temporal) fluent literal l, *not* l represents the default negation of l. A (simple or temporal) fluent literal possibly preceded by a default negation, will be called an *extended fluent literal*.

The laws are formulated as rules of a temporally extended logic programming language. Rules have the form

$$t_0 \leftarrow t_1, \ldots, t_m, not\ t_{m+1}, \ldots, not\ t_n \qquad (1)$$

where the t_i's are either simple fluent literals or temporal fluent literals. As usual in ASP, the rules with variables will be used as a shorthand for the set of their ground instances.

In the following, to define our action language, we make use of a notion of *state*: a set of ground fluent literals. A state is said to be *consistent* if it is not the case that both f and $\neg f$ belong to the state, or that $\bot$ belongs to the state. A state is said to be *complete* if, for each fluent name $p \in \mathcal{P}$, either p or $\neg p$ belong to the state. The execution of an action in a state may possibly change the values of fluents in the state through its direct and indirect effects, thus giving rise to a new state.

We assume that a law as (1) can be applied in all states, while we prefix a law with **Init** if it only applies to the initial state.

Example 1 This example describes a mail delivery agent, which checks if there is mail in the mailbox of some employees and delivers the mail to them. The actions in Σ are: $sense\_mail$ (the agent verifies if there is mail in all mailboxes), $deliver(E)$ (the agent delivers the mail to employee E),$wait$. The fluent names are $mail(E)$ (there is mail in the mailbox of E). The domain description Π contains the following immediate effects and persistency laws:

$$\begin{array}{l} [deliver(E)]\neg mail(E) \\ [sense\_mail]mail(E) \leftarrow\ not\ [sense\_mail]\neg mail(E) \\ \bigcirc mail(E) \leftarrow mail(E), not \bigcirc \neg mail(E) \\ \bigcirc \neg mail(E) \leftarrow \neg mail(E), not \bigcirc mail(E) \end{array}$$

Their meaning is (in the order) that: after delivering the mail to E, there is no mail for E any more; the action $sense\_mail$ may (non-monotonically) cause $mail(E)$ to become true. The last two rules define the persistency of fluent $mail$.

Observe that, the persistency laws interact with the immediate effect laws above. The execution of $sense\_mail$ in a state in which there is no mail for some E ($\neg mail(E)$), may either lead to a state in which $mail(E)$ holds (by the second action law) or to a state in which $\neg mail(E)$ holds (by the persistency of $\neg mail(E)$). Thus $sense\_mail$ is a nondeterministic action.

We can also add the following precondition laws:

$$\begin{array}{l} [deliver(E)]\ \bot \leftarrow \neg mail(E) \\ [wait]\ \bot \leftarrow mail(E) \end{array}$$

specifying that, if there is no mail for E, $deliver(E)$ is not executable, while, if there is mail for E, $wait$ is not executable.

We assume that there are only two employees, a and b, and that in the initial state there is neither mail for a nor for b:
Init $\neg mail(a)$
Init $\neg mail(b)$.
are included in Π.

Although not included in the example, the language is also well suited to describe causal dependencies among fluents by means of *static causal laws* such as, for instance, $light\_on \leftarrow voltage$ (if there is voltage, the light is on), or *dynamic causal laws* as (form the shooting domain) $\bigcirc frightened \leftarrow \bigcirc in\_sight, \neg in\_sight, alive$ (if the turkey is alive, it *becomes* frightened, if it is not already, when it *starts* seeing the hunter). Similar causal rules can be formulated in the action languages $\mathcal{K}$ [9] and $\mathcal{C}^+$ [16].

3.1 Temporal answer sets

To define the the semantics of a domain description, we extend the notion of *answer set* [10] to capture the linear structure of temporal models. In the following, we consider the ground instantiation of the domain description Π, and we denote by Σ the set of all the ground instances of the action names in Π.

We define a temporal interpretation as a pair (σ, S), where $\sigma \in \Sigma^\omega$ is a sequence of actions and S is a consistent set of literals of the form $[a_1; \ldots; a_k]l$, where $a_1 \ldots a_k$ is a prefix of σ, meaning that l holds in the state obtained by executing $a_1 \ldots a_k$. S is *consistent* iff it is not the case that both $[a_1; \ldots; a_k]l \in S$ and $[a_1; \ldots; a_k]\neg l \in S$, for some l, or

[1] We define $\tau \leq \tau'$ iff $\exists\tau''$ such that $\tau\tau'' = \tau'$. Moreover, $\tau < \tau'$ iff $\tau \leq \tau'$ and $\tau \neq \tau'$.

$[a_1; \ldots; a_k]\bot \in S$. A temporal interpretation (σ, S) is said to be *total* if either $[a_1; \ldots; a_k]p \in S$ or $[a_1; \ldots; a_k]\neg p \in S$, for each $a_1 \ldots a_k$ prefix of σ and for each fluent name p.

We define the *satisfiability of a simple, temporal or extended literal* t *in a partial temporal interpretation* (σ, S) *in the state* $a_1 \ldots a_k$, (written $(\sigma, S), a_1 \ldots a_k \models t$) as follows:

$(\sigma, S), a_1 \ldots a_k \models \top, \qquad (\sigma, S), a_1 \ldots a_k \not\models \bot$
$(\sigma, S), a_1 \ldots a_k \models l$ *iff* $[a_1; \ldots; a_k]l \in S$, for a literal l
$(\sigma, S), a_1 \ldots a_k \models [a]l$ *iff* $[a_1; \ldots; a_k; a]l \in S$ or $a_1 \ldots a_k, a$ is not a prefix of σ
$(\sigma, S), a_1 \ldots a_k \models \bigcirc l$ *iff* $[a_1; \ldots; a_k; b]l \in S$, where $a_1 \ldots a_k b$ is a prefix of σ
$(\sigma, S), a_1 \ldots a_k \models not\ l$ *iff* $(\sigma, S), a_1 \ldots a_k \not\models l$

The satisfiability of rule bodies in a temporal interpretation are defined as usual. A rule $H \leftarrow Body$ is satisfied in a temporal interpretation (σ, S) if, for all action sequences $a_1 \ldots a_k$ (including the empty one), $(\sigma, S), a_1 \ldots a_k \models Body$ implies $(\sigma, S), a_1 \ldots a_k \models H$.

A rule **Init** $H \leftarrow Body$ is satisfied in a partial temporal interpretation (σ, S) if, $(\sigma, S), \varepsilon \models Body$ implies $(\sigma, S), \varepsilon \models H$, where ε is the empty action sequence.

To define the answer sets of Π, we introduce the notion of *reduct* of Π, containing rules of the form: $[a_1; \ldots; a_h](H \leftarrow Body)$. Such rules are evaluated in the state $a_1 \ldots a_h$.

Let Π be a set of rules over an action alphabet Σ, not containing default negation, and let $\sigma \in \Sigma^\omega$.

Definition 1 *A temporal interpretation* (σ, S) *is a* temporal answer set of Π *if* S *is minimal (in the sense of set inclusion) among the* S' *such that* (σ, S') *is a partial interpretation satisfying the rules in* Π.

To define answer sets of a program Π containing negation, given a temporal interpretation (σ, S) over $\sigma \in \Sigma^\omega$, we define the *reduct,* $\Pi^{(\sigma,S)}$, *of* Π *relative to* (σ, S) extending Gelfond and Lifschitz' transform [11] to compute a different reduct of Π for each prefix $a_1, \ldots, a_h$ of σ.

Definition 2 *The* reduct, $\Pi^{(\sigma,S)}_{a_1,\ldots,a_h}$, of Π relative to (σ, S) and to the prefix $a_1, \ldots, a_h$ of σ , *is the set of all the rules*

$$[a_1; \ldots; a_h](H \leftarrow l_1, \ldots, l_m)$$

such that $H \leftarrow l_1, \ldots, l_m, not\ l_{m+1}, \ldots, not\ l_n$ *is in* Π *and* $(\sigma, S), a_1, \ldots, a_h \not\models l_i$, *for all* $i = m+1, \ldots, n$.

The reduct $\Pi^{(\sigma,S)}$ of Π relative to (σ, S) *is the union of all reducts* $\Pi^{(\sigma,S)}_{a_1,\ldots,a_h}$ *for all prefixes* $a_1, \ldots, a_h$ *of* σ.

Definition 3 *A temporal interpretation* (σ, S) *is an* answer set of Π *if* (σ, S) *is an answer set of the reduct* $\Pi^{(\sigma,S)}$.

Although the answer sets of a domain description Π are partial interpretations, in some cases, e.g., when the initial state is complete and all fluents are inertial, it is possible to guarantee that the temporal answer sets of Π are total.

In case the initial state is not complete,we consider all the possible ways to complete the initial state by introducing in Π, for each fluent name f, the rules:

Init $f \leftarrow not\ \neg f$
Init $\neg f \leftarrow not\ f$

The case of total temporal answer sets is of special interest as a total temporal answer set (σ, S) can be regarded as temporal model (σ, V), where, for each finite prefix $a_1 \ldots a_k$ of σ, $V(a_1, \ldots, a_k) = \{p : [a_1, \ldots, a_k]p \in S\}$. In the following, we restrict our consideration to domain descriptions Π, such that all the answer sets of Π are total.

A total temporal interpretation (σ, S) provides, for each prefix $a_1 \ldots a_k$, a complete state corresponding to that prefix. We denote by $w^{(\sigma,S)}_{a_1 \ldots a_k}$ the state obtained by the execution of the actions $a_1 \ldots a_k$ in the sequence, namely $w^{(\sigma,S)}_{a_1 \ldots a_k} = \{l : [a_1; \ldots; a_k]l \in S\}$.

Given a domain description Π over Σ with total answer sets, a *transition system* (W, I, T) can be associated with Π as follows:

- W is the set of all the possible consistent and complete states of the domain description;
- I is the set of all the states in W satisfying the initial state laws in Π;
- $T \subseteq W \times \Sigma \times W$ is the set of all triples (w, a, w') such that: $w, w' \in W$, $a \in \Sigma$ and for some total answer set (σ, S) of Π: $w = w^{(\sigma,S)}_{[a_1;\ldots;a_h]}$ and $w' = w^{(\sigma,S)}_{[a_1;\ldots;a_h;a]}$, for some h.

3.2 Reasoning with LTL on domain descriptions

As a total temporal answer set of a domain description can be interpreted as an LTL model, it is easy to combine domain descriptions with LTL formulas. This can be done in two ways: on the one hand, LTL formulas can be used as constraints $\mathcal{C}$ on the executions of the domain description; on the other hand, LTL formulas can encode properties ϕ to be verified on the domain description.

Example 2 Assume we want to constrain our domain description in Example 1 so that the agent continuously executes a loop where it senses mail, but there cannot be two consecutive executions of $sense\_mail$. These constraints can be formulated as follows:

$\Box\Diamond\langle sense\_mail\rangle\top$
$\Box[sense\_mail]\neg\langle sense\_mail\rangle\top$

Furthermore, we may want to check that, if there is mail for a, the agent will eventually deliver it to a. This property, which can be formalized as $\Box(mail(a) \supset \Diamond\neg mail(a))$, does not hold as there is a possible scenario in which there is always mail for a and for b, but the mail is repeatedly delivered to b and never to a. The mail delivery agent we have described is not correct with respect to this property.

In the following, we will assume that a set of constraints $\mathcal{C}$ is added to the domain description, beside the rules in Π, and we denote by $(\Pi, \mathcal{C})$ the enriched domain description. We define the *extensions of* $(\Pi, \mathcal{C})$ to be the temporal answer sets of Π satisfying the constraints $\mathcal{C}$.

4 Model checking

The above verification and satisfiability problems can be solved by means of *model checking* techniques. Given a *domain description*, with its associated transition system, the extension of the domain description satisfying a set of constraints $\mathcal{C}$ can be found by looking for a path in the transition system satisfying the formulas in $\mathcal{C}$. On the other hand, given a property φ formulated as a LTL formula, we can check its validity by checking the unsatisfiability of $\neg\varphi$ in the transition system. In this case, if a model satisfying $\neg\varphi$ is found, it represents a counterexample to the validity of φ.

The standard approach to model checking for LTL is based on Büchi automata. A *Büchi automaton* over an alphabet Σ is a tuple $\mathcal{B} = (Q, \to, Q_{in}, F)$ where: Q is a finite nonempty set of states; $\to \subseteq Q \times \Sigma \times Q$ is a transition relation; $Q_{in} \subseteq Q$ is the set of initial states; $F \subseteq Q$ is a set of accepting states.

Let $\sigma \in \Sigma^{\omega}$; a *run* of $\mathcal{B}$ over σ is a map $\rho : \mathit{prf}(\sigma) \to Q$ such that: $\rho(\varepsilon) \in Q_{in}$ and $\rho(\tau) \xrightarrow{a} \rho(\tau a)$ for each $\tau a \in \mathit{prf}(\sigma)$ with $a \in \Sigma$. The run ρ is *accepting* iff $\mathit{inf}(\rho) \cap F \neq \emptyset$, where $\mathit{inf}(\rho) \subseteq Q$ is given by: $q \in \mathit{inf}(\rho)$ iff $\rho(\tau) = q$ for infinitely many $\tau \in \mathit{prf}(\sigma)$. The language of ω-words accepted by $\mathcal{B}$ is: $\mathcal{L}(\mathcal{B}) = \{\sigma | \exists$ an accepting run of $\mathcal{B}$ over $\sigma\}$.

The satisfiability problem for LTL can be solved in deterministic exponential time by constructing for each formula $\alpha \in LTL(\Sigma)$ a Büchi automaton $\mathcal{B}_{\alpha}$ [14] such that the language of ω-words accepted by $\mathcal{B}_{\alpha}$ is non-empty if and only if α is satisfiable. In case of *model checking* we have a property which is represented as an LTL formula φ, and a model (transition system) which directly corresponds to a Büchi automaton where all the states are accepting. The property can be proved by taking the product of the model and of the automaton derived from $\neg\varphi$, and by checking for emptiness of the accepted language.

In [4] it has been shown that, in some cases, model checking can be more efficient if, instead of building the product automaton and checking for an accepting run on it, we look for an infinite path of the transition system satisfying $\mathcal{C} \cup \{\neg\varphi\}$. This technique is called *bounded model checking* (BMC), since it looks for paths whose length is bounded by some integer k, by iteratively increasing the length k until a model satisfying the formulas in $\mathcal{C} \cup \{\neg\varphi\}$ is found (if one exists). More precisely, it considers infinite paths which can be represented as a finite path of length k with a back loop, i.e. with an edge from state k to a previous state in the path.

A BMC problem can be efficiently reduced to a propositional satisfiability problem or to an ASP problem [18]. Unfortunately, if no model exists, the iterative procedure will never stop, if the transition system contains a loop. Thus it is a partial decision procedure for checking validity. Techniques for achieving completeness are described in [4] for some kinds of LTL formulas.

5 Büchi automata and Model checking

In this paper, we propose an approach which combines the advantages of BMC and the possibility of formulating it easily and efficiently as an ASP problem, with the advantages of reasoning on the product Büchi automaton described above, mainly its completeness. In this section, we show how to build the product automaton and how to use the automaton tableau construction for BMC. In the next section we describe how to encode the transition system and BMC in ASP.

The problem of constructing a Büchi automaton from a LTL formula has been deeply studied. In this section we show how to build a Büchi automaton for a given LTL(Σ) formula ϕ using the tableau-like procedure. The construction is adapted from the procedure given in [19; 15] for Dynamic Linear Time Logic (DLTL), a logic which extends LTL by indexing the until operator with regular programs.

The main procedure to construct the Büchi automaton for a formula ϕ builds a graph $\mathcal{G}(\phi)$ whose nodes are labelled by sets of formulas, and whose edges are labelled by symbols from the alphabet Σ. States and transitions of the Büchi automaton are obtained directly from the nodes and edges of the graph. The construction of the states makes use of an auxiliary tableau-based function *tableau* which handles *signed formulas*, i.e. formulas prefixed with the symbol **T** or **F**. This function takes as input a set of formulas[2] and returns a set of sets of formulas, obtained by expanding the input set according to a set of tableau rules, formulated as follows:

- $\phi \Rightarrow \psi_1, \psi_2$, if ϕ belongs to the set of formulas, then add ψ_1 and ψ_2 to the set
- $\phi \Rightarrow \psi_1 | \psi_2$, if ϕ belongs to the set of formulas, then replace the set with two copies of the set and add ψ_1 to one of them and ψ_2 to the other one.

The rules are the following:

Tor:	$\mathbf{T}(\alpha \vee \beta) \Rightarrow \mathbf{T}\alpha | \mathbf{T}\beta$
For:	$\mathbf{F}(\alpha \vee \beta) \Rightarrow \mathbf{F}\alpha, \mathbf{F}\beta$
Tneg:	$\mathbf{T}\neg\alpha \Rightarrow \mathbf{F}\alpha$
Fneg:	$\mathbf{F}\neg\alpha \Rightarrow \mathbf{T}\alpha$
Tuntil:	$\mathbf{T}\alpha\mathcal{U}\beta \Rightarrow \mathbf{T}(\beta \vee (\alpha \wedge \bigcirc\alpha\mathcal{U}\beta))$
Funtil:	$\mathbf{F}\alpha\mathcal{U}\beta \Rightarrow \mathbf{F}(\beta \vee (\alpha \wedge \bigcirc\alpha\mathcal{U}\beta))$

where the tableau rules for the until formula make use of the equivalence: $\alpha\mathcal{U}\beta \equiv (\beta \vee (\alpha \wedge \bigcirc\alpha\mathcal{U}\beta))$. This set of rules can be easily extended to deal with other boolean connectives and modal operators like $\Box$ or $\Diamond$ by making use of the equivalences $\Box\beta \equiv (\beta \wedge \bigcirc\Box\beta))$ and $\Diamond\beta \equiv (\beta \vee \bigcirc\Diamond\beta))$.

Given a set of formulas s, function *tableau* repeatedly applies the above rules to the formulas of s (by possibly creating new sets) until all formulas in all sets have been expanded. If the expansion of a set of formulas produces an inconsistent set, then this set is deleted. A set of formulas s is *inconsistent* in the following cases: (i) $\mathbf{T}\bot \in s$; (ii) $\mathbf{F}\top \in s$; (iii) $\mathbf{T}\alpha \in s$ and $\mathbf{F}\alpha \in s$; (iv) $\mathbf{T}\langle a\rangle\alpha \in s$ and $\mathbf{T}\langle b\rangle\beta \in s$ with $a \neq b$, because in a linear time logic two different actions cannot be executed in the same state.

To build the graph for a formula ϕ, we begin by building the initial states, obtained by applying function *tableau* to the set $\{\phi\ ,\ \mathbf{T}\bigvee_{a\in\Sigma}\langle a\rangle\top\}$, where the second formula takes into account the fact that runs must be infinite and thus there must be at least an outgoing edge from each state. After execution of *tableau*, every resulting set contains exactly one $\mathbf{T}\langle a\rangle\top$ formula, for some $a \in \Sigma$.

The above tableau rules do not expand formulas whose top operator is a *next time operator*, i.e. $\langle a\rangle\alpha$ or $\bigcirc\alpha$. Expanding such formulas from a node n means creating a new node containing α connected to n through an edge labelled with a

[2] In this section "formulas" means "signed formulas".

in the first case, or with any symbol in Σ in the second case. Thus an obvious procedure for building the graph is to apply to all sets obtained by the tableau procedure the following construction: if node n contains a formula $\mathbf{T}\langle a\rangle\alpha$, then build the set of the nodes connected to n through an edge labelled a as $tableau(\{\mathbf{T}\alpha|\mathbf{T}\langle a\rangle\alpha \in n\} \cup \{\mathbf{T}\alpha|\mathbf{T} \bigcirc \alpha \in n\} \cup \{\mathbf{F}\alpha|\mathbf{F}\langle a\rangle\alpha \in n\} \cup \{\mathbf{F}\alpha|\mathbf{F}\bigcirc\alpha \in n\} \cup \{\mathbf{T}\bigvee_{a\in\Sigma}\langle a\rangle\top\})$. The construction is iterated on the new nodes.

States and transitions of the Büchi automaton correspond directly to the nodes and edges of the graph. We must now define the *accepting states* of the automaton. Intuitively, we would like to define as accepting those runs in which all the until formulas of the form $\mathbf{T}\alpha\mathcal{U}\beta$ are fulfilled. If a node n contains the formula $\mathbf{T}\alpha\mathcal{U}\beta$, then we can accept an infinite run containing n, if node n is followed in the run by a node n' containing $\mathbf{T}\beta$. Furthermore all nodes between n and n' must contain $\mathbf{T}\alpha$.

Let us assume that a node n contains the until formula $\mathbf{T}\alpha\mathcal{U}\beta$. After the expansion of this formula, n either contains $\mathbf{T}\beta$ or $\mathbf{T} \bigcirc \alpha\mathcal{U}\beta$. In the latter case, each successor node will contain a formula $\mathbf{T}\alpha\mathcal{U}\beta$. We say that this until formula is *derived* from formula $\mathbf{T}\alpha\mathcal{U}\beta$ in node n. If a node contains an until formula which is not derived from a predecessor node, we will say that the formula is *new*. New until formulas are obtained during the expansion of the *tableau* procedure.

In order to formulate the accepting condition, we must be able to trace the until formulas along the paths of the graph to make sure that they are fulfilled. To do this we extend $\mathbf{T}$-signed formulas so that all until formulas have a label 0 or 1, i.e. they have the form $\mathbf{T}\alpha\mathcal{U}^l\beta$ where $l \in \{0,1\}$[3]. Note that two formulas $\mathbf{T}\alpha\mathcal{U}^0\beta$ and $\mathbf{T}\alpha\mathcal{U}^1\beta$ are considered to be different. Furthermore, we define each node of the graph as a triple $(\mathcal{F}, x, f)$, where $\mathcal{F}$ is an expanded set of formulas built by function *tableau*, $x \in \{0,1\}$, and $f \in \{\downarrow, \checkmark\}$. $f = \checkmark$ means that the node represents an accepting state.

For each node $(\mathcal{F}, x, f)$, the label of an until formula in $\mathcal{F}$ will be assigned as follows: if it is a derived until formula, then its label is the same as that of the until formula in the predecessor node it derives from, otherwise, if the formula is new, it is given the label $1 - x$.

Given a node $(\mathcal{F}, x, f)$ and a successor $(\mathcal{F}', x', f')$, x' and f' are defined as follows:

if $f = \checkmark$ **then** $x' := 1 - x$ **else** $x' := x$,

if there is no $\mathbf{T} \bigcirc \alpha\mathcal{U}^{x'}\beta \in \mathcal{F}'$ **then** $f' := \checkmark$ **else** $f' :=\downarrow$

Let us call *0-sequences* or *1-sequences* the sequences of nodes of a run ρ with $x = 0$ or $x = 1$ respectively. Intuitively, every new until formula created in a node of a 0-sequence will be fulfilled within the end of the next 1-sequence, and vice versa. In fact, the formula will be given label 1 and propagated in the following nodes with the same label, and the 1-sequence cannot terminate until the until formula is fulfilled. If ρ is an *accepting run*, then it must contain infinitely many nodes containing $\checkmark$, and thus all 0-sequences and 1-sequences must be finite and, as a consequence, all until formulas will be fulfilled.

Given a graph $\mathcal{G}(\phi)$, the states and transitions of the Büchi automaton $\mathcal{B}(\phi)$ correspond directly to the nodes and edges of $\mathcal{G}(\phi)$, and the set of accepting states of $\mathcal{B}(\phi)$ consists of all states whose corresponding node contains $f = \checkmark$.

In [15] it is proved that there is a $\sigma \in \mathcal{L}(\mathcal{B}(\phi))$ if and only if there is a model $M = (\sigma, V)$ such that $M, \varepsilon \models \phi$.

The same construction can be used in model checking for building the product automaton of $\mathcal{B}(\phi)$ and the transition system. Every state of the product automaton is the union of a set of fluents forming a state of the transition system and a set of signed formulas corresponding to a state of $\mathcal{B}(\phi)$, while transitions must agree both with transitions of $\mathcal{B}(\phi)$ and those of the action theory. We assume that the action theory and the LTL formulas refer to the same set of actions and atomic propositions. Of course, the states of the product automaton must be consistent, i.e. they cannot contain the literal $\neg f$ and the signed formula $\mathbf{T}f$ or f and $\mathbf{F}f$[4].

The construction of the automaton can be done on-the-fly, while checking for the emptiness of the language accepted by the automaton. In this paper, following the BMC approach, we aim at generating a single path of the automaton at a time. Given an integer k, we look for a path of length k of the automaton, with a loop back from the last state to a previous state l in the path, such that there is an accepting state j, $l \leq j \leq k$. Such a k-loop finitely represents an accepting run of the automaton. Note that we can consider only *simple* paths, that is paths without repeated nodes. This property allows to define a terminating algorithm, thus achieving *completeness*: the bound k is increased until a k-loop is found or the length of the longest path of the automaton is reached.

To find the length of the longest path we can proceed iteratively by looking for a simple path of length k (without loop), incrementing k at each iteration. Since the product automaton has a finite size, this procedure terminates.

Example 3 Let us consider the domain description in Example 1 with the constraint $\Box\Diamond\langle sense\_mail\rangle\top$. The following is a k-loop satisfying the constraint for $k = 4$. It consist of the states $s_0, \ldots, s_4$ with the transitions $s_0 \xrightarrow{wait} s_1$, $s_1 \xrightarrow{wait} s_2$, $s_2 \xrightarrow{sense\_mail} s_3$, $s_3 \xrightarrow{deliver(b)} s_4$, $s_4 \xrightarrow{deliver(a)} s_1$.

State s_0 is obtained by applying *tableau* to the LTL formula expressing the constraint, and adding the fluent literals holding in the initial state. Thus we get[5]:

$\mathbf{T}\Box\Diamond\langle sense\_mail\rangle\top$, $\mathbf{T}\Diamond^1\langle sense\_mail\rangle\top$,
$\mathbf{T} \bigcirc \Box\Diamond\langle sense\_mail\rangle\top$, $\mathbf{T} \bigcirc \Diamond^1\langle sense\_mail\rangle\top$,
$\mathbf{T}\langle wait\rangle\top$, $\neg a$, $\neg b$, $x = 0$, $f = \checkmark$

The second and third formulas are obtained by the expansion of the first one, while the fourth formula is obtained by the expansion of the second one.

State s_1 is obtained by propagating the next time formulas and expanding them:

$\mathbf{T}\Box\Diamond\langle sense\_mail\rangle\top$, $\mathbf{T}\Diamond^1\langle sense\_mail\rangle\top$,
$\mathbf{T}\Diamond^0\langle sense\_mail\rangle\top$, $\mathbf{T} \bigcirc \Box\Diamond\langle sense\_mail\rangle\top$,
$\mathbf{T} \bigcirc \Diamond^1\langle sense\_mail\rangle\top$, $\mathbf{T} \bigcirc \Diamond^0\langle sense\_mail\rangle\top$,
$\mathbf{T}\langle wait\rangle\top$, $\neg a$, $\neg b$, $x = 1$, $f =\downarrow$

[3] If we introduce also the $\Box$ and $\Diamond$ operators, we have to label them in the analogous way.

[4] Remember that the states of the transition systems are complete and thus each state must contain either f or $\neg f$

[5] We omit formulas having as topmost operator a boolean connective, and we use a and b as a shorthand for $mail(a)$, $mail(b)$.

The second and third formulas are identical but the index of the $\Diamond$ operator: the second formula derives from the previous state, while the third one derives from the first formula of this state; f is $\downarrow$ because there is a next time formula with label 1.

State s_2 is:
$\mathbf{T}\Box\Diamond\langle sense\_mail\rangle\top$, $\mathbf{T}\Diamond^1\langle sense\_mail\rangle\top$,
$\mathbf{T}\Diamond^0\langle sense\_mail\rangle\top$, $\mathbf{T}\bigcirc\Box\Diamond\langle sense\_mail\rangle\top$,
$\mathbf{T}\langle sense\_mail\rangle\top$, $\neg a$, $\neg b$, $x = 1$, $f = \checkmark$

The value of f is $\checkmark$ because there are no next time formulas with label 1. The formulas $\mathbf{T}\Diamond^l\langle sense\_mail\rangle\top$ are fulfilled because $sense\_mail$ will be the next action.

State s_3 is:
$\mathbf{T}\Box\Diamond\langle sense\_mail\rangle\top$, $\mathbf{T}\Diamond^1\langle sense\_mail\rangle\top$,
$\mathbf{T}\bigcirc\Box\Diamond\langle sense\_mail\rangle\top$, $\mathbf{T}\bigcirc\Diamond^1\langle sense\_mail\rangle\top$,
$\mathbf{T}\langle deliver\_mail(b)\rangle\top$, a, b, $x = 0$, $f = \checkmark$

Note that the execution of $sense\_mail$ changes the value of a and b.

State s_4 is:
$\mathbf{T}\Box\Diamond\langle sense\_mail\rangle\top$, $\mathbf{T}\Diamond^0\langle sense\_mail\rangle\top$,
$\mathbf{T}\Diamond^1\langle sense\_mail\rangle\top$, $\mathbf{T}\bigcirc\Box\Diamond\langle sense\_mail\rangle\top$,
$\mathbf{T}\bigcirc\Diamond^1\langle sense\_mail\rangle\top$, $\mathbf{T}\bigcirc\Diamond^0\langle sense\_mail\rangle\top$,
$\mathbf{T}\langle deliver\_mail(a)\rangle\top$, a, $\neg b$, $x = 1$, $f = \downarrow$

By executing action $deliver\_mail(a)$ we have a transition back to state s_1.

Example 4 Let us consider now our domain description with the two constraints in Example 2. To check whether the formula $\varphi = \Box(mail(a) \supset \Diamond\neg mail(a))$ is valid, we add to the domain description the two constraints and $\neg\varphi$, and we get the following k-loop which represent a counterexample to the property: $s_0 \xrightarrow{sense\_mail} s_1$, $s_1 \xrightarrow{deliver(b)} s_2$, $s_2 \xrightarrow{sense\_mail} s_3$, $s_3 \xrightarrow{deliver(b)} s_2$. Furthermore, we have the following fluents in each state: $s_0 : \neg a, \neg b$, $s_1 : a, b$, $s_2 : a, \neg b$, $s_3 : a, b$. Thus the mail of a is never delivered.

Let us now modify the domain theory by adding the precondition $[sense\_mail] \perp \leftarrow mail(E)$. In this case, we expect φ to hold. To check this, we first compute the length of the longest path in the Büchi automaton, which turns out to be 9, and then check that there is no k-loop for k up to 9.

6 Encoding bounded model checking in ASP

We give now a translation into standard ASP of the above procedure for building a path of the product Büchi automaton. The translation has been run in the DLV-Complex extension of DLV [20].

In the translation we use predicates like `fluent`, `action`, `state`, to express the type of atoms.

As we are interested in infinite runs represented as k-loops, we assume a bound K to the number of states. States are represented in ASP as integers from 0 to K, where K is given by the predicate `laststate(State)`. The predicate `occurs(Action,State)` describes transitions. Occurrence of exactly one action in each state can be encoded as:

```
-occurs(A,S):- occurs(A1,S),action(A),
   action(A1),A!=A1,state(S).
occurs(A,S):- not -occurs(A,S),action(A),
   state(S).
```

As we have seen, states are associated with a set of fluent literals, a set of signed formulas, and the values of x and f. Fluent literals are represented with the predicate `holds(Fluent, State)`, **T** or **F** formulas with `tt(Formula,State)` or `ff(Formula,State)`, x with the predicate `x(Val,State)` and f with the predicate `acc(State)`, which is true if `State` is an accepting state.

States on the path must be all different, and thus we need to define a predicate `eq(S1,S2)` to check whether the two states $S1$ and $S2$ are equal:

```
eq(S1,S2):- state(S1), state(S2),
   not diff(S1,S2).
diff(S1,S2):- state(S1), state(S2),
   tt(F,S1), not tt(F,S2).
diff(S1,S2):- state(S1), state(S2),
   holds(F,S1), not holds(F,S2).
```

and similarly for any other kind of component of a state.

The following constraint requires all states up to K to be different:

```
:- state(S1), state(S2), S1!=S2, eq(S1,S2),
   laststate(K), S1<=K, S2<=K.
```

Furthermore we have to define suitable constraints stating that there will be a transition from state K to a previous state L[6], and that there must be a state S, $L \leq S \leq K$, such that `acc(S)` holds, i.e. S is an accepting state. To do this we compute the successor of state K, and check that it is equal to S.

```
loop(L):- state(L), laststate(K), L<=K,
   SuccK=K+1, eq(L,SuccK).
accept:- loop(L), state(S), laststate(K),
    L<=S, S<=K, acc(S).
:- not accept.
```

A problem we want to translate to ASP consists of a domain description Π and a set of LTL formulas $\varphi_1, \ldots \varphi_n$, representing constraints or negated properties, to be satisfied on the domain description. The rules of the domain description can be easily translated to ASP, similarly to [10]. In the following we give the translation of our running example[7].

```
action(sense_mail).
action(deliver(a)).
action(deliver(b)).
action(wait).
fluent(mail(a)).
fluent(mail(b)).
```

action effects:

```
holds(mail(E),NS):- occurs(sense_mail,S),
   fluent(mail(E)), NS=S+1,
   not -holds(mail(E),NS).
-holds(mail(E),NS):- occurs(deliver(E),S),
   fluent(mail(E)), NS=S+1.
```

persistence:

```
holds(F,NS):- holds(F,S), fluent(F), NS=S+1,
```

[6] Since states are all different, there will be at most one state equal to the successor of K.

[7] Actually, a more general approach to deal with variables in action names and fluents, consists in introducing , as done in [8], type predicates for fluents and actions and to include type conditions in the translation.

```
   not -holds(F,NS).
holds(F,NS):- -holds(F,S), fluent(F), NS=S+1,
   not holds(F,NS).
```

preconditions:

```
:- occurs(deliver(E),S),-holds(mail(E),S).
:- occurs(wait,S), holds(mail(E),S).
```

initial state:

```
-holds(mail(a),0).  -holds(mail(b),0).
```

LTL formulas are represented as ASP terms. The expansion of signed formulas can be formulated by means of ASP rules corresponding to the rules given in the previous section.

Disjunction:

```
tt(F1,S) v tt(F2,S):- tt(or(F1,F2),S).
ff(F1,S):- ff(or(F1,F2),S).
ff(F2,S):- ff(or(F1,F2),S).
```

Negation:

```
ff(F,S):- tt(neg(F),S).
tt(F,S):- ff(neg(F),S).
```

Until:

```
tt(lab_until(F1,F2,Lab),S):-
   tt(until(F1,F2),S), x(VX,S), 1=Lab+VX.
ff(or(F2,and(F1,next(until(F1,F2)))),S):-
   ff(until(F1,F2),S).
tt(or(F2,and(F1,next(lab_until(F1,F2,L)))),S):-
   tt(lab_until(F1,F2,L),S).
```

Note that, to express splitting of sets of formulas, as in the case of disjunction, we can exploit disjunction in the head of clauses, provided by some ASP languages such as DLV. We have introduced the term `lab_until(F1,F2,Lab)` for labeled until formulas, as described in the previous section.

Expansions of next time formulas $\langle a \rangle$ (diamond) and $\bigcirc$ (next) are defined as:

```
occurs(Act,S):- tt(diamond(Act,F),S).
tt(F,NS):- tt(diamond(Act,F),S), NS=S+1.
ff(F,NS):- ff(diamond(Act,F),S),
   occurs(Act,S), NS=S+1.
tt(F,NS):- tt(next(F),S), NS=S+1.
ff(F,NS):- ff(next(F),S), NS=S+1.
```

Inconsistency of signed formulas is formulated with the following constraints:

```
:- ff(true,S), state(S).
:- tt(F,S), ff(F,S), state(S).
:- tt(diamond(Act1,F),S),
   tt(diamond(Act2,F),S), Act1!=Act2.
:- tt(F,S), not holds(F,S).
:- ff(F,S), not -holds(F,S).
```

Finally, predicates x and acc are defined as follows.

```
x(NN,NS):- acc(S), x(N,S), NS=S+1, 1=NN+N.
x(N,NS):- -acc(S), x(N,S), NS=S+1.
-acc(NS):- x(N,NS), tt(lab_until(_,_,N),NS),
   NS=S+1.
acc(NS):- not -acc(NS), NS=S+1.
x(0,0).  acc(0).
```

We must also add a fact `tt(`$tr(\varphi_i)$`,0)` for each φ_i, where $tr(\varphi_i)$ is the ASP term representing φ_i.

It is easy to see that the (groundization of the) encoding in ASP is linear in the size of the formula and quadratic in the size of k. Observe that the number of the ground instances of all predicates is $O(|\phi| \times k)$, except for eq, whose ground instances are k^2.

We can prove that there is a one to one correspondence between the extensions of a domain description satisfying a given temporal formula and the answer sets of the ASP program encoding the domain and the formula.

Proposition 1 *Let Π be a domain description whose temporal answer sets are total, let $tr(\Pi)$ be the ASP encoding of Π (for a given k), and let ϕ be an LTL formula. There is a one to one correspondence between the temporal answer sets of Π that satisfy the formula ϕ and the answer sets of the ASP program $tr(\Pi) \cup tt(tr(\phi, 0))$, where $tr(\phi)$ is the ASP term representing ϕ.*

Completeness of BMC can be achieved considering that that the longest simple path in the product Büchi automaton determines an upper bound k_0 on the length of the k-loops searched for by the iterative procedure. To check the validity of a formula ϕ, we look for a k-loop satisfying $\neg\phi$. During the iterative search, we can check whether the bound k_0 has been already reached or not. In practice, at each iteration k, either we find a k-loop of length k, or we check if there is a simple path (a path with no loop and without repeated nodes) of length k. If not, we can conclude that the bound has been reached and we can stop.

The search for a simple path of length k can be done by removing from the above ASP encoding the rules for defining loops and the rules for defining the Buchi acceptance condition (the definitions of `x`, `acc` and `accept` and the constraint `:- not accept`).

7 Conclusions

We have presented a bounded model checking approach for the verification of properties of temporal action theories in ASP. The temporal action theory is formulated in a temporal extension of ASP, where the presence of LTL constraints in the domain description, allows for state trajectory constraints to be captured, as advocated in PDDL3 [13]. The proposed approach can be easily extended to the logic DLTL, which extends LTL with regular programs of propositional dynamic logic [19]. It provides a uniform ASP methodology for specifying domain descriptions and for verifying them, which can to be used for several reasoning tasks, including reasoning about communication protocols [2; 15], business process verification [7], planning with temporal constraints [1], to mention some of them.

Helianko and Niemelä [18] developed a compact encoding of bounded model checking of LTL formulas as the problem of finding stable models of logic programs. In this paper, to achieve completeness, we follow a different approach to BMC which exploits the Büchi automaton construction. This makes the proposed approach well suited both for verifying that there is an extension of the domain description satisfying/falsifying a given property, and for verifying that all the extensions of the domain description satisfy a given property.

[5] first proposed the use of the Büchi automaton in BMC. As a difference, our encoding in ASP is defined without assuming that the Büchi automaton is computed in advance. The states of the Büchi automaton are indeed computed on

the fly, when building the path of the product automaton. This requires the equality among states to be checked during the construction of k-loops, which makes the size of the translation quadratic in k. This quadratic blowup is the price we pay for achieving completeness with respect to the translation to stable models in [18].

Apart from the presence of the temporal constraints, the action language we introduced in Section 3 has strong relations with the languages $\mathcal{K}$ and $\mathcal{C}$. The logic programming based planning language $\mathcal{K}$ [8; 9] is well suited for planning under incomplete knowledge and which allows concurrent actions. The temporal action language introduced in section 3 for defining the rules in Π can be regarded as a fragment of $\mathcal{K}$ in which concurrent actions are not allowed.

The planning system $DLV^{\mathcal{K}}$ provides an implementation of $\mathcal{K}$ in the disjunctive logic programming system DLV. $DLV^{\mathcal{K}}$ does not appear to support other kinds of reasoning besides planning, and, in particular, does not allow to express temporal properties and to verify them.

The languages $\mathcal{C}$ and $\mathcal{C}^+$ [17; 16] also deal with actions with indirect and non-deterministic effects and with concurrent actions, and are based on nonmonotonic causation rules syntactically similar to those of $\mathcal{K}$, where head and body of causation rules can be boolean combination of constants. Their semantics is based on a nonmonotonic causal logic [16]. Due to the different semantics, a mapping between our action language and the languages $\mathcal{C}$ and $\mathcal{C}^+$ appears not to be straightforward. If a causal theory is definite (the head of a rule is an atom), it is possible to reason about it by turning the theory into a set of propositional formulas by means of a completion process, and then invoke a satisfiability solver. In this way it is possible to perform various kinds of reasoning such as prediction, postdiction or planning. However the language does not exploits standard temporal logic constructs to reason about actions.

The action language defined in this paper can be regarded as a temporal extension of the language $\mathcal{A}$ [12]. The extension allows to deal with general temporal constraints and infinite computations. Instead, it does not deal with concurrent actions and incomplete knowledge.

The presence of temporal constraints in our action language is related to the work on temporally extended goals in [6; 3], which, however, is concerned with expressing preferences among goals and exceptions in goal specification.

References

[1] F. Bacchus and F. Kabanza. Planning for temporally extended goals. *Annals of Mathematics and AI*, 22:5–27, 1998.

[2] M. Baldoni, C. Baroglio, and E. Marengo. Behavior-oriented Commitment-based Protocols. In *Proc. 19th ECAI*, pages 137–142, 2010.

[3] C. Baral and J. Zhao. Non-monotonic temporal logics for goal specification. In *IJCAI 2007*, pages 236–242, 2007.

[4] A. Biere, A. Cimatti, E. M. Clarke, O. Strichman, and Y. Zhu. Bounded model checking. *Advances in Computers*, 58:118–149, 2003.

[5] E.M. Clarke, D. Kroening, J. Ouaknine, and O. Strichman. Completeness and complexity of bounded model checking. In *VMCAI*, pages 85–96, 2004.

[6] U. Dal Lago, M. Pistore, and P: Traverso. Planning with a language for extended goals. In *Proc. AAAI02*, 2002.

[7] D. D'Aprile, L. Giordano, V. Gliozzi, A. Martelli, G. L. Pozzato, and D. Theseider Dupré. Verifying Business Process Compliance by Reasoning about Actions. In *CLIMA XI*, volume 6245 of *LNAI*, 2010.

[8] T. Eiter, W. Faber, N. Leone, G. Pfeifer, and A. Polleres. A logic programming approach to knowledge-state planning, II: The DLV$^{\text{k}}$ system. *Artificial Intelligence*, 144(1-2):157–211, 2003.

[9] T. Eiter, W. Faber, N. Leone, G. Pfeifer, and A. Polleres. A logic programming approach to knowledge-state planning: Semantics and complexity. *ACM Trans. Comput. Log.*, 5(2):206–263, 2004.

[10] M. Gelfond. *Handbook of Knowledge Representation, chapter 7, Answer Sets*. Elsevier, 2007.

[11] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *Logic Programming, Proc. of the 5th Int. Conf. and Symposium*, 1988.

[12] M. Gelfond and V. Lifschitz. Representing action and change by logic programs. *Journal of logic Programming*, 17:301–322, 1993.

[13] A. Gerevini and D. Long. Plan constraints and preferences in PDDL3. *Technical Report, Department of Electronics and Automation, University of Brescia, Italy*, 2005.

[14] R. Gerth, D. Peled, M.Y.Vardi, and P. Wolper. Simple on-the-fly automatic verification of linear temporal logic. In *Proc. 15th Work. Protocol Specification, Testing and Verification*, 1995.

[15] L. Giordano and A. Martelli. Tableau-based automata construction for dynamic linear time temporal logic. *Annals of Mathematics and AI*, 46(3):289–315, 2006.

[16] E. Giunchiglia, J. Lee, V. Lifschitz, N. McCain, , and H. Turner. Nonmonotonic causal theories. *Artificial Intelligence*, 153(1-2):49–104, 2004.

[17] E. Giunchiglia and V. Lifschitz. An action language based on causal explanation: Preliminary report. In *AAAI/IAAI*, pages 623–630, 1998.

[18] K. Heljanko and I. Niemelä. Bounded LTL model checking with stable models. *TPLP*, 3(4-5):519–550, 2003.

[19] J.G. Henriksen and P.S. Thiagarajan. Dynamic linear time temporal logic. *Annals of Pure and Applied logic*, 96(1-3):187–207, 1999.

[20] N. Leone, G. Pfeifer, W. Faber, T. Eiter, G. Gottlob, S. Perri, and F. Scarcello. The DLV system for knowledge representation and reasoning. *ACM Transactions on Computational Logic*, 7(3):499–562, 2006.

Efficient Epistemic Reasoning in Partially Observable Dynamic Domains Using Hidden Causal Dependencies

Theodore Patkos and Dimitris Plexousakis
Foundation for Research and Technology Hellas - FORTH
Heraklion, Greece
{patkos,dp}@ics.forth.gr

Abstract

Reasoning about knowledge and action in real-world domains requires establishing frameworks that are sufficiently expressive yet amenable to efficient computation. Where partial observability is concerned, contemporary research trends concentrate on alternative representations for knowledge, as opposed to the possible worlds semantics. In this paper we study *hidden causal dependencies*, a recently proposed approach to model an agent's epistemic notions. We formally characterize its properties, substantiate its computational benefits and provide a generic implementation method.

1 Introduction

Research in epistemic action theories has extended the competence of cognitive robotics to different domains. Powerful frameworks with expressive formal accounts for knowledge and change and elegant mathematical specifications have been developed, motivated primarily by the adaptation of the possible worlds semantics in action theories, e.g., [Scherl and Levesque, 2003; Thielscher, 2000; Lob, 2001]. Nevertheless, their employment to large-scale systems raises legitimate concerns, due to their dependence on a computationally intensive structure: according to the possible worlds specifications, for a domain of n atomic formulae determining whether a formula is known may require up to 2^n worlds to check truth in. Aiming at efficiency, contemporary research departs from the standard practice and explores alternative characterizations of knowledge focusing on classes of restricted expressiveness or sacrificing completeness. In many cases, a demand for context-complete domains, i.e., deterministic domains where all action preconditions are known upon action execution, is introduced to prove logical equivalence to possible worlds. Yet, for real-world domains such restrictions are often too strong to accept.

In a previous study we proposed a formal theory for reasoning about knowledge, action and time that can reach sound and complete conclusions with respect to possible worlds-based theories for a wide range of commonsense phenomena [Patkos and Plexousakis, 2009]. One innovative aspect of the approach was the introduction of a generic form of implication rules, called *hidden causal dependency* (HCD), that captures the relation among unknown preconditions and effects of actions. In this paper we uncover the insights of HCDs and present the complete axiomatization. HCDs can be a valuable tool as they offer a way to represent an agent's best knowledge about a partially observable world without having to maintain all alternative states. Setting off from previous studies, we adopt a more concrete representation that does not require the introduction of auxiliary fluents, offering a generic theory that is not attached to a particular underlying formalism. In addition, we elaborate on complexity issues substantiating the claims of efficiency, which stems from the fact that HCDs are treated as ordinary epistemic concepts. As the objective of this research is to contribute to the creation of practical applications without sacrificing expressiveness, this paper also discusses a way to implement the axiomatization and introduces a new tool for epistemic reasoning.

We begin with a presentation of state-of-the-art approaches and, after a brief familiarization with the underlying knowledge theory, we describe in detail the axiomatization concerning HCDs. Section 5 provides a property analysis, while section 6 discusses implementation issues.

2 Related Work

To alleviate the computational intractability of reasoning under the possible worlds specifications, as well as to address other problematic issues, such as the logical omniscience side-effect, alternative approaches for handling knowledge change have been proposed that are disengaged from the accessibility relation. These theories are rapidly moving from specialized frameworks to an important research field, but adopt certain restrictions on the type of knowledge formulae or domain classes that they can support.

Maybe the most influential initial approach towards an alternative formal account for reasoning about knowledge and action is due to Demolombe and Pozos-Parra [2000] who introduced two different *knowledge fluents* to explicitly represent the knowledge that an ordinary fluent is true or false. Working on the Situation Calculus, they treated knowledge change as changing each of these fluents individually, the same way ordinary fluent change is performed in the calculus, thus reducing reasoning complexity by linearly increasing the number of fluents. Nevertheless, the expressive power of the representation was limited to knowledge of literals, while it enforced knowledge of disjunctions to be broken apart into knowledge of the individual disjuncts. Petrick and Levesque [2002] proved the correspondence of this approach to the possible worlds-based Situation Calculus axiomatization for successor state axioms of a restricted form. Moreover, they defined a combined action theory that extended knowledge fluents to also account for first-order formulae when disjunctive knowledge is tautology-free, still enforcing it to be broken apart into knowledge of the individual parts.

Regression used by standard Situation Calculus is considered impractical for large sequences of actions and introduces restrictive assumptions, such as closed-world and domain closure, which are problematic when reasoning with incomplete knowledge. Recent approaches deploy different forms of progression. Liu and Levesque [2005] for instance, study a class of incomplete knowledge that can be represented in so called *proper KBs* and perform progression on them. The idea is to focus on domains where a proper KB will remain proper after progression, so that an efficient evaluation-based reasoning procedure can be applied. Domains where the actions have local effects (i.e., when the properties of fluents that get altered are contained in the action) provide such a safeguard. The approach is efficient and sound for local effect action theories and may also be complete given certain restrictions, still proper KBs under this *weak progression* do not permit some general forms of disjunctions to emerge. Recently, Vassos et al. [Vassos *et al.*, 2009] investigated an extension to theories with incomplete knowledge in the Situation Calculus where the effects are not local and progression is still appropriate for practical purposes.

Dependencies between unknown preconditions and effects have been incorporated in an extension of the FLUX programming language [Thielscher, 2005b] under the Fluent Calculus semantics. The extension, presented in [Thielscher, 2005a], handles dependencies by appending implication constraints to the existing store of constraint handling rules, in a spirit very similar to the HCDs proposed in the present study. The emphasis is on building an efficient constraint solver, thus limiting the expressiveness. Moreover, it is not clear how the extensive set of complicated implication rules that are defined there are related to possible worlds.

Apart from Thielscher's work, a recent study by Forth and Shanahan [2004] is highly related to ours, as they attempt to capture knowledge change as ordinary fluent change. The authors utilized knowledge fluents in the Event Calculus to specify when an agent possesses enough knowledge to execute an action in a partially observable environment. Still, their intention was to handle ramifications, focusing on closed, controlled environments, rather than building a generic epistemic theory for the Event Calculus. An agent is only assumed to perform "safe" actions, i.e., actions for which enough knowledge about its preconditions is available. In an open environment the occurrence of exogenous actions might also fall under the agent's attention, whose effects are dependent on -unknown to it- preconditions. It is not clear how knowledge evolves in terms of such uncertain effects, neither how knowledge about disjunction of fluents can be modeled. Within DECKT we attempt a broader treatment of knowledge evolution within open environments, unifying a wide range of complex commonsense phenomena.

3 Background

This study uses the DECKT knowledge theory [Patkos and Plexousakis, 2009] as an underlying formalism. DECKT extends the Event Calculus [Kowalski and Sergot, 1986] with epistemic features enabling reasoning about a wide range of commonsense phenomena, such as temporal and delayed knowledge effects, knowledge ramifications, concurrency, non-determinism and others. The Event Calculus is a narrative-based many-sorted first-order language for reasoning about action and change, where *events* indicate changes in the environment, *fluents* denote time-varying properties and a *timepoint* sort implements a linear time structure. The calculus applies the *principle of inertia*, which captures the property that things tend to persist over time unless affected by some event; when released from inertia, a fluent may have a fluctuating truth value at each time instant. It also uses *circumscription* to solve the frame problem and support default reasoning. A set of predicates is defined to express which fluents hold when ($HoldsAt$), what events happen ($Happens$), which their effects are ($Initiates$, $Terminates$, $Releases$) and whether a fluent is subject to the law of inertia or released from it ($ReleasedAt$).

DECKT employs the discrete time Event Calculus axiomatization described in [Mueller, 2006]. It assumes agents acting in dynamic environments having accurate but potentially incomplete knowledge and able to perform sensing and actions with context-dependent effects. It uses four new epistemic fluents, namely $Knows$, Kw (for "knows whether"), KP (for "knows persistently") and KPw. The $Knows$ fluent expresses knowledge about domain fluents and formulae. Whenever knowledge is subject to inertia the KP fluent is used that is related with the $Knows$ fluent by the axiom[1]:
(KT2) $HoldsAt(KP(\phi), t) \Rightarrow HoldsAt(Knows(\phi), t)$.
Moreover, knowledge can also be inferred indirectly by means of appropriate ramifications, usually modeled as state constraints. In brief, direct action effects that are subject to inertia affect the KP fluent, while indirect effects and ramifications may interact with the $Knows$ fluent explicitly. Finally, we have that $HoldsAt((Kw(f), t)) \equiv HoldsAt(Knows(f), t) \vee HoldsAt(Knows(\neg f), t)$ (the abbreviation for $HoldsAt(KPw(f), t)$ is analogous)[2].

The objective is to extend a given domain axiomatization with a set of meta-axioms that enable an agent to perform epistemic derivations under incomplete information. For instance, for positive effect axioms that specify under what condition action e causes fluent f to become true, i.e., $\bigwedge^{i} HoldsAt(f_i, t) \Rightarrow Initiates(e, f, t)$, DECKT introduces a statement expressing that if the conjunction of preconditions $C = \{\vec{f_i}\}$ is known then after e the effect will be known:

(KT3.1) $\bigwedge^{f_i \in C}[HoldsAt(Knows(f_i), t)] \wedge$
$Happens(e, t) \Rightarrow Initiates(e, KP(f), t)$

[1] Free variables are implicitly universally quantified. Fluent formulae inside any epistemic fluent are reified, i.e., $Knows(f_1 \wedge f_2)$ is a term of first-order logic, not an atom.

[2] To clarify matters, the abbreviation only refers to the Kw and KPw fluents inside the distinguished predicate $HoldsAt$; these epistemic fluents can still be used as ordinary fluents inside any other predicate of the calculus, e.g., Terminates(e,KPw(f),t).

However, if some precondition is unknown while none is known false, then after e knowledge about the effect is lost:

(KT5.1) $\bigvee^{f_i \in C}[\neg HoldsAt(Kw(f_i), t)] \wedge$
$\neg HoldsAt(Knows(\bigvee^{f_i \in C} \neg f_i), t) \wedge$
$\neg HoldsAt(Knows(f), t) \wedge Happens(e, t) \Rightarrow$
$Terminates(e, KPw(f), t)$

The approach is analogous for negative effect axioms $\bigwedge^i HoldsAt(f_i, t) \Rightarrow Terminates(e, f, t)$ and release axioms $\bigwedge^i HoldsAt(f_i, t) \Rightarrow Releases(e, f, t)$. The latter model non-deterministic effects, therefore they result in loss of knowledge about the effect. Finally, knowledge-producing (sense) actions provide information about the truth value of fluents and, by definition, only affect the agent's mental state:
(KT4) $Initiates(sense(f), KPw(f), t)$

4 Hidden Causal Dependencies

HCDs emerge when an agent performs actions with unknown preconditions. Consider the positive effect axiom $HoldsAt(f', t) \Rightarrow Initiates(e, f, t)$, with f' unknown and f known to be false at t (f may denote that a door is open, f' that a robot stands in front of that door and e the action of pushing forward gently). If e happens at t, f becomes unknown at $t+1$, as dictated by (KT5.1), still a dependency between f' and f must be created to denote that if we later sense any of them we can infer information about the value of the other, assuming no event interacted with them in the meantime (either the robot was standing in front of the door and opened it or the door remained closed). We propose here a representation of HCDs as disjunctive formulae and describe when they are created and destroyed and what knowledge is preserved when a HCD is destroyed.

First, a word about notation. Let C denote the context of an effect axiom (the set of precondition fluents), i.e. $C = \{f_0, ..., f_n\}$, $n \geq 0$ (we omit to specify the axiom it refers to as it will be clear from the context). Let $C(t)^+$ be the subset of known fluents from C at a given time instant t, i.e., $C(t)^+ = \{f \in C | HoldsAt(Knows(f), t)\}$. Finally, let $C(t)^- = C \setminus C(t)^+$ be the set of fluents that the agent either does not know or knows that they do not hold at t.

4.1 Creation of HCDs

Each time an action with unknown effect preconditions occurs, a HCD is created. We assume in this study that no action affects the preconditions at the same time (except, of course, if the effect's preconditions is the effect fluent itself).

Pos. effect axioms $\bigwedge^i HoldsAt(f_i, t) \Rightarrow Initiates(e, f, t)$:
If an action with a positive effect occurs with none of its preconditions known to be false, but some unknown to the agent, a HCD is created between the latter and the effect (by sensing that the robot is standing in front of the door after the *push gently* action, it can infer that the door must be open):

(KT6.1.1) $\neg HoldsAt(Knows(\bigvee^{f_i \in C} \neg f_i), t) \wedge$
$\bigvee^{f_i \in C}[\neg HoldsAt(Kw(f_i), t)] \Rightarrow$
$Initiates(e, KP(f \vee \bigvee^{f_j \in C(t)^-} \neg f_j), t)$

In other words and considering (KT2), we augment the theory with a disjunctive knowledge formula that is equivalent to $HoldsAt(Knows(\bigwedge^{f_j \in C(t)^-} f_j \Rightarrow f), t+1)$.

In addition, a HCD is also created between the effect fluent and its unknown preconditions, given that the agent knew that the effect did not hold before the action:

(KT6.1.2) $\neg HoldsAt(Knows(\bigvee^{f_i \in C} \neg f_i), t) \wedge$
$\bigvee^{f_i \in C}[\neg HoldsAt(Kw(f_i), t)] \wedge HoldsAt(Knows(\neg f), t)$
$\Rightarrow \bigwedge^{f_j \in C(t)^-} Initiates(e, KP(\neg f \vee f_j), t)$

Axiom (KT6.1.2) is triggered with (KT6.1.1), resulting in the creation of an epistemic biimplication relation among the preconditions and the effect fluent.

Example 1. Consider the positive effect axiom $HoldsAt(f_1, t) \wedge HoldsAt(f_2, t) \Rightarrow Initiates(e, f, t)$ denoting that when the robot stands in front of a door (f_1) and the door is not locked (f_2), a gentle push (e) will cause the door to open (f). Let the robot know initially that f does not hold, that f_1 holds but does not know whether f_2 holds, i.e., $HoldsAt(Knows(\neg f), 0) \wedge HoldsAt(Knows(f_1), 0) \wedge \neg HoldsAt(Kw(f_2), 0)$. In this case, $C = \{f_1, f_2\}$, while $C(0)^- = \{f_2\}$. After $Happens(e, 0)$ both (KT6.1.1,2) are triggered resulting in $HoldsAt(KP(\neg f_2 \vee f), 1) \wedge HoldsAt(KP(\neg f \vee f_2), 1)$. This is equivalent to $HoldsAt(Knows(f_2 \Leftrightarrow f), 1)$. □

Neg. effect axioms $\bigwedge^i Holds(f_i, t) \Rightarrow Terminates(e, f, t)$
The situation is similar. Now, the HCDs are created for $\neg f$:

(KT6.1.3) $\neg HoldsAt(Knows(\bigvee^{f_i \in C} \neg f_i), t) \wedge$
$\bigvee^{f_i \in C}[\neg HoldsAt(Kw(f_i), t)] \Rightarrow$
$Initiates(e, KP(\neg f \vee \bigvee^{f_j \in C(t)^-} \neg f_j), t)$

(KT6.1.4) $\neg HoldsAt(Knows(\bigvee^{f_i \in C} \neg f_i), t) \wedge$
$\bigvee^{f_i \in C}[\neg HoldsAt(Kw(f_i), t)] \wedge HoldsAt(Knows(f), t) \Rightarrow$
$\bigwedge^{f_j \in C(t)^-} Initiates(e, KP(f \vee f_j), t)$

Release axioms $\bigwedge^i HoldsAt(f_i, t) \Rightarrow Releases(e, f, t)$:
It is trivial to see that in the case of non-deterministic effects a HCD is only created if the agent has prior knowledge about the effects. Specifically, only if it senses that a previously known effect fluent has changed its truth value will the agent be certain that the preconditions must have been true:

(KT6.1.5) $\neg HoldsAt(Knows(\bigvee^{f_i \in C} \neg f_i), t) \wedge$
$\bigvee^{f_i \in C}[\neg HoldsAt(Kw(f_i), t)] \wedge$
$HoldsAt(Knows((\neg)f), t) \Rightarrow$
$Initiates(e, KP((\neg)f \vee \bigvee^{f_j \in C(t)^-} f_j), t)$

4.2 Expiration of HCDs

In contrast to state constraints that express implication relations that must be satisfied at all times, HCDs are valid only for limited time periods, as they are created due to the agent's epistemic state. Specifically, the dependency is valid for as long as the involved fluents remain unaffected by occurring events; if an event modifies them the HCD expires.

First, let us define an abbreviation stating that an event e affects or *may* affect a fluent f if there is some effect axiom none of whose preconditions $\vec{f_i}$ is known false:
(KmA) $KmAffect(e, f, t) \equiv (KmInitiate(e, f, t) \vee$

$KmTerminate(e,f,t) \vee KmRelease(e,f,t))$
where $KmInitiate(e,f,t) \equiv Initiates(e,f,t) \wedge \neg HoldsAt(Knows(\bigvee^{f_i \in C} \neg f_i),t)$ and similarly for $KmTerminate(e,f,t)$ and $KmRelease(e,f,t)$. These epistemic predicates do not cause any actual effect to f.

HCD Termination

If an event occurs that affects or may affect any fluent of a HCD then this HCD is no longer valid:

(KT6.2.1) $HoldsAt(KP(\bigvee^{d} f_d),t) \wedge$
$Happens(e,t) \wedge \bigvee^{d}[KmAffect(e,f_d,t)] \Rightarrow$
$Terminates(e,KP(\bigvee^{d} f_d),t)$

Example 2. Imagine a robot that speaks the correct passcode into a microphone (action e) with the intention of opening a safe (fluent f), without knowing whether the microphone is recording (fluent f_1); the following HCD is created from (KT6.1.1): $HoldsAt(Knows(\neg f_1 \vee f),t)$. Under this simplistic setting, if later on the robot obtains information through sensing (axiom (KT4)) that the safe is still locked, it will also infer that the device is not recording. Now, at some timepoint $t_1 > t$ the robot becomes aware of an action by a third party that switches the microphone on. At this point the HCD needs to expire, as the epistemic relation among the fluents in no longer valid and no sense action about any of the two fluents can provide information about the other. Axiom (KT6.2.1) accomplishes this effect. □

HCD Reduction

Depending on the type of action and the related context there are situations where although a HCD becomes invalidated due to (KT6.2.1) there may still be knowledge that should be preserved. Specifically, if before the action the agent has inferred indirectly that the fluent that may be affected does not hold, then this fluent did not contribute to the HCD in the first place; the remaining components should create a new HCD:

(KT6.2.2) $HoldsAt(KP(\bigvee^{f \in D} f),t) \wedge Happens(e,t) \wedge$
$\bigvee^{f \in D}[KmAffect(e,f,t) \wedge HoldsAt(Knows(\neg f),t)] \Rightarrow$
$Initiates(e,KP(\bigvee^{f' \in D'(t)} f'),t)$

where if $f \in D$ are the fluents of the initial HCD, then $D'(t)$ denotes those fluents of D that are not known at t.

HCD Expansion

Consider now the particular situation where a context-dependent event occurs, the preconditions of which are unknown to the agent and its effect is part of some HCD. In this case, the agent cannot be certain whether the HCD will be affected by the event, as this depends on the truth value of the preconditions. In fact, the HCD itself becomes contingent on this set; if the preconditions prove to be false, the original HCD should still be applicable, otherwise it must be invalidated, according to the previous analysis. The way to handle this situation is to expand the original HCD with the negation of the action's unknown preconditions. As a result, by obtaining knowledge about them the agent can distinguish whether the original dependency should persist or not.

Example 2. (cont'd) If at timepoint t_2 ($t_1 > t_2 > t$) the robot itself attempts to switch the microphone on under the (unknown to it) precondition of having pressed the proper button (fluent f_2) then, apart from the new HCD $HoldsAt(Knows(\neg f_2 \vee f_1), t_2+1)$ according to (KT6.1.1), the initial HCD needs to be modified. In particular, it should capture the fact that only if the microphone has not been switched on, should the HCD remain valid, i.e., $HoldsAt(Knows(f_2 \vee \neg f_1 \vee f), t_2+1)$. □

It becomes clear that the unknown preconditions of a context-dependent effect should result in the expansion of any HCD that includes the effect. Before modeling this situation though, one must notice a crucial contingency: the agent uses these preconditions to determine whether the original HCD is applicable or not; what if this distinction cannot be made? Such a situation may be, for instance, the result of an action leading to a world state where the precondition fluents have the same truth value regardless of the state before the action (e.g., the action of closing a door if it is open). To capture such a situation we introduce the following abbreviation stating that a fluent may be inverted by an occurring event:

(INV) $KmInverted(f,t) \equiv \exists e(Happens(e,t) \wedge$
$(EffectPredicate(e,f,t) \vee KmRelease(e,f,t)))$
where, for a fluent literal f and its corresponding atom F, $EffectPredicate(e,f,t)$ denotes $KmTerminate(e,F,t)$ when $f = F$, and $KmInitiate(e,F,t)$ when $f = \neg F$. Notice that the $KmInverted$ predicate is completely independent of the truth value a fluent might have at any time instant. For example, for an effect axiom of the form $HoldsAt(f_1,t) \Rightarrow Initiates(e,f,t)$ we are interested whether $KmInverted(f_1,t)$ is true, while for the axiom $\neg HoldsAt(f_1,t) \Rightarrow Initiates(e,f',t)$ we should seek whether $KmInverted(\neg f_1,t)$ holds.

We can now formalize the axiomatization for HCD expansion: for any action e that may initiate, terminate or release a fluent of a HCD, if its unknown preconditions $\vec{f_i}$ are not or may not be inverted, then a new HCD is created that involves all the components of the original HCD along with the unknown preconditions of e's effect axiom:

(KT6.2.3) $HoldsAt(KP(\bigvee^{f \in D} f),t) \wedge Happens(e,t) \wedge$
$\bigvee^{f \in D}[KmAffect(e,f,t) \wedge \neg HoldsAt(Kw(f),t)] \wedge$
$\neg(\bigwedge^{f_i \in C(t)^-}[KmInverted(f_i,t)]) \Rightarrow$
$\bigwedge^{f_i \in C(t)^-}[Initiates(e,KP(f_i \vee \bigvee^{f' \in D'(t)} f'),t)]$

Intuitively, since any HCD represents an epistemic implication relation, axiom (KT6.2.3) creates a nested implication relation with head the HCD and body the negated unknown preconditions of the effect axiom that may affect it.

Transitivity

Finally, we also need to consider the transitivity property of implication relations. Whenever an agent knows that f_1 implies f_2 and f_2 implies f_3 there is an implicit relation stating that also f_1 implies f_3. If an action affects f_2 the two original HCDs will expire due to (KT6.2.1), still the relation between f_1 and f_3 that has been established should persist:

(KT6.2.4) $HoldsAt(Knows(f \vee (\bigvee^{f_i \in D_i} f_i)),t) \wedge$
$HoldsAt(Knows(\neg f \vee (\bigvee^{f_j \in D_j} f_j)),t) \wedge$
$Happens(e,t) \wedge KmAffect(e,f,t) \Rightarrow$
$Initiates(e,KP(\bigvee^{f'_i \in D'_i(t)} f'_i \vee \bigvee^{f'_j \in D'_j(t)} f'_j),t)$

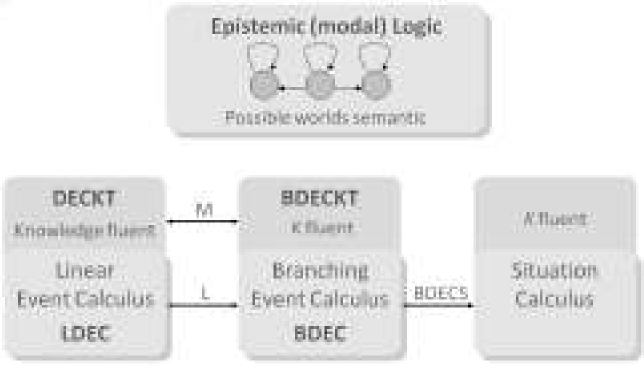

Figure 1: Relation among axiomatic sets.

5 Correctness and Complexity

DECKT has been shown to derive sound and complete inferences with respect to possible worlds-based theories [Patkos and Plexousakis, 2009], based on a correspondence established to an epistemic extension of Mueller's branching Event Calculus (BDEC) [Mueller, 2007]. Exploiting the existing machinery we arrived to the same result about our extension with HCDs, proving that the (KT6) axioms constitute a complete and sufficient set:

Corollary 1 *After any ground sequence of actions with deterministic effects but with potentially unknown preconditions, a fluent formula ϕ is known whether it holds in DECKT if and only if it is known whether it holds in BDECKT, under the bridging set of axioms L and M.*

Proof sketch[3]*:* The branching discrete Event Calculus (BDEC) devised by Mueller is a modified version of the linear discrete Event Calculus (LDEC) (see Figure 1). It replaces the timepoint sort with the sort of situations, lifting the requirement that every situation must have a unique successor state. The Branching Discrete Event Calculus Knowledge Theory (BDECKT) that we have developed follows on from Moore's [1985] formalization of possible world semantics in action theories, where the number of K-accessible worlds remains unchanged upon ordinary event occurrences and reduces as appropriate when sense actions occur. Similar to Scherl and Levesque's [2003] approach for the Situation Calculus , BDECKT generalizes BDEC in that there is no single initial situation in the tree of alternative situations, rather a forest of trees each with its own initial situation.

The DECKT axiomatization is based on the linear Event Calculus that treats knowledge as a fluent and uses a set of axioms to determine the way this epistemic fluent changes its truth value as a result of event occurrences and the knowledge already obtained about relevant context. BDECKT on the other hand is based on a branching time version of the Event Calculus where knowledge is understood as reasoning about the accessibility relation over possible situations (Figure 1). Mueller has established a set L of mapping rules between the underlying linear and branching versions of the Event Calculus and proved that these two versions can be logically equivalent [Mueller, 2007]. The L axioms restrict -among others- BDEC to a linear past. Based on this corollary established by Mueller, we have shown that the two knowledge theories manipulate knowledge change the same way, i.e., the set of known formulae is the same after a sequence of actions (in contrast to the underlying theories, our equivalence result is not an one-to-one mapping of all the axioms). We define a set M that serves as a bridge between DECKT and BDECKT and construct each individual effect axiom of one theory from the axioms of the other and the bridging rules (and vice versa). This way, the conjunction of DECKT, BDEC, LDEC, L and M can provide all BDECKT epistemic derivations leading to completeness with respect to the possible worlds semantics and respectively, the conjunction of BDECKT, BDEC, LDEC, L and M can provide all DECKT epistemic derivations resulting in soundness of DECKT inferences. □

In what follows, we additionally show that reasoning with HCDs is computationally more efficient. It is an important conclusion as it paves the way for practical applications of knowledge representation without substantial sacrifice in expressiveness. The objective is to study the complexity of checking whether a fluent formula holds after the occurrence of an event sequence in total chronological order, given a domain theory comprising a fixed set of context-dependent effect axioms and a set of implication rules (either in the form of state constraints or in the form of HCDs).

5.1 Classic Event Calculus Without Knowledge

For n domain fluents there are potentially 2^n distinct knowledge bases (KBs) (when all n fluents are released) that need to be progressed according to occurring events and at most n $HoldsAt()$ and n $ReleasedAt()$ predicates to search through for each KB. All predicates are stored as facts:

Algorithm: *For each event e_i occurring at t_i and each KB*

1. *Retrieve all effect axioms of e_i: $\bigwedge^j [HoldsAt(f_j, t)] \Rightarrow \theta(e_i, f', t)$?, for θ = $Initiates, Terminates, Releases$*
   This information is already known at design time. Therefore, it requires constant time, regardless of the type of action, the number of effect axioms or the size of the domain (number of fluents).
2. *Query the KB for the truth value of the precondition fluents of e_i: $\bigwedge^j [HoldsAt(f_j, t_i)]$?*
   The intention is to determine which of the previously retrieved axioms will be triggered, i.e., which effect fluents will change their truth value. The problem of query answering on (untyped) ground facts (without rules) reduces to the problem of unifying the query with the facts, which is $O(n)$, where n is the size of the KB.
3. *Determine which fluents are inertial: $\neg Released(f, t)$?*
   Inertial fluents that have not been affected by e_i in step 2, i.e., are neither released nor the event releases them, need to maintain their truth value in the successor timepoint. As before, the cost of the query is O(n).
4. *Assert in the KB the new truth values of fluents.*
   As the new truth values refer to the successor timepoint, this step does not involve any update of existing facts,

[3] The full proof is available at: *http://www.csd.uoc.gr/∼patkos/Proof.pdf*

rather an assertion of facts to an empty KB. We assume constant time, regardless of the number of assertions.

5. *Use state constraints to infer all indirect effects.* The truth value of those fluents that are released from inertia, yet ruled by state constraints, is determined. In order to perform all derivations from the set of rules, one may apply standard techniques from classical logic inference, such as resolution. To preserve generality of results, by a minor abuse of notation we denote this complexity as $O(INF^{SC})$ or $O(INF^{HCD})$ in the sequel, based on whether the rules involve only the state constraints or both state constraints and HCDs. We revert to this complexity at the end of this section. Also, in this step multiple models may be produced and added to the set of KBs, owed to the unconstrained released fluents, i.e., non-inertial fluents subject to no state constraint.

Summarizing, the complexity of reasoning with the Event Calculus given a sequence of e actions is characterized by $O(e*2^n*(2*n+INF^{SC}))$[4]. The steps of the algorithm follow on from previous complexity analysis of simpler formulations of the classic Event Calculus, as in [Paschke, 2006].

5.2 Possible Worlds Approach

The number of possible worlds depends on the number of unknown fluents, i.e., in a domain of n fluents, u of which being unknown, we need to store at most 2^u possible worlds, where ($u \leq n$). One reasoning task needs to be performed for each of these worlds, since the same effect axioms of a given domain theory may give rise to diverse conclusions in each world. As such, the size of the KB of fluents that is maintained at each timepoint is $O(2^{u-1})$ (a fluent may hold only in half of the total 2^u worlds). Moreover, it is easy to verify that, according to the definition of knowledge, answering if a conjunctive (resp. disjunctive) query of m fluents is known requires at most 2^{u-m} (resp. $2^u - 2^{u-m}$) worlds to check truth in (plus one, if the formula turns out to be unknown).

The algorithm and its logical inferences need to be performed for each possible world, given as input the domain fluents and the fixed set of effect axioms and state constraints. Given a sequence of e actions, the complexity for conjunctive queries is $O(e * 2^u * (2 * n + INF^{SC}) + 2^{u-m} * n)$ (resp. $O(e*2^u*(2*n+INF^{SC})+(2^u-2^{u-m})*n)$ for disjunctive queries), as we first need to progress all possible worlds and then issue a query concerning the desirable fluent formula to a subset of them, with cost $O(n)$.

It should be noted that each fluent that becomes released from the law of inertia causes the number of possible worlds to double, i.e., u increases by one. As a result, both the size of the KB and the reasoning effort increase significantly. Unsettlingly, one should also expect that $u \simeq n$ even for the real-world case, as we argue below.

5.3 DECKT Approach

DECKT performs a single reasoning task with each action using the new axiomatization's meta axioms and substitutes each atomic domain fluent with the corresponding KP and $Knows$ epistemic fluents. KP is always subject to inertia, whereas $Knows$ is always released, therefore step 3 can be disregarded altogether, along with the need to preserve multiple versions of KBs for unconstrained fluents (the $Knows$ fluents never fluctuate). Furthermore, disjunctive epistemic expressions are preserved in this single KB without the need to be broken apart, since all appropriate derivations are reached by means of HCDs. The size of the input for step 2 is equal to that of reasoning without knowledge, as we only search through those $Knows$ fluents that refer to atomic domain ones. The difference now is that each domain effect axiom is replaced by 5 new: 2 due to (KT3), 1 due to (KT5) and 2 due to (KT6.1). Nevertheless, as with the non-epistemic theory, all we need to query in order to progress the KB after any action are the precondition fluents (plus the effect fluent for some of the axioms). Therefore, as before, the complexity of this step is $O(n)$, since one predefined query to a domain of n fluents suffices to provide the necessary knowledge about all DECKT effect axioms mentioned above.

Apart from the epistemic effect axioms, we also introduced axioms for handling HCDs (KT6.2-4). Since HCDs are treated as ordinary inertial fluents (they are modeled in terms of the KP fluent), they fall under the influence of traditional Event Calculus inference (steps 1,2). For these axioms the necessary knowledge that needs to be obtained is whether some HCD that incorporates the effect fluent is among the HCDs stored in the KB. Their number increases as the agent performs actions with unknown preconditions. Let d denote the number of KP fluents that represent HCDs, then the complexity of querying the KB is $O(d)$, where $d \leq 2^n$.

Following the algorithm, we can see that the complexity of reasoning with DECKT is $O(e * (n + d + INF^{HCD}) + n)$, where $O(INF^{HCD})$ is the complexity of logical inference with state constraints and HCDs. The input is the atomic inertial fluents, as usual, reified in the $Knows$ fluent. The last addition refers to querying n atomic epistemic fluents, i.e., the query formula, after the narrative of actions.

In fact, even when the underlying domain axiomatization is non-deterministic, its epistemic meta-program introduces no additional complexity: the KP fluent is always subject to inertia and whenever a domain fluent is released due to uncertainty, its corresponding KP fluents become false according to (KT5). As such, reasoning with DECKT requires only a reduced version of the Event Calculus, where the $Releases$ predicate is removed from the foundational axioms.

5.4 Discussion on Complexity Results

We see that the dominant factor in the complexity of reasoning with possible worlds is the number u of unknown world aspects. In the worst case, $u = n$ resulting in exponential complexity to the size of the domain; yet, even in real-world problems $u \simeq n$, as we expect that in large-scale dynamic domains many more world aspects would be unknown to the agent than known ones at any given time instant. Furthermore, since in practice the query formula that needs to be evaluated is often orders of magnitude smaller in size than the domain itself, i.e., ($n \gg m$), query answering of either conjunctive or disjunctive formulae spirals quickly out of control.

[4] Although constants could be eliminated, we include them for emphasis, so that the reader can follow each step of the algorithm.

With DECKT, on the other hand, it is the number of extra fluents capturing HCDs that predominates the complexity. In fact, although in the worst case it can be that $d = 2^n$ this is a less likely contingency to meet in practice: it would mean that the agent has interacted with all world aspects having no knowledge about any precondition or that state constraints that capture interrelated fluents embody the entire domain (so called *dominos domains* which lead to chaotic environments are not commonly met in commonsense reasoning). Moreover, HCDs fall under the agent's control; even for long-lived agents that execute hundreds of actions, HCDs provide a guide as to which aspects to sense in order to obtain knowledge about the largest set of interrelated fluents, thus enabling the agent to manage their number according to resources.

Apparently, the number and length of HCDs also affect the inference task. Still, the transition from $O(INF^{SC})$ to $O(INF^{HCD})$ has polynomial cost; the complexity of most inference procedures, such as resolution, is linearly affected when increasing the number of implication rules, given that the size of the domain is constant. Finally, one should notice that even in the worst case one reasoning task needs to be performed for each action. Specifically, the factor d does not influence the entire process, as is the case of 2^u for possible worlds, significantly reducing the overall complexity.

6 Implementation of HCD-enabled Reasoner

The formal treatment of knowledge and change we develop aims at programming rational agents for practical implementations. Faced with the task of implementing an agent's mental state, two features are most desirable by a reasoner in order to exploit DECKT's full potential:

- It should enable reasoning to progress incrementally to allow for run-time execution of knowledge-based programs, where an agent can benefit from planning with the knowledge at hand (online reasoning). Each time a program interpreter adds a new action to its agenda, the reasoner should update its current KB appropriately.
- It should permit reification of the epistemic fluents in Event Calculus predicates, to allow for instance the epistemic proposition $Knows(Open(S1))$ to be handled as a term of a first-order logic rather than an atom. Based on this syntactical treatment proposition $HoldsAt(Knows(Open(S1)), 0)$ can be regarded as a well-formed formula.

Most existing Event Calculus reasoners do not satisfy the latter requirement, while only recently an online reasoner was released based on the Cached Event Calculus [Chesani *et al.*, 2009]. Consequently, in order to implement and evaluate different use cases we have constructed an Event Calculus reasoner on top of Jess[5], a rule-based engine that deploys the efficient Rete algorithm for rule matching. Predicates are asserted as facts in the reasoner's agenda, specified by the following template definition:

```
(deftemplate EC (slot predicate)
                (slot event (default nil))
                (slot epistemic (default no))
                (multislot posLtrs )
                (multislot negLtrs )
                (slot time (default 0)))
```

Multislots create lists denoting fluent disjunctions (conjunctions are decomposable into their components according to the definition for knowledge). For instance, knowledge about formula $(f_1 \lor f_2 \lor \neg f_3)$ at time 1 is captured by the fact:

```
(EC (predicate HoldsAt)
    (epistemic Knows)
    (posLtrs f_1 f_2)
    (negLtrs f_3)
    (time 1))
```

The exploitation of lists for maintaining positive and negative literals of formulae enables the representation of HCDs in a syntax-independent manner, so that all meta-axioms of DECKT be translated into appropriately defined rules. This way, the reasoning process can be fully automated, despite the fact that the (KT6) set is time-dependent: the meta-axioms adapt to the facts that exist in the reasoner's agenda at each timepoint. Among the basic features of the new reasoner[6] are:

- given a domain axiomatization, the user can select between the execution of classical Event Calculus reasoning or epistemic reasoning using DECKT.
- the domain axiomatization is written based on a simple, intuitive Event Calculus-like syntax, which is then parsed into appropriate Jess rules (Figure 2). The user may modify the Jess program as well, thus augmenting the axiomatization with advanced and more expressive components, such as rules and constraints not yet supported by the Event Calculus parser.
- new events and observations can be asserted on-the-fly, based on information acquired at execution time, e.g., from the user or the agent's sensors.
- reasoning can progress incrementally, while the user can decide the time span of the execution step.
- a GUI is provided for modifying and storing Event Calculus or Jess programs, for visualizing the output and for providing input to the reasoner at execution time.

We should note, though, that the implementation of DECKT described here is general enough to be written in any prolog-like syntax and is not restricted to the Jess tool.

7 Conclusions

The DECKT framework has been used to extended benchmark commonsense problems with incomplete knowledge, e.g., those included in [Mueller, 2006]. It is also integrated in an Ambient Intelligence project that is currently in progress in our institute, which introduces highly demanding challenges within dynamic environments. The benefits of HCDs are investigated in a number of other interesting aspects in cognitive robotics as well, such as for representing the potential effects of physical actions in unknown worlds, on whose occurrences the agent can only speculate, as well as for temporal indeterminacy of events. Among our future goals is also to extend the applicability of the new reasoner, constituting it a usable educational tool for epistemic action theories.

[5] Jess, *http://www.jessrules.com/* (last accessed: May 2011)

[6] Jess-EC Reasoner: *http://www.csd.uoc.gr/∼patkos/deckt.htm*

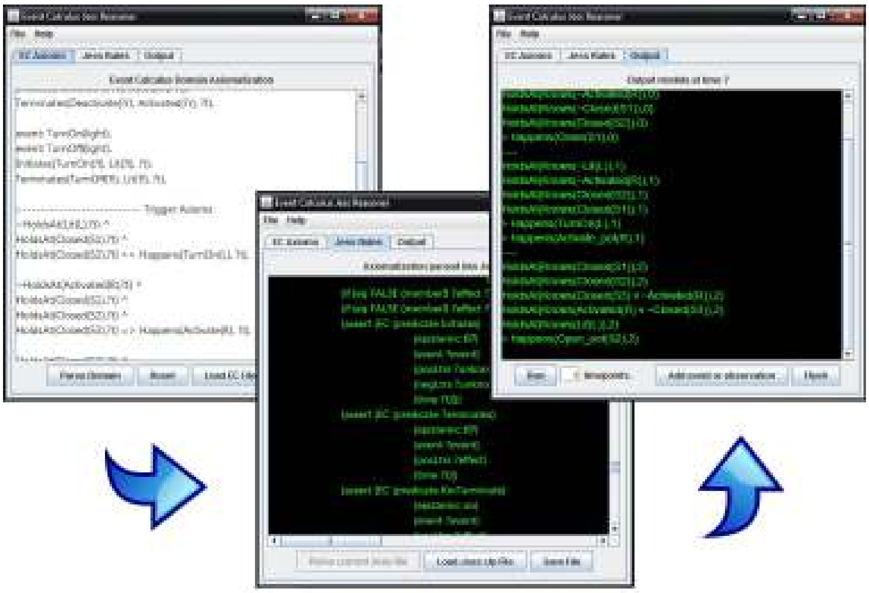

Figure 2: The Jess-EC Reasoner with epistemic capabilities: the Event Calculus domain is translated into Jess rules, whose input and execution the user can modify at execution time.

References

[Chesani *et al.*, 2009] Federico Chesani, Paola Mello, Marco Montali, and Paolo Torroni. Commitment tracking via the reactive event calculus. In *Proceedings of the 21st international jont conference on Artifical intelligence*, pages 91–96, San Francisco, CA, USA, 2009. Morgan Kaufmann Publishers Inc.

[Demolombe and Parra, 2000] Robert Demolombe and Maria del Pilar Pozos Parra. A simple and tractable extension of situation calculus to epistemic logic. pages 515–524, 2000.

[Forth and Shanahan, 2004] Jeremy Forth and Murray Shanahan. Indirect and Conditional Sensing in the Event Calculus. In *ECAI*, pages 900–904, 2004.

[Kowalski and Sergot, 1986] R Kowalski and M Sergot. A logic-based calculus of events. *New Generation Computing*, 4:67–95, January 1986.

[Liu and Levesque, 2005] Yongmei Liu and Hector J. Levesque. Tractable reasoning with incomplete first-order knowledge in dynamic systems with context-dependent actions. In *Proceedings of the 19th international joint conference on Artificial intelligence*, pages 522–527, San Francisco, CA, USA, 2005.

[Lob, 2001] Knowledge and the Action Description Language A. *Theory and Practice of Logic Programming*, 1:129–184, 2001.

[Moore, 1985] R. C. Moore. A Formal Theory of Knowledge and Action. In *Formal Theories of the Commonsense World*, pages 319–358. J. Hobbs, R. Moore (Eds.), 1985.

[Mueller, 2006] Erik Mueller. *Commonsense Reasoning*. Morgan Kaufmann, 1st edition, 2006.

[Mueller, 2007] Erik Mueller. Discrete Event Calculus with Branching Time. In *Eigth International Symposium on Logical Formalizations of Commonsense Reasoning (Commonsense'07)*, pages 126–131, 2007.

[Paschke, 2006] Adrian Paschke. ECA-RuleML: An Approach Combining ECA Rules with Temporal Interval-based KR Event/Action Logics and Transactional Update Logics. *Computer Research Repository*, abs/cs/0610167, 2006.

[Patkos and Plexousakis, 2009] Theodore Patkos and Dimitris Plexousakis. Reasoning with knowledge, action and time in dynamic and uncertain domains. In *Proceedings of the 21st international jont conference on Artifical intelligence*, pages 885–890, USA, 2009. Morgan Kaufmann Publishers Inc.

[Petrick and Levesque, 2002] R. Petrick and H. Levesque. Knowledge Equivalence in Combined Action Theories. In *Proceedings of the 8th International Conference on Principles of Knowledge Representation and Reasoning (KR-02)*, pages 303–314, 2002.

[Scherl and Levesque, 2003] Richard B. Scherl and Hector J. Levesque. Knowledge, Action, and the Frame Problem. *Artificial Intelligence*, 144(1-2):1–39, 2003.

[Thielscher, 2000] Michael Thielscher. Representing the knowledge of a robot. In A. Cohn, F. Giunchiglia, and B. Selman, editors, *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning (KR)*, pages 109–120. Morgan Kaufmann, 2000.

[Thielscher, 2005a] M. Thielscher. Handling Implication and Universal Quantification Constraints in FLUX. In *Proceedings of the 11th International Conference on Principles and Practice of Constraint Programming (CP11)*, pages 667–681, 2005.

[Thielscher, 2005b] Michael Thielscher. FLUX: A Logic Programming Method for Reasoning Agents. *Theory and Practice of Logic Programming*, 5(4–5):533–565, 2005.

[Vassos *et al.*, 2009] Stavros Vassos, Stavros Sardina, and Hector Levesque. Progressing Basic Action Theories with non-Local Effect Actions. In *Proceedings of the Ninth International Symposium on Logical Formalizations of Commonsense Reasoning (CS'09)*, pages 135–140, 2009.

Preferred Explanations: Theory and Generation via Planning*

Shirin Sohrabi
Department of Computer Science
University of Toronto
Toronto, Canada
shirin@cs.toronto.edu

Jorge A. Baier
Depto. de Ciencia de la Computación
Pontificia Universidad Católica de Chile
Santiago, Chile
jabaier@ing.puc.cl

Sheila A. McIlraith
Department of Computer Science
University of Toronto
Toronto, Canada
sheila@cs.toronto.edu

Abstract

In this paper we examine the general problem of generating preferred explanations for observed behavior with respect to a model of the behavior of a dynamical system. This problem arises in a diversity of applications including diagnosis of dynamical systems and activity recognition. We provide a logical characterization of the notion of an explanation. To generate explanations we identify and exploit a correspondence between explanation generation and planning. The determination of *good* explanations requires additional domain-specific knowledge which we represent as preferences over explanations. The nature of explanations requires us to formulate preferences in a somewhat retrodictive fashion by utilizing Past Linear Temporal Logic. We propose methods for exploiting these somewhat unique preferences effectively within state-of-the-art planners and illustrate the feasibility of generating (preferred) explanations via planning.

1. Introduction

In recent years, planning technology has been explored as a computational framework for a diversity of applications. One such class of applications is the class that corresponds to *explanation generation* tasks. These include narrative understanding, plan recognition (Ramírez and Geffner 2009), finding excuses (Göbelbecker et al. 2010), and diagnosis (e.g., Sohrabi, Baier, and McIlraith 2010; Grastien et al. 2007).[1] While these tasks differ, they share a common computational core, calling upon a dynamical system model to account for system behavior, observed over a period of time. The observations may be over aspects of the state of the world, or over the occurrence of events; the account typically takes the form of a set or sequence of actions and/or state that is extracted from the construction of a plan that embodies the observations. For example, in the case of diagnosis, the observations might be of the, possibly aberrant, behavior of an electromechanical device over a period of time, and the explanation a sequence of actions that conjecture faulty events. In the case of plan recognition, the observations might be of the actions of an agent, and the explanation a plan that captures what the agent is doing and/or the final goal of that plan.

Here we conceive the computational core underlying explanation generation of dynamical systems as a nonclassical planning task. Our focus in this paper is with the generation of *preferred* explanations – how to specify preference criteria, and how to compute preferred explanations using planning technology. Most explanation generation tasks that distinguish a subset of preferred explanations appeal to some form of domain-independent criteria such as minimality or simplicity. Domain-specific knowledge has been extensively studied within the static-system explanation and abduction literature as well as in the literature on specific applications such as diagnosis. Such domain-specific criteria often employ probabilistic information, or in its absence default logic of some notion of specificity (e.g., Brewka 1994).

In 2010, we examined the problem of diagnosis of discrete dynamical systems (a task within the family of explanation generation tasks), exploiting planning technology to compute diagnoses and suggesting the potential of planning preference languages as a means of specifying preferred diagnoses (Sohrabi, Baier, and McIlraith 2010). Building on our previous work, in this paper we explicitly examine the use of preference languages for the broader task of explanation generation. In doing so, we identify a number of somewhat unique representational needs. Key among these is the need to talk about the *past* (e.g., "*If I observe that my car has a flat tire then I prefer explanations where my tire was previously punctured.*") and the need to encode complex observation patterns (e.g., "*My brakes have been failing intermittently.*") and how these patterns relate to possible explanations. To address these requirements we specify preferences in Past Linear Temporal Logic (PLTL), a superset of Linear Temporal Logic (LTL) that is augmented with modalities that reference the past. We define a finite variant of PLTL, f-PLTL, that is augmented to include action occurrences.

Motivated by a desire to generate explanations using state-of-the-art planning technology, we propose a means of compiling our f-PLTL preferences into the syntax of PDDL3, the Planning Domain Description Language 3 that supports the representation of temporally extended preferences (Gerevini et al. 2009). Although, f-PLTL is more

*A version of this paper appears in the Proceedings of the Twenty-Fifth Conference on Artificial Intelligence (AAAI-11)

[1] (Grastien et al. 2007) characterized diagnosis in terms of SAT but employed a planning-inspired encoding.

expressive than the preference subset of PDDL3 (e.g., f-PLTL has action occurrences and arbitrary nesting of temporal modalities), our compilation preserves the f-PLTL semantics while conforming to PDDL3 syntax. This enables us to exploit PDDL3-compliant preference-based planners for the purposes of generating preferred explanations. We also propose a further compilation to remove all temporal modalities from the syntax of our preferences (while preserving their semantics) enabling the exploitation of cost-based planners for computing preferred explanations. Additionally, we exploit the fact that observations are *known* a priori to pre-process our suite of explanation preferences prior to explanation generation in a way that further simplifies the preferences and their exploitation. We show that this compilation significantly improves the time required to find preferred explanations, sometimes by orders of magnitude. Experiments illustrate the feasibility of generating (preferred) explanations via planning.

2. Explaining Observed Behavior

In this section we provide a logical characterization of a preferred explanation for observed behavior with respect to a model of a dynamical system. In what follows we define each of the components of this characterization, culminating in our characterization of a preferred explanation.

2.1 Dynamical Systems

Dynamical systems can be formally described in many ways. In this paper we assume a finite domain and model dynamical systems as transition systems. For convenience, we define transitions systems using a planning language. As such transitions occur as the result of actions described in terms of preconditions and effects. Formally, a dynamical system is a tuple $\Sigma = (F, A, I)$, where F is a finite set of fluent symbols, A is a set of actions, and I is a set of clauses over F that defines a set of possible initial states. Every action $a \in A$ is defined by a precondition $prec(a)$, which is a conjunction of fluent literals, and a set of conditional effects of the form $C \to L$, where C is a conjunction of fluent literals and L is a fluent literal.

A *system state* s is a set of fluent symbols, which intuitively defines all that is true in a particular state of the dynamical system. For a system state s, we define $M_s : F \to \{true, false\}$ as the truth assignment that assigns the truth value $true$ to f if $f \in s$, and assigns $false$ to f otherwise. We say a state s is *consistent* with a set of clauses $\mathcal{C}$, if $M_s \models c$, for every $c \in \mathcal{C}$. Given a state s consistent with I, we denote Σ/s as the dynamical system $(F, A, I/s)$, where I/s stands for the set of unit clauses whose only model is M_s. We say a dynamical system $\Sigma = (F, A, I)$ has a *complete initial state* iff there is a unique truth assignment $M : F \to \{true, false\}$ such that $M \models I$.

We assume that an action a is executable in a state s if $M_s \models prec(a)$. If a is executable in a state s, we define its successor state as $\delta(a, s) = (s \setminus Del) \cup Add$, where Add contains a fluent f iff $C \to f$ is an effect of a and $M_s \models C$. On the other hand Del contains a fluent f iff $C \to \neg f$ is an effect of a, and $M_s \models C$. We define $\delta(a_0 a_1 \dots a_n, s) = \delta(a_1 \dots a_n, \delta(a_0, s))$, and $\delta(\epsilon, s) = s$. A sequence of actions α is *executable* in s if $\delta(\alpha, s)$ is defined. Furthermore α is executable in Σ iff it is executable in s, for any s consistent with I.

2.2 Past LTL with Action Occurrences

Past modalities have been exploited for a variety of specialized verification tasks and it is well established that LTL augmented with such modalities has the same expressive power as LTL restricted to future modalities (Gabbay 1987). Nevertheless, certain properties (including fault models and explanation models) are more naturally specified and read in this augmentation of LTL. For example specifying that every alarm is due to a fault can easily be expressed by $\square(alarm \to \blacklozenge fault)$, where $\square$ means always and $\blacklozenge$ means once in the past. Note that $\neg \mathsf{U}(\neg fault, (alarm \wedge \neg fault))$ is an equivalent formulation that uses only future modalities but is much less intuitive. In what follows we define the syntax and semantics of, f-PLTL, a variant of LTL that is augmented with past modalities and action occurrences.

Syntax Given a set of fluent symbols F and a set of action symbols A, the atomic formulae of the language are: either a fluent symbol, or $occ(a)$, for any $a \in A$. Non-atomic formulae are constructed by applying negation, by applying a standard boolean connective to two formulae, or by including the future temporal modalities "until" (U), "next" ($\bigcirc$), "always"($\square$), and "eventually"($\lozenge$), or the past temporal modalities "since" (S), "yesterday" ($\bullet$), "always in the past"($\blacksquare$), and "eventually in the past"($\blacklozenge$). We say ϕ is expressed in future f-PLTL if it does not contain any past temporal modalities. Similarly, ϕ is expressed in *past f-*PLTL if it does not contain any future temporal modalities. A *non-temporal* formula does not contain any temporal modalities.

Semantics Given a system Σ, a sequence of actions α, and an f-PLTL formula φ, the semantics defines when α satisfies φ in Σ. An f-PLTL formula is interpreted over *finite* rather than infinite sequences of states. Its semantics resembles that of LTL on so-called truncated paths (Eisner et al. 2003). Before we define the semantics formally, we give two definitions. Let s be a state and $\alpha = a_0 a_1 \dots a_n$ be a (finite) sequence of actions. We say that σ is an *execution trace* of α in s iff $\sigma = s_0 s_1 s_2 \dots s_{n+1}$ and $\delta(a_i, s_i) = s_{i+1}$, for any $i \in [0, n]$. Furthermore, if l is the sequence $\ell_0 \ell_1 \dots \ell_n$, we abbreviate its suffix $\ell_i \ell_{i+1} \dots \ell_n$ by l_i.

Definition 1 (Truth of an f-PLTL Formula) *An f-*PLTL *formula φ is satisfied by α in a dynamical system $\Sigma = (F, A, I)$ iff for any state s consistent with I, the execution trace σ of α in s is such that $\langle \sigma, \alpha \rangle \models \varphi$, where* [2]

- $\langle \sigma_i, \alpha_i \rangle \models \varphi$, *where $\varphi \in F$ iff φ is an element of the first state of σ_i.*
- $\langle \sigma_i, \alpha_i \rangle \models occ(a)$ *iff $i < |\alpha|$ and a_i is the first action of α_i.*
- $\langle \sigma_i, \alpha_i \rangle \models \bigcirc\varphi$ *iff $i < |\sigma| - 1$ and $\langle \sigma_{i+1}, \alpha_{i+1} \rangle \models \varphi$*
- $\langle \sigma_i, \alpha_i \rangle \models \mathsf{U}(\varphi, \psi)$ *iff there exists a $j \in \{i, ..., |\sigma| - 1\}$ such that $\langle \sigma_j, \alpha_j \rangle \models \psi$ and for every $k \in \{i, ..., j-1\}$, $\langle \sigma_k, \alpha_k \rangle \models \varphi$*

[2] We omit standard definitions for $\neg$, $\vee$.

- $\langle \sigma_i, \alpha_i \rangle \models \bullet\varphi$ *iff* $i > 0$ *and* $\langle \sigma_{i-1}, \alpha_{i-1} \rangle \models \varphi$
- $\langle \sigma_i, \alpha_i \rangle \models \mathsf{S}(\varphi, \psi)$ *iff there exists a* $j \in \{0, ..., i\}$ *such that* $\langle \sigma_j, \alpha_j \rangle \models \psi$ *and for every* $k \in \{j+1, ..., i\}$, $\langle \sigma_k, \alpha_k \rangle \models \varphi$

The semantics of other temporal modalities are defined in terms of these basic elements, e.g., $\blacksquare\varphi \overset{\text{def}}{=} \neg\blacklozenge\neg\varphi$, $\blacklozenge\varphi \overset{\text{def}}{=} \mathsf{S}(\mathsf{true}, \varphi)$, and $\Diamond\varphi \overset{\text{def}}{=} \mathsf{U}(\mathsf{true}, \varphi)$. Observe that our semantics adopts a strong next operator; i.e., $\bigcirc\phi$ will not be satisfied if evaluated in the final state of a finite sequence.

It is well recognized that some properties are more naturally expressed using past modalities. An additional property of such modalities is that they can construct formulae that are exponentially more succinct than their future modality counterparts. Indeed let Σ_n be a system with $F_n = \{p_0, \ldots, p_n\}$, let $\psi_i = p_i \leftrightarrow \blacklozenge(\neg\bullet\mathsf{true} \wedge p_i)$, and let $\Psi = \Box\big(\bigwedge_{i=1}^{n} \psi_i \rightarrow \psi_0\big)$. Intuitively, ψ_i expresses that "p_i has the same truth value now as it did in the initial state".

Theorem 1 (Following Markey 2003) *Any formula ψ, expressed in future f-*PLTL*, equivalent to Ψ (defined as above) has size $\Omega(2^{|\Psi|})$.*

Note that although Markey's theorem is related to temporal logic evaluated on infinite paths, the property also holds when it is interpreted on truncated paths.

In the following sections we provide a translation of formulae with past modalities into future-only formulae, in order to use existing planning technology. Despite Markey's theorem, it is possible to show that the blowup for Ψ can be avoided if one modifies the transition system to include additional predicates that keep track of the initial truth value of each of $p_0, \ldots, p_n$. Such a modification can be done in linear time.

2.3 Characterizing Explanations

Given a description of the behavior of a dynamical system and a set of observations about the state of the system and/or action occurrences, we define an explanation to be a pairing of actions, orderings, and possibly state that account for the observations in the context of the system dynamics. The definitions in this section follow (but differ slightly from) the definitions of dynamical diagnosis we proposed in (Sohrabi, Baier, and McIlraith 2010), which in turn elaborate and extend previous work (e.g., McIlraith 1998; Iwan 2001).

Assuming our system behavior is defined as a dynamical system and that the observations are expressed in future f-PLTL, we define an explanation as a tuple (H, α) where H is a set of clauses representing an assumption about the initial state and α is an executable sequence of actions that makes the observations satisfiable. If the initial state is complete, then H is empty, by definition. In cases where we have incomplete information about the initial state, H denotes assumptions that we make, either because we need to establish the preconditions of actions we want to conjecture in our explanation or because we want to avoid conjecturing further actions to establish necessary conditions. Whether it is better to conjecture more actions or to make an assumption is dictated by domain-specific knowledge, which we will encode in preferences.

Definition 2 (Explanation) *Given a dynamical system $\Sigma = (F, A, I)$, and an observation formula φ, expressed in future f-*PLTL*, an explanation is a tuple (H, α), where H is a set of clauses over F such that $I \cup H$ is satisfiable, $I \not\models H$, and α is a sequence of actions in A such that α satisfies φ in the system $\Sigma_A = (F, A, I \cup H)$.*

Example Assume a standard logistics domain with one truck, one package, and in which all that is known initially is that the truck is at loc_1. We observe pkg is unloaded from $truck_1$ in loc_1, and later it is observed that pkg is in loc_2. One can express the observation as

$$\Diamond[occ(unload(pkg, loc_1)) \wedge \bigcirc\Diamond at(pkg, loc_2)]$$

A possible explanation (H, α), is such that $H = \{in(pkg, truck_1)\}$, and α is $unload(pkg, loc_1)$, $load(pkg, loc_1)$, $drive(loc_1, loc_2)$, $unload(pkg, loc_2)$.

Note that aspects of H and α can be further filtered to identify elements of interest to a particular user following techniques such as those in (McGuinness et al. 2007).

Given a system and an observation, there are many possible explanations, not all of high quality. At a theoretical level, one can assume a reflexive and transitive preference relation $\preceq$ between explanations. If E_1 and E_2 are explanations and $E_1 \preceq E_2$ we say that E_1 is *at least as preferred as* E_2. $E_1 \prec E_2$ is an abbreviation for $E_1 \preceq E_2$ and $E_2 \not\preceq E_1$.

Definition 3 (Optimal Explanation) *Given a system Σ, E is an optimal explanation for observation φ iff E is an explanation for φ and there does not exist another explanation E' for φ such that $E' \prec E$.*

3. Complexity and Relationship to Planning

It is possible to establish a relationship between explanation generation and planning. Before doing so, we give a formal definition of planning.

A *planning problem with temporally extended goals* is a tuple $P = (\Sigma, G)$, where Σ is a dynamical system, and G is a goal formula expressed in future f-PLTL. The sequence of actions α is a *plan* for P if α is executable in Σ and α satisfies G in Σ. A planning problem (Σ, G) is *classical* if Σ has a complete initial state, and *conformant* otherwise.

The following is straightforward from the definition.

Proposition 1 *Given a dynamical system $\Sigma = (F, A, I)$ and an observation formula φ, expressed in future f-*PLTL*, then (H, α) is an explanation iff α is a plan for conformant planning problem $P = ((F, A, I \cup H), \varphi)$ where $I \cup H$ is satisfiable and where φ is a temporally extended goal.*

In systems with complete initial states, the generation of a single explanation corresponds to classical planning with temporally extended goals.

Proposition 2 *Given a dynamical system Σ such that Σ has complete initial state, and an observation formula φ, expressed in future f-*PLTL*, then $(\emptyset, \alpha)$ is an explanation iff α is a plan for classical planning problem $P = (\Sigma, \varphi)$ with temporally extended goal φ.*

Indeed, the complexity of explanation existence is the same as that of classical planning.

Theorem 2 *Given a system Σ and a temporally extended formula φ, expressed in future f-*PLTL*, explanation existence is PSPACE-complete.*

Proof sketch. For membership, we propose the following NPSPACE algorithm: guess an explanation H such that $I \cup H$ has a unique model, then call a PSPACE algorithm (like the one suggested by de Giacomo and Vardi (1999)) to decide (classical) plan existence. Then we use the fact that NPSPACE=PSPACE. Hardness is given by Proposition 2 and the fact that classical planning is PSPACE-hard (Bylander 1994). §

The proof of Theorem 2 appeals to a non-deterministic algorithm that provides no practical insight into how to translate plan generation into explanation generation. At a more practical level, there exists a deterministic algorithm that maps explanation generation to classical plan generation.

Theorem 3 *Given an observation formula φ, expressed in future f-*PLTL*, and a system Σ, there is an exponential-time procedure to construct a classical planning problem $P = (\Sigma', \varphi)$ with temporally extended goal φ, such that if α is a plan for P, then an explanation (H, α') can be generated in linear time from α.*

Proof sketch. Σ', the dynamical system that describes P is the same as $\Sigma = (F, A, I)$, augmented with additional actions that "complete" the initial state. Essentially, each such action generates a successor state s that is consistent with I. There is an exponential number of them. If $a_0 a_1 \ldots a_n$ is a plan for P, we construct the explanation (H, α') as follows. H is constructed with the facts true in the state s that a_0 generates. α' is set to $a_1 \ldots a_n$. §

All the previous results can be re-stated in a rather straightforward way if the desired problem is to find an optimal explanation. In that case the reductions are made to preference-based planning (Baier and McIlraith 2008).

The proofs of the theorems above unfortunately do not provide a *practical* solution to the problem of (high-quality) explanation generation. In particular, we have assumed that planning problems contain temporally extended goals expresed in future f-PLTL. No state-of-the-art planner that we are aware of supports these goals directly. We have not provided a compact and useful way to represent the $\preceq$ relation.

4. Specifying Preferred Explanations

The specification of preferred explanations in dynamical settings presents a number of unique representational requirements. One such requirement is that preferences over explanations be **contextualized with respect to observations**, and these observations themselves are not necessarily single fluents, but rich temporally extended properties – sometimes with characteristic forms and patterns. Another unique representational requirement is that the generation of explanations (and preferred explanations) necessitates **reflecting on the past**. Given some observations over a period of time, we wish to conjecture what preceded these observations in order to account for their occurrence. Such explanations may include certain system state that explains the observations, or it may include action occurrences. Explanations may also include reasonable facts that we wish to posit about the initial state (e.g., that it's below freezing outside – a common precursor to a car battery being dead).

In response to the somewhat unique representational requirements, we express preferences in f-PLTL. In order to generate explanations using state-of-the-art planners, an objective of our work was to make the preference input language PDDL3 compatible. However, f-PLTL is more expressive than the subset of LTL employed in PDDL3, and we did not wish to lose this expressive power. In the next section we show how to compile away some or all temporal modalities by exploiting the correspondence between past and future modalities and by exploiting the correspondence between LTL and Büchi automata. In so doing we preserve the expressiveness of f-PLTL within the syntax of PDDL3.

4.1 Preferred Explanations

A high quality explanation is determined by the optimization of an objective function. The PDDL3 metric function we employ for this purpose is a weighted linear sum of formulae to be minimized. I.e., (minimize (+ ($*$ w_1 ϕ_1) $\ldots$($*$ w_k ϕ_k))) where each ϕ_i is a formula that evaluates to 0 or 1 depending on whether an associated preference formula, a property of the explanation trajectory, is satisfied or violated; w_i is a weight characterizing the importance of that property (Gerevini et al. 2009). The key role of our preferences is to convey domain-specific knowledge regarding the most preferred explanations for particular observations. Such preferences take on the following canonical form.

Definition 4 (Explanation Preferences) $\Box(\phi_{obs} \rightarrow \phi_{expl})$ *is an explanation preference formula where ϕ_{obs}, the observation formula, is any formula expressed in future f-*PLTL*, and ϕ_{expl}, the explanation formula, is any formula expressed in past f-*PLTL*. Non-temporal expressions may appear in either formula.*

An observation formula, ϕ_{obs}, can be as simple as the observation of a single fluent or action occurrence (e.g., *my car won't start.*), but it can also be a complex formula. In many explanation scenarios, observations describe a telltale ordering of system properties or events that suggest a unique explanation such as a car that won't start every time it rains. To simplify the description of observation formulae, we employ precedes as a syntactic constructor of observation patterns. φ_1 precedes φ_2 indicates that φ_1 is observed before φ_2. More generally, one can express ordering among observations by using formula of the form (φ_1 precedes φ_2 ... precedes φ_n) with the following interpretation:

$$\varphi_1 \wedge \bigcirc\Diamond(\varphi_2 \wedge \bigcirc\Diamond(\varphi_3 ...(\varphi_{n-1} \wedge \bigcirc\Diamond\varphi_n)...)) \qquad (1)$$

Equations (2) and (3) illustrate the use of precedes to encode a total (respectively, partial) ordering among observations. These are two common forms of observation formulae.

$$(\mathsf{obs}_1 \text{ precedes } \mathsf{obs}_2 \text{ precedes } \mathsf{obs}_3 \text{ precedes } \mathsf{obs}_4) \qquad (2)$$

$$(\mathsf{obs}_3 \text{ precedes } \mathsf{obs}_4) \wedge (\mathsf{obs}_1 \text{ precedes } \mathsf{obs}_2) \qquad (3)$$

Further characteristic observation patterns can also be easily described using precedes. The following is an example of an intermittent fault.

(alarm precedes no_alarm precedes alarm precedes no_alarm)

Similarly, explanation formulae, ϕ_{exp}, can be complex temporally extended formulae over action occurrences and fluents. However, in practice these explanations may be reasonably simple assertions of properties or events that held (resp. occurred) in the past. The following are *some* canonical forms of explanation formulae: $(\blacklozenge e_1 \wedge ... \wedge \blacklozenge e_n)$, and $(\blacklozenge e_1 \otimes ... \otimes \blacklozenge e_n)$, where $n \geq 1$, and e_i is either a fluent $\in F$ or $occ(a)$, $a \in A$ and $\otimes$ is exclusive or.

5. Computing Preferred Explanations

In previous sections we addressed issues related to the specification and formal characterization of preferred explanations. In this section we examine how to effectively *generate* explanations using state-of-the-art planning technology.

Propositions 1 and 2 establish that we can generate explanations by treating an observation formula φ as the temporally extended goal of a conformant (resp. classical) planning problem. Preferred explanations can be similarly computed using preference-based planning techniques. To employ state-of-the-art planners, we must represent our observation formulae and the explanation preferences in syntactic forms that are compatible with some version of PDDL. Both types of formulae are expressed in f-PLTL so PDDL3 is a natural choice since it supports preferences and some LTL constructs. However, f-PLTL is more expressive than PDDL3, supporting arbitrarily nested past and future temporal modalities, action occurrences, and most importantly the *next* modality, $\bigcirc$, which is essential to the encoding of an ordered set of properties or action occurrences that occur over time. As a consequence, partial- and total-order observations are *not* expressible in PDDL3's subset of LTL, and so it follows that the precedes constructor commonly used in the ϕ_{obs} component of explanation preferences is not expressible in the PDDL3 LTL subset. There are similarly many typical ϕ_{expl} formulae that cannot be expressed directly in PDDL3 because of the necessity to nest temporal modalities. So to generate explanations using planners, we must devise other ways to encode our observation formulae and our explanation preferences.

5.1 Approach 1: PDDL3 via Compilation

Although it is not possible to express our preferences directly in PDDL3, it is possible to compile unsupported temporal formulae into other formulae that *are* expressible in PDDL3. To translate to PDDL3, we utilize Baier and McIlraith's future LTL compilation approach (2006), which we henceforth refer to as the BM compilation. Given an LTL goal formula ϕ, expressed in future f-PLTL, and a planning problem P, the BM compilation executes the following two steps: (*phase 1*) generates a finite state automaton for ϕ, and (*phase 2*) encodes the automaton in the planning problem by adding new predicates to describe the changing configuration of the automaton as actions are performed. The result is a new planning problem P' that augments P with a newly introduced *accepting predicate* $accept_\phi$ that becomes true after performing a sequence of actions α in the initial state if and only if α satisfies ϕ in P's dynamical system. Predicate $accept_\phi$ is the (classical) goal in problem P'. Below we introduce an extension of the BM compilation that allows compiling away formulae expressed in past f-PLTL.

Our compilation takes dynamical system Σ, an observation φ, a set Γ of formulae corresponding to explanation preferences, and produces a PDDL3 planning problem.

Step 1 Takes Σ and φ and generates a classical planning problem P_1 with temporally extended goal φ using the procedure described in the proof for Theorem 3.

Step 2 Compiles away *occ* in P_1, generating P_2. For each occurrence of $occ(a)$ in Γ or φ, it generates an additional fluent $happened_a$ which is made true by a and is deleted by all other actions. Replace $occ(a)$ by $happened_a$ in Γ and φ.

Step 3 Compiles away all the *past* elements of preferences in Γ. It uses the BM compilation over P_2 to compile away past temporal operators in preference formulae of the form $\Box(\phi_{obs} \rightarrow \phi_{expl})$, generating P_3. For every explanation formula ϕ_{expl}, expressed in past f-PLTL, in Γ we do the following. We compute the reverse of ϕ_{expl}, ϕ^r_{expl}, as a formula just like ϕ_{expl} but with all past temporal operators changed to their future counterparts (i.e., $\bullet$ by $\bigcirc$, $\blacklozenge$ by $\Diamond$, S by U). Note that ϕ_{expl} is satisfied in a trajectory of states σ iff ϕ^r_{expl} is satisfied in the reverse of σ. Then, we use phase 1 of the BM compilation to build a finite state automaton $A_{\phi^r_{expl}}$ for ϕ^r_{expl}. We now compute the reverse of $A_{\phi^r_{expl}}$ by switching accepting and initial states and reversing the direction of all transitions. Then we continue with phase 2 of the BM compilation, generating a new planning problem for the reverse of $A_{\phi^r_{expl}}$. In the resulting problem the new predicate $accept_{\phi_{expl}}$ becomes true as soon as the formula ϕ_{expl}, expressed in past f-PLTL, is made true by the execution of an action sequence. We replace any occurrence of ϕ_{expl} in Γ by $accept_{\phi_{expl}}$. We similarly use the BM compilation to remove future temporal modalities from φ and ϕ_{obs}. This is only necessary if they contain nested modalities or $\bigcirc$, which they often will. The output of this step is PDDL3 compliant. To generate PDDL3 output without *any* temporal operators, we perform the following further step.

Step 4 (optional) Compiles away temporal operators in Γ and φ using the BM compilation, ending with simple preferences that refer only to the final state.

Theorem 4 *Let P_3 be defined as above for a description Σ, an observation φ, and a set of preferences Γ. If α is a plan for P_3 with an associated metric function value M, then we can construct an explanation (H, α) for Σ and φ with associated metric value M in linear time.*

Although Step 4 is not required, it has practical value. Indeed, it enables potential application of other compilation approaches that work directly with PDDL3 without temporal operators. For example, it enables the use of Keyder and Geffner's compilation (2009) to compile preferences into corresponding actions costs so that standard cost-based planners can be used to find explanations. This is of practical importance since cost-based planners are (currently) more mature than PDDL3 preference-based planners.

5.2 Approach 2: Pre-processing (Sometimes)

The compiled planning problem resulting from the application of Approach 1 can be employed with a diversity of planners to generate explanations. Unfortunately, the preferences may not be in a form that can be effectively exploited by delete relaxation based heuristic search. Consider the preference formula $\gamma = \Box(\phi_{obs} \rightarrow \phi_{expl})$. Step 4 culminates in an automaton with accepting predicate $accept_{\gamma}$. Unfortunately, $accept_{\gamma}$ is generally true at the outset of plan construction because ϕ_{obs} is false – the observations have not yet occurred in the plan – making $\phi_{obs} \rightarrow \phi_{expl}$, and thus γ, trivially true. This deactivates the heuristic search to achieve $accept_{\gamma}$ and thus the satisfaction of this preference does not benefit from heuristic guidance. For a restricted but compelling class of preferences, namely those of the form $\Box(\phi_{obs} \rightarrow \bigwedge_i \blacklozenge e_i)$ with e_i a non-temporal formula, we can pre-process our preference formula in advance of applying Approach 1, by exploiting the fact that we *know* a priori what observations have occured. Our pre-processing utilizes the following LTL identity:

$$\Box(\phi_{obs} \rightarrow \bigwedge_i \blacklozenge e_i) \wedge \Diamond\phi_{obs} \equiv \bigwedge_i \neg\phi_{obs} \mathsf{U}(e_i \wedge \Diamond\phi_{obs}).$$

Given a preference in the form $\Box(\phi_{obs} \rightarrow \bigwedge_i \blacklozenge e_i)$ we determine whether ϕ_{obs} is entailed by the observation φ (this can be done efficiently given the form of our observations). If this is the case, we use the identity above to transform our preferences, followed by application of Approach 1. The accepting predicate of the resulting automaton becomes true if ϕ_{expl} is satisfied prior to ϕ_{obs}. In the section to follow, we see that exploiting this pre-processing can improve planner performance significantly.

6. Experimental Evaluation

The objective of our experimental analysis was to gain some insight into the behavior of our proposed preference formalism, specifically, we wanted to: 1) develop a set of somewhat diverse benchmarks and illustrate the use of planners in the generation of explanations; 2) examine how planners perform when the number of preferences is increased; and 3) investigate the computational time gain resulting from Approach 2. We implemented all compilation techniques discussed in Section 5 to produce PDDL3 planning problem with simple preferences that are equivalent to the original explanation generation problems.

We used four domains in our experiments: a computer domain (see Grastien et al. 2007), a car domain (see McIlraith and Scherl 2000), a power domain (see McIlraith 1998), and the trucks domain from IPC 2006. We modified these domains to account for how observations and explanations occur within the domain. In addition, we created two instances of the same problem, one with total-order observations and another with partial-order observations. Since the observations we considered were either total- or partial-order, we were able to compile them away using a technique that essentially makes an observation possible only after all preceding observations have been observed (Haslum and Grastien 2009; 2011). Finally, we increased problem difficulty by increasing the number of observations in each problem.

To further address our first objective, we compared the performance of FF (Hoffmann and Nebel 2001), LAMA (Richter, Helmert, and Westphal 2008), SGPlan$_6$ (Hsu and Wah 2008) and HPLAN-P (Baier, Bacchus, and McIlraith 2009) on our compiled problems but with no preferences. The results show that in the total-order cases, all planners except HPLAN-P solved all problems within seconds, while HPLAN-P took much longer, and could not solve all problems (i.e., it exceeded the 600 second time limit). The same results were obtained with the partial-order problems, except that LAMA took a bit longer but still was far faster than HPLAN-P. This suggests that our domains are reasonably challenging.

To address our second objective we turned to preference-based planner HPLAN-P. We created different versions of the same problem by increasing the number of preferences they used. In particular, for each problem we tested with 10, 20, and 30 preferences. To measure the change in computation time between problems with different numbers of preference, we calculated the percentage difference between the computation time for the problem with the larger and with the smaller number of preferences, all relative to the computation time of the larger numbered problem. The average percentage difference was 6.5% as we increased the number of preferences from 10 to 20, and was 3.5% as we went from 20 to 30 preferences. The results suggest that as we increase the number of preferences, the time it takes to find a solution does increase but this increase is not significant.

As noted previously the Approach 1 compilation technique (including Step 4) results in the complete removal of temporal modalities and therefore enables the use of the Keyder and Geffner compilation technique (2009). This techniques supports the computation of preference-based plans (and now preferred explanations) using cost-based planners. However, the output generated by our compilation requires a planner compatible with ADL or derived predicates. Among the rather few that support any of these, we chose to experiment with LAMA since it is currently the best-known cost-based planner. Figure 1 shows the time it takes to find the optimal explanation using HPLAN-P and LAMA as well as the time comparison between our "Approach 1" and "Approach 2" encodings (Section 5). To measure the gain in computation time from the "Approach 2" technique, we computed the percentage difference between the two, relative to "Approach 1". (We assigned a time of 600 to those marked NF.) The results show that on average we gained 22.9% improvement for HPLAN-P and 29.8 % improvement for LAMA in the time it takes to find the optimal solution. In addition, we calculated the time ratio ("Approach 1"/ "Approach 2"). The results show that on average HPLAN-P found plans 2.79 times faster and LAMA found plans 31.62 times faster when using "Approach 2". However, note that "Approach 2" does not always improve the performance. There are a few cases where the planners take longer when using "Approach 2". While the definite cause of this decrease in performance is currently unknown, we believe this decrease may depend on the structure of the problem and/or on the difference in the size of the translated domains. On average the translated problems used in "Ap-

	Total-Order				Partial-Order			
	HPLAN-P		LAMA		HPLAN-P		LAMA	
	Appr 1	Appr 2	Appr 1	Appr 2	Appr 1	Appr 2	Appr 1	Appr 2
computer-1	1.05	0.78	5.29	0.25	2.26	0.58	5.93	0.57
computer-2	5.01	4.88	0.19	0.41	1.49	1.42	0.50	0.42
computer-3	23.44	22.85	0.75	0.75	15.92	15.57	1.02	1.94
computer-4	55.69	51.98	6.94	4.58	15.97	13.93	3.64	4.33
computer-5	128.50	125.98	2.05	3.20	57.28	56.19	3.42	6.12
computer-6	83.17	82.78	2.64	4.63	43.92	43.86	16.99	16.27
computer-7	505.73	484.68	4.23	5.85	188.45	181.44	89.03	68.47
computer-8	236.03	205.81	3.75	6.13	159.92	152.49	29.35	28.91
car-1	1.60	1.53	0.66	0.08	0.60	0.53	2.11	0.07
car-2	8.96	8.31	10.72	0.20	3.04	2.59	15.14	0.25
car-3	563.60	40.17	13.98	0.59	593.10	15.06	16.51	0.62
car-4	NF	103.80	24.00	1.41	NF	38.48	33.79	0.95
car-5	NF	245.69	35.93	1.56	NF	103.18	NF	1.23
car-6	NF	522.50	117.45	2.44	NF	176.11	NF	1.56
car-7	NF	NF	62.00	3.47	NF	170.54	NF	2.02
car-8	NF	NF	108.07	4.46	NF	257.10	NF	2.94
power-1	0.02	0.01	0.02	0.02	0.82	0.53	0.02	0.02
power-2	0.18	0.18	0.13	0.06	0.14	0.13	0.40	0.06
power-3	0.47	0.50	0.13	0.13	0.31	0.33	3.50	0.11
power-4	1.62	1.52	0.63	0.58	75.37	69.85	14.92	18.37
power-5	26.98	24.60	5.97	0.97	NF	NF	46.43	0.64
power-6	51.65	51.48	11.26	6.84	NF	NF	NF	NF
power-7	177.58	177.42	15.09	9.42	NF	NF	NF	NF
power-8	565.77	564.71	30.67	16.38	NF	NF	NF	NF
truck-1	1.90	1.62	0.13	0.29	3.08	1.98	0.24	0.24
truck-2	5.25	5.10	0.85	0.74	3.12	3.07	0.32	0.49
truck-3	108.83	92.57	0.38	1.07	36.92	27.57	0.59	1.15
truck-4	323.18	323.06	2.03	2.48	402.96	219.73	2.15	1.87
truck-5	177.68	174.22	3.31	4.93	NF	NF	2.71	3.90
truck-6	NF	NF	2.69	1.88	NF	NF	8.53	6.14
truck-7	NF	NF	10.23	11.92	NF	NF	8.42	8.41
truck-8	NF	NF	11.60	8.76	NF	NF	8.19	11.81

Figure 1: Runtime comparison between HPLAN-P and LAMA on problems of known optimal explanation. NF means the optimal explanation was not found within the time limit of 600 seconds.

proach 2" are 1.4 times larger, hence this increase in the size of the problem may be one reason behind the decrease in performance. Nevertheless, this result shows that "Approach 2" can significantly improve the time required to find the optimal explanation, sometimes by orders of magnitude, in so doing it allows us to solve more problem instances than with "Approach 1" alone (see car-4 and car-5).

7. Summary

In this paper, we examined the task of generating preferred explanations. To this end, we presented a logical characterization of the notion of a (preferred) explanation and established its correspondence to planning, including the complexity of explanation generation. We proposed a finite variant of LTL, f-PLTL, that includes past modalities and action occurrences and utilized it to express observations and preferences over explanation. To generate explanations using state-of-the-art planners, we proposed and implemented a compilation technique that preserves f-PLTL semantics while conforming to PDDL3 syntax. This enables computation of preferred explanations with PDDL3-compliant preference-based planners as well as with cost-based planners. Exploiting the property that observations are known a priori we transformed explanation preferences into a form that was amenable to heuristic search. In so doing, we were able to reduce the time required for explanation generation by orders of magnitude, sometimes.

Acknowledgements

We thank Alban Grastien and Patrik Haslum for providing us with an encoding of the computer problem, which we modified and used in this paper for benchmarking. We also gratefully acknowledge funding from the Natural Sciences and Engineering Research Council of Canada (NSERC). Jorge Baier was funded by the VRI-38-2010 grant from Universidad Católica de Chile.

References

Baier, J., and McIlraith, S. 2006. Planning with first-order temporally extended goals using heuristic search. In *Proceedings of the 21st National Conference on Artificial Intelligence (AAAI)*, 788–795.

Baier, J., and McIlraith, S. 2008. Planning with preferences. *AI Magazine* 29(4):25–36.

Baier, J.; Bacchus, F.; and McIlraith, S. 2009. A heuristic search approach to planning with temporally extended preferences. *Artificial Intelligence* 173(5-6):593–618.

Brewka, G. 1994. Adding priorities and specificity to default logic. In *Proceedings of the Logics in Artificial Intelligence, European Workshop (JELIA)*, 247–260.

Bylander, T. 1994. The computational complexity of propositional STRIPS planning. *Artificial Intelligence* 69(1-2):165–204.

de Giacomo, G., and Vardi, M. Y. 1999. Automata-theoretic approach to planning for temporally extended goals. In Biundo, S., and Fox, M., eds., *ECP*, volume 1809 of *LNCS*, 226–238. Durham, UK: Springer.

Eisner, C.; Fisman, D.; Havlicek, J.; Lustig, Y.; McIsaac, A.; and Campenhout, D. V. 2003. Reasoning with temporal logic on truncated paths. In *Proceedings of the 15th International Conference on Computer Aided Verification (CAV)*, volume 2725 of *LNCS*. Boulder, CO: Springer. 27–39.

Gabbay, D. M. 1987. The declarative past and imperative future: Executable temporal logic for interactive systems. In *Temporal Logic in Specification*, 409–448.

Gerevini, A.; Haslum, P.; Long, D.; Saetti, A.; and Dimopoulos, Y. 2009. Deterministic planning in the 5th int'l planning competition: PDDL3 and experimental evaluation of the planners. *Artificial Intelligence* 173(5-6):619–668.

Göbelbecker, M.; Keller, T.; Eyerich, P.; Brenner, M.; and Nebel, B. 2010. Coming up with good excuses: What to do when no plan can be found. In *Proceedings of the 20th International Conference on Automated Planning and Scheduling (ICAPS)*, 81–88.

Grastien, A.; Anbulagan; Rintanen, J.; and Kelareva, E. 2007. Diagnosis of discrete-event systems using satisfiability algorithms. In *Proceedings of the 22nd National Conference on Artificial Intelligence (AAAI)*, 305–310.

Haslum, P., and Grastien, A. 2009. Personal communication.

Haslum, P., and Grastien, A. 2011. Diagnosis as planning: Two case studies. In *Proceedings of the International Scheduling and Planning Applications workshop (SPARK)*.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research* 14:253–302.

Hsu, C.-W., and Wah, B. 2008. The SGPlan planning system. In *6th International Planning Competition Booklet (IPC-2008)*.

Iwan, G. 2001. History-based diagnosis templates in the framework of the situation calculus. In *Proceedings of the Joint German/Austrian Conference on Artificial Intelligence (KR/ÖGAI)*. 244–259.

Keyder, E., and Geffner, H. 2009. Soft Goals Can Be Compiled Away. *Journal of Artificial Intelligence Research* 36:547–556.

Markey, N. 2003. Temporal logic with past is exponentially more succinct, concurrency column. *Bulletin of the EATCS* 79:122–128.

McGuinness, D. L.; Glass, A.; Wolverton, M.; and da Silva, P. P. 2007. Explaining task processing in cognitive assistants that learn. In *Proceedings of the 20th International Florida Artificial Intelligence Research Society Conference (FLAIRS)*, 284–289.

McIlraith, S., and Scherl, R. B. 2000. What sensing tells us: Towards a formal theory of testing for dynamical systems. In *Proceedings of the 17th National Conference on Artificial Intelligence (AAAI)*, 483–490.

McIlraith, S. 1998. Explanatory diagnosis: Conjecturing actions to explain observations. In *Proceedings of the 6th International Conference of Knowledge Representation and Reasoning (KR)*, 167–179.

Ramírez, M., and Geffner, H. 2009. Plan recognition as planning. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI)*, 1778–1783.

Richter, S.; Helmert, M.; and Westphal, M. 2008. Landmarks revisited. In *Proceedings of the 23rd National Conference on Artificial Intelligence (AAAI)*, 975–982.

Sohrabi, S.; Baier, J.; and McIlraith, S. 2010. Diagnosis as planning revisited. In *Proceedings of the 12th International Conference on the Principles of Knowledge Representation and Reasoning (KR)*, 26–36.

The Method of ILP+ASP on Psychological Models

J. Romero, A. Illobre, J. Gonzalez and R. Otero
AI Lab. Computer Science Department, University of Corunna (Spain)
{jromerod,infjdi00,jgonzalezi,otero}@udc.es

Abstract

We propose to apply a new method of Inductive Logic Programming (ILP) and Answer Set Programming (ASP) to Experimental Psychology. The idea is to use ILP to build a model from experimental data and then use ASP with the resulting model to solve reasoning tasks as explanation or planning. For learning in dynamic domains without the frame problem we use the method of [Otero, 2003] and for reasoning in dynamic domains without the frame problem we use actions in ASP [Lifschitz, 2002]. We have applied this method to an experiment in a dynamic domain of Human Reasoning and Decision Making. The results show that the method can be used for learning and reasoning in real-world dynamic domains, thus improving the methods used in Experimental Psychology, that do not consider these problems.

1 Introduction

The objective of Experimental Psychology is building models of human behavior supported by experimental data. These models are often incomplete and not formally defined, and usually a method of linear regression is used to complete and formalize them. In this paper we propose to use logic programming methods instead. To our knowledge there are few previous attempts to use symbolic methods in this area [Gigerenzer and Selten, 2002][Balduccini and Girotto, 2010].

The idea of the method is to apply Inductive Logic Programming (ILP) to build a psychological model and then apply Answer Set Programming (ASP) with the resulting model to solve reasoning tasks. For induction in dynamic domains without the frame problem [McCarthy and Hayes, 1969] we use the method of [Otero, 2003] and for reasoning in dynamic domains without the frame problem we use actions in ASP [Lifschitz, 2002].

We have applied this method to an experiment in a dynamic domain of Human Reasoning and Decision Making (HRDM), a field of Experimental Psychology. The objective of the experiment is to study how people select the strategies for solving repeatedly a given task. We use ILP to automatically build a model of the process of strategy selection, and then use ASP to reason about the model.

In the next section we introduce the logic programming methods that are used in the method of ILP+ASP. Then in section 3 we describe the method of Experimental Psychology and the method of ILP+ASP, and in section 4 we show the application of the proposed method to HRDM. Finally, section 5 presents conclusions and future work.

2 Logic programming methods

Inductive Logic Programming. ILP [Muggleton, 1995] is an area of Machine Learning for the induction of hypothesis from examples and background knowledge, using logic programming as a single representation for them. Inverse Entailment (IE) is a correct and complete ILP method proposed by S. Muggleton that can deal with recursive rules, implemented in the ILP systems Progol [Muggleton, 1995] and Aleph[1]. Given a set of examples and background knowledge, these systems can find the simplest hypothesis that explains every example and is consistent with the background. ILP has been successfully applied in other areas of science such as Molecular Biology (see for example [King *et al.*, 1996]).

Inductive Logic Programming for Actions. Induction of the effects of actions consists in learning an action description of a dynamic system from evidence on its behavior. General logic-based induction methods can deal with this problem but, unfortunately, most of the solutions provided have the frame problem. Instead we propose to use the method of [Otero, 2003], implemented in the system Iaction [Otero and Varela, 2006]. This is a correct, complete and efficient method for induction of action descriptions that can cope with the frame problem in induction.

Answer Set Programming. ASP is a form of logic programming based in the stable models (answer set) semantics [Gelfond and Lifschitz, 1991]. An ASP program can have none, one or several answer sets that can be computed with an ASP system (e.g., Clasp[2]). The method of ASP is (1) to encode a problem as an ASP program such that solutions of the problem correspond to answer sets of the program, and (2) to use an ASP system to compute answer sets of the program.

Answer Set Programming for Actions. ASP is suitable for representing action descriptions without the frame problem, and it can be used to solve different tasks like prediction,

[1] http://www.comlab.ox.ac.uk/activities/machinelearning/Aleph

[2] http://www.cs.uni-potsdam.de/clasp/

diagnosis and planning [Lifschitz, 2002]. For example, the Decision Support System of the Space Shuttle [Nogueira *et al.*, 2000] is an ASP system capable of solving planning and diagnostic tasks related to the operation of the Space Shuttle.

3 The Method of ILP+ASP

In this section we explain the method of Experimental Psychology, we present the method of ILP+ASP and describe its application in detail to an example of Experimental Psychology.

3.1 The Method of Experimental Psychology

The method of Experimental Psychology follows these steps:

Step 1. Psychological Theory
First a psychological theory about human behavior is proposed.

For example, the Theory of Planned Behavior [Ajzen, 1985] states that human behavior can be modeled by the following concepts:

- Intention: the intention to perform the behavior.
- Perceived Behavioral Control (control): the perceived ease or difficulty of performing the behavior.

According to this theory, behavior is related to intention and control but the particular relation is not known in advance, and it is assumed to depend on the type of behavior. In this example we will consider ecological behavior, i.e. human behavior that is relevant for environmental issues: waste management, water and energy consumption, etc.

Step 2. Representation
The concepts of the proposed theory are represented formally to be used in the step of model construction. In Experimental Psychology it is usual to represent the concepts as variables with several values.

For example, behavior is not represented as a boolean variable that states whether a given behavior is performed or not, but instead it has values ranging from 1 to 5, which represent how often the behavior is performed. The same holds for intention and control.

Step 3. Data Collection
Models are based in experimental data, that can be collected following different methods.

For example, we do a survey to many persons in which we ask them whether they performed a given ecological behavior and what was their intention and their perceived behavioral control towards that behavior. Each answer is then quantified so that for each person we have a value for behavior, intention and control. Example data is shown in table 1, where each row corresponds to a subject and each column corresponds to a concept. For example, subject $s1$ has behavior 1, intention 2 and control 1.

Step 4. Model Construction
A model with the particular relation among the concepts is built. Typically it is used a method of linear regression, that uses the representation chosen in step 2 and the data collected in step 3.

Subject	**Behavior**	**Intention**	**Control**
s1	1	2	1
s2	2	3	2
s3	3	3	4
s4	5	5	4

Table 1: Behavior, intention and control for 4 subjects.

For example, the resulting model for behavior may consist of the following equation:

$$behavior = 1 * intention + 0.4 * control - 1.5 \qquad (1)$$

Step 5. Reasoning
The model built can be used to predict the behavior of other persons.

For example, if someone's intention is high (4) but control is very low (1), the model predicts its behavior will be medium (2.9).

Actions
Actions can modify the behavior of people. For example, giving a course on ecology may promote ecological behavior, and removing recycling bins may diminish it. The behavior and the other concepts can be measured before and after the execution of the actions. Again the relation between these actions and the concepts of the theory is not known in advance. We know that some instances of these actions have been done on different people and we want to know the particular relation that holds on every one, which may depend on conditions of the particular person. This is a problem of learning the effects of actions, and the result is a model of the particular relation among the actions and the concepts of the theory. This problem is not considered in the method of Experimental Psychology.

3.2 The Method of ILP+ASP

The method of ILP+ASP can be outlined as follows:

1. Substitute the numerical representation by a logic programming representation such that the final model is a logic program instead of a linear equation.
2. Substitute the linear regression method by a method for induction of logic programs (ILP). Thus the logic program solution is built from instance data of the survey by a Machine Learning method.
3. Substitute the use of the model for prediction by reasoning with ASP. Thus we can do additional relevant tasks like explanation and planning, which are not considered in the method of Experimental Psychology.

To justify the correctness and significance of the ILP+ASP method consider the following:

1. Logic programs provide a representation more general than linear equations. The relation among the variables may not be as simple as a linear equation, and a logic program can represent alternative relations, e.g. not continuous, which could be the case in the domain of Psychology. Note that logic programming allows the repre-

sentation of simple numerical formulas, e.g. linear equations, so we are clearly improving the form of representation.

2. ILP is a correct and complete method of induction for logic programs from instances. Thus the result will have the correction at the same level as linear regression has. The method has the power to identify a model if it exists or to tell us that there is no model, i.e. to validate the psychological theory on experimental data.
3. ASP is able to use the model built, which is a logic program, for tasks done with linear equations like prediction, but also for other additional relevant tasks like explanation and planning.

In summary, the correctness of ILP provides the correctness of the method when building the model and the correction of ASP provides the correctness of the method when using the model.

3.3 An example

Next we explain the particular steps of the method of ILP+ASP with an example.

Step 1. Psychological Theory

This step is the same as in the method of Experimental Psychology. We apply the Theory of Planned Behavior to study ecological behavior.

Step 2. Representation

Define the concepts of the theory as predicates of a logic program. In this example:

- $behavior(S, X)$: subject S behaves ecologically with degree X. For example, $behavior(s4, 5)$ represents that $s4$ behaves very ecologically.
- $intention(S, X)$: subject S has the intention to be ecological with degree X.
- $control(S, X)$: subject S perceives that it is easy for her to be ecological to a degree X.

Step 3. Data Collection

This step is the same as in the method of Experimental Psychology: a survey is done and the results are those of table 1.

Step 4. Model construction

Apply an ILP system (in this example, Progol 4.4) to automatically build a model.

Progol constructs logic programs from examples and background knowledge. In this case there are 4 examples of behavior that we represent as ground facts (rules without body or variables) with predicate *behavior*:

```
behavior(s1,1). behavior(s2,2).
behavior(s3,3). behavior(s4,5).
```

Progol uses the background knowledge to construct rules that explain these examples. The background knowledge encodes the knowledge that the expert thinks that is relevant for the learning process, e.g., part of a psychological theory. In this example we represent the other concepts of the theory with predicates *intention* and *control* and we add predicates *gteq* and *lteq* to allow Progol to make comparisons with different values of those concepts:

```
intention(s1,2). control(s1,1). intention(s2,3). control(s2,2).
intention(s3,3). control(s3,4). intention(s4,5). control(s4,4).

gteq(X,Y) :- value(X), value(Y), X >= Y.
lteq(X,Y) :- value(X), value(Y), X =< Y.
```

These sentences, together with the code below, tell Progol to construct a definition for the *behavior* predicate.

```
subject(s1). subject(s2). subject(s3). subject(s4).
value(1). value(2). value(3). value(4). value(5).

:- modeh(1,behavior(+subject,+value))?
:- modeb(*,intention(+subject,-value))?
:- modeb(*,control(+subject,-value))?
:- modeb(*,gteq(+value,#value))?
:- modeb(*,lteq(+value,#value))?

:- set(inflate,1000)?
:- set(nodes,1000)?

:- behavior(S,X), behavior(S,Y), not(X==Y).
```

The first two lines define some predicates, called types, used in the mode declarations. The following sentences, called modes, describe the predicates that the system can use in the head of the learned rules (*modeh* declarations) or in the body (*modeb* declarations)[3]. The *modeh* declaration states that the head of a learned rule has predicate symbol $behavior$ and two parameters, one of type subject and another of type value. The meaning of the *modeb* declarations is very similar, but they refer to the predicates that can appear in the body of the rules learned: *intention*, *control*, *gteq* and *lteq*. Sentences with the *set* predicate are used to configure the search of the hypothesis. The final sentence states that a subject cannot have two levels of behavior.

Given these sentences Progol finds two rules:

```
behavior(S,X) :- control(S,X), lteq(X,2).
behavior(S,X) :- intention(S,X), control(S,Y), gteq(Y,3).
```

If the control of a subject is less or equal to 2, behavior has the same value as control, and in other case the behavior is equal to the intention. The rules are a solution to induction because they explain every example and they are consistent with the background.

We can compare these rules with the linear equation of section 3.1. Only in 6 out of 25 pairs of values of intention and control their predictions differ in more than one unit, so they predict similar values. But the result of Progol is more insightful. The linear equation simply states how much does every change in intention or control contribute to a change in behavior, while the rules of Progol provide more information about how do these changes happen: when control is very low it blocks the behavior, and in other case the behavior is determined by the intention.

There is an underlying problem in the construction of the model. Surveys are not a safe instrument to measure the concepts of the theory: some people can give answers that, intentionally or not, are false. We can handle this problem with Progol allowing it to predict incorrectly a given proportion of the examples. Besides, the method of ILP+ASP provides a very precise way to detect outliers. To improve the quality of

[3] For further details we refer the reader to [Muggleton, 1995].

the surveys psychologists can introduce related questions, so that some combinations of answers are inconsistent. For example, question *q1* could be *"Do you recycle the paper?"* and question *q2* could be *"Do you recycle anything?"*. If someone answers 5 (always) to *q1* and 1 (never) to *q2* that person is answering inconsistently. ASP can be used to precisely identify the subjects, outliers, that give inconsistent answers. For example, consider the program

```
q1(s1,2). q2(s1,2). q1(s2,5). q2(s2,1).
q1(s3,5). q2(s3,4). q1(s4,3). q2(s4,3).
outlier(S) :- q1(S,5), q2(S,1).
```

where the first lines represent the answer of different subjects to questions *q1* and *q2* and the last rule is used to detect outliers. This program has a unique answer set, which can be computed with an ASP system like Clasp, that contains the atom *outlier(s2)*, thus detecting the unique outlier of the experiment. The outliers identified can be removed from the data set and studied apart, and the model construction step can be repeated with the new set of examples.

Step 5. Reasoning

The logic program built in the previous step is a model of ecological behavior:

```
behavior(S,X) :- control(S,X), lteq(X,2).
behavior(S,X) :- intention(S,X), control(S,C), gteq(C,3).
gteq(X,Y) :- value(X), value(Y), X >= Y.
lteq(X,Y) :- value(X), value(Y), X <= Y.
value(1). value(2). value(3). value(4). value(5).
```

This program can be used in ASP to solve different tasks.

Prediction. Given the intention and control of a new subject we can use the model to predict her behavior. For example, if subject *s5* has intention 5 and control 2 we add to the previous program the facts:

```
intention(s5,5). control(s5,2).
```

The resulting program has a unique answer set that can be computed by Clasp and contains the prediction for the behavior of *s5*:

```
behavior(s5,2)
```

Explanation. Given the behavior of a new subject and possibly some additional information we can use the model to explain her behavior. For example, we want to explain why *s6* has behavior 5, and we know her control is 3. For this task we add the following sentences:

```
1 { intention(s6,1), intention(s6,2), intention(s6,3),
    intention(s6,4), intention(s6,5)} 1.
control(s6,3). behavior(s6,5).
:- behavior(S,X), behavior(S,Y), X!=Y.
```

The first rule forces to choose among one of the possible values of intention, the next rule represents the known data, and the last one, like the one we used in Progol, eliminates the answer sets that predict two different values for behavior. The output of Clasp:

```
intention(s6,5), control(s6,3), behavior(s6,5)
```

gives the explanation for the very high behavior: the intention is also very high.

3.4 An example in Actions

We explain the application of the method of ILP+ASP to dynamic domains.

Step 1. Psychological theory

We consider the application of the Theory of Planned Behavior to study the effects of actions on ecological intention and on ecological control.

Step 2. Representation

Define predicates for the actions and for the concepts of the theory that change over time, called fluents. Each instant of time is called a situation, and situations range from 0 (initial situation) to *n* (final situation), where *n* depends on each experiment. Predicates now have a new term to represent the situation in which the action took place or the situation in which the fluent value holds.

Actions. Two actions may change the fluents:

- $course(S)$: a course on ecology is given at situation *S*.
- $car\_sharing(S)$: a project for sharing cars to go to work is done at situation *S*.

Fluents. We modify the predicates of the static case:

- $intention(S, A, X)$: at *S* subject *A* has intention to be ecological with degree *X*. For example, $intention(0, s1, 5)$ represents that at 0 subject $s1$ has a high ecological intention, and $intention(2, s1, 1)$ represents that at 2 her intention is low.
- $control(S, A, X)$: at *S* subject *A* perceives that it is easy for her to be ecological to a degree *X*.

Step 3. Data Collection

To study the effects of actions we do surveys at different situations. In this example a survey was done to 2 subjects, then a course on ecology was given and another survey was done, and finally a car sharing project was done followed by another survey. The next program represents the data:

```
intention(0,s1,3). control(0,s1,2).
intention(0,s2,2). control(0,s2,3).
course(1).
intention(1,s1,3). control(1,s1,5).
intention(1,s2,2). control(1,s2,5).
car_sharing(2).
intention(2,s1,5). control(2,s1,5).
intention(2,s2,2). control(2,s2,5).
```

Step 4. Model construction

Apply system Iaction [Otero and Varela, 2006] to automatically build a model of the relations between the actions and the concepts considered. Iaction implements the method of [Otero, 2003]. The syntax and use of Iaction is very similar to that of Progol. For example, the mode declarations for this example are:

```
:- modeh(*,control(+situation,+subject,#value))?
:- modeh(*,intention(+situation,+subject,#value))?
:- modeb(*,course(+situation))? %ACTION
:- modeb(*,car_sharing(+situation,+subject))? %ACTION
:- modeb(*,intention(+situation,+subject,-value))?
:- modeb(*,gteq(+value,#value))?
:- modeb(*,lteq(+value,#value))?
```

Symbol *%ACTION* tells the system which predicates are actions. We have instructed Iaction to induce an action description for fluents control and intention (we use one *modeh* for each). Finally, Iaction finds this action description:

```
intention(S,A,5) :- course(S), prev(S,PS),
                    intention(PS,A,X), gteq(X,3).
control(S,A,5) :- car_sharing(S).
```

The course improves the intention of subjects that had at least medium intention, and the car sharing project improves the control of all subjects. The solution found by Iaction, as guaranteed by the method of [Otero, 2003], explains all the examples and is consistent with the background.

Step 5. Reasoning

We have a description of the effects of actions on fluents control and behavior. From previous experiments we also know what is the relation of behavior with intention and control. In this step we apply ASP to this model for reasoning about actions. For all tasks we use a file representing the domain description and another file representing the particular task. The domain description file contains these rules to represent the changes in the domain:

```
intention(S,A,5) :- course(S), prev(S,PS),
                    intention(PS,A,X), gteq(X,3).
control(S,A,5) :- car_sharing(S).
behavior(S,A,X) :- control(S,A,X), lteq(X,2).
behavior(S,A,X) :- intention(S,A,X),
                   control(S,A,Y), gteq(Y,3).
```

Rules for intention and control are the result of the previous learning process, and rules for behavior are known from previous experiments. For each fluent we have to add the indirect effects for the negation of the fluent and the inertia law. For example, for fluent behavior we add rules:

```
-behavior(S,A,X) :- behavior(S,A,Y), X!=Y.

behavior(S,A,X) :- behavior(PS,A,X),
                   not -behavior(S,A,X), prev(S,PS).
-behavior(S,A,X) :- -behavior(PS,A,X),
                    not behavior(S,A,X), prev(S,PS).
```

This domain description can be used for solving different tasks.

Prediction. Given the state of a domain at an initial situation and a sequence of actions, the objective of prediction is to determine the state of the domain in the final state and others. For example, at initial situation subject *s3* has low intention and control. Then a car sharing project is done in his workplace and after that he goes to a course on ecology. We can represent this adding the following program to the domain description:

```
situation(0..2). subject(s3).
intention(0,s3,2). control(0,s3,2).
car_sharing(1).
course(2).
```

This program has a unique answer set that solves the prediction problem. This is part of the output of Clasp:

```
... behavior(2,s3,2) intention(2,s3,2) control(2,s3,5) ...
```

The course had no effect on the intention of *s3* so even if the car sharing project increased her control the behavior remains low.

Planning. Given the initial and the final state of a domain, the objective of planning is to find a sequence of actions that leads from the initial state to the final one. For example, subject *s4* has low behavior, medium intention and low control, and we want to find a sequence of actions that can make him become very ecological. This problem can be represented adding the next program to the domain description:

```
situation(0..2). subject(s4).
intention(0,s4,3). control(0,s4,2).
1 { course(S), car_sharing(S) } 1 :- prev(S,PS).
:- not behavior(2,s4,5).
```

The line with brackets states that each answer set must contain one and only one of the atoms inside, i.e. for each situation we must choose one of the actions. The last line defines the goal of the planning problem. The program has two answer sets that represent the solutions to the planning problems. This is part of the output of Clasp:

```
Answer: 1
... course(1) car_sharing(2) ...
Answer: 2
... car_sharing(1) course(2) ...
```

To improve the behavior it is necessary to improve both intention and control, and to this aim both *course* and *car_sharing* actions have to be executed. However, if instead of *intention(0,s4,3)* we write *intention(0,s4,2)* the program has no answer set and thus there is no solution to the planning problem: the intention is too low to be improved giving a course, so there is no way to improve her behavior.

4 An experiment on Human Reasoning and Decision Making

The theory of the *Adaptive Toolbox* ([Gigerenzer and Selten, 2002] proposes that human reasoning and decision making can be modeled as a collection of *fast and frugal* heuristics. A fast and frugal heuristic is a simple algorithm to both build a model and make predictions on a domain.

Under this hypothesis people use one of these fast and frugal heuristics to decide the solution to a problem. However, the mechanism by which a subject would select one of the heuristics is still under study. It is also possible that the same subject, trying to solve a similar problem several times, uses different heuristics in different moments.

In [Rieskamp and Otto, 2006] SSL (*Strategy Selection Learning*), a theory based on reinforcement learning, is proposed. It explains how people decide to apply one heuristic depending on feedback received. The theory is tested on 4 experimental studies. We apply the ILP+ASP method to one of these experiments to: 1) model how a subject decides to use one of the heuristics available in a given situation, and 2) use this model to solve prediction and planning tasks.

Step 1. Psychological Theory

Suppose we have two unnamed companies, *A* and *B*, and we want to decide which one is the most creditworthy. Each company is described by 6 binary cues (Financial Flexibility, Efficiency, Capital Structure...). For each cue a *validity* value, representing the Probability of success, is also available. Both companies with their respective cues are showed to a subject, see table 2. The subject, based on this information, has to

Cue	Validity	A	B
Financial Flexibility	79%	1	0
Efficiency	90%	1	1
Capital Structure	75%	0	0
Management	70%	1	0
Own Financial Resources	85%	1	0
Qualifications of Employees	60%	1	0

Table 2: Example of a possible trial. A subject will select company *A* or company *B* as the most creditworthy. After the subject's choice, feedback saying if the answer is correct or incorrect is given.

decide whether the company *A* or the company *B* is the most creditworthy. After the subject's choice it is shown if the answer is correct or not. Then another two unnamed companies are presented to the same subject and the previous process is repeated, each repetition is named a *trial*. The objective of the experiment is to model how a subject decides which company is the most creditworthy.

In this study, subjects are shown 168 selection problems, *trials*. The study is divided in 7 trial blocks, each consisting on 24 trials. In each trial block, the same 24 pairs of companies, *items*, are shown to the subjects, but its order is varied. The order of each company in the screen is varied too, e.g. suppose that the trial involves two companies *o1* and *o2*, in some trials the company *o1* is named *A* and it is showed on the left side (the column under the label *A* in the table 2), while the company *o2* is named *B* and it is showed on the right (the column under the label *B*), while in other trials the company *o1* is named *B* and showed on the right, and the company *o2* is named *A* and showed on the left.

For each cue c_i (e.g. Efficiency) the value *1* for a given company *O* represents that *O* has the property c_i (e.g. the company *O* is efficient). The value *0* represents that *O* does not have the property c_i (e.g. the company *O* is not efficient). Each cue c_i has a value, *1* or *0*, associated to each company *A* and *B*.

Also each cue has associated a *validity* value, representing the *Probability of success*. For example, in the table 2 for the first cue *Financial Flexibility* the company *A* has the property Financial Flexibility (*1*), the company *B* has not the property Financial Flexibility (*0*) and the validity for this cue is 79%. It means that the probability of choosing the right answer, if you select the company with *Financial Flexibility*, is of 0.79.

In the experiment it is assumed that, for each trial, subjects decide which is the most creditworthy company based on the results of one of these two heuristics: Take the Best (TTB) or Weighted Additive Heuristic (WADD) [Gigerenzer and Selten, 2002]. However a subject can use TTB in one trial and WADD in another trial. Note that the subject decides to select company *A* or company *B* in each trial. Hence it is not directly known which of the two heuristic has been used by the subject. In this study it is assumed that, if only one heuristic selects the same object as the subject, then the subject must necessarily have used that heuristic. For example, if the TTB heuristic selects company A, the WADD heuristic selects company B, and the subject has selected the company A, then it is assumed that he has used the TTB heuristic. In any other case, the heuristic selected by the subject (from TTB or WADD) is unknown.

Summarizing, for each subject in the experiment it is shown to him a sequence of trials. In each trial the subject has to select the company *A* or the company *B* as the most creditworthy. It is assumed that the subject has used the heuristic (TTB or WADD) which selects the same answer as the subject. After each trial feedback showing if the answer was correct or incorrect is shown. The objective is to model which heuristic is used by a subject in each trial.

Step 2. Representation

We define the trials of the survey, and the answers given by the subjects, with the following predicates of a logic program.

Actions. For this experiment, it is enough to define a single action predicate, *show*.

- *show(T,Sb,I,P):* item *I* is shown to subject *Sb* on trial *T*. The parameter *P* represents if the correct company is at the *left* or *right* of the computer screen.

Fluents. To represent the decisions of the subject on each trial, and the information she might keep to decide which heuristic to apply, we define the following fluents:

- *selectedheuristic(T,Sb,H):* subject *Sb* used heuristic *H* (TTB or WADD) on trial *T*. The system Iaction [Otero and Varela, 2006] will learn an action description for this fluent.
- *itemselectedheuristic(T,Sb,I,H):* subject *Sb* used heuristic *H* the last time item *I* was shown.
- *answeredcorrectly(T,Sb,D)*: subject *Sb* answered correctly on trial *T*. Parameter *D* is used to keep information from previous trials, that the subject might use to take a decision. For example, *answeredcorrectly(T,Sb,d0)* represents that the subject answered correctly at trial *T*, and *answeredcorrectly(T,Sb,d1)* that the subject answered correctly at $T - 1$.
- *itemansweredcorrectly(T,Sb,I):* subject *Sb* answered correctly the last time item *I* was shown.
- *selected(T,Sb,O,D):* subject *Sb* selected company *O* on trial *T*.
- *itemselected(T,Sb,I,O)*: subject *Sb* selected company *O* the last time item *I* was shown.
- *selectedposition(T,Sb,P)*: subject *Sb* selected the company at position *P* of the computer screen on trial *T*.
- *itemselectedposition(T,Sb,I,P):* subject *Sb* selected the company at position *P* of the computer screen the last time item *I* was shown.
- *showed(T,Sb,I,P,D):* item *I* was shown on trial *T*.
- *feedback(T,Sb,H,R,D):* the subject might have chosen the company selected by *ttb*, *wadd*, or *any* of the two (parameter *H*). This decision was either *correct* or *incorrect* (parameter *R*).

- *sfeedback(T,Sb,H,R,C,D):* counts the feedback received by the subject on the last *D* trials. For example, *sfeedback(T,Sb,ttb,incorrect,2,d2)* represents that the subject used the TTB heuristic in T and $T - 1$, and both times received negative feedback.

Static background. The following static background is defined to represent companies, their cue values, items and the selection that the TTB and WADD heuristic would make for each of them.

- *within(I,P,O1,O2)*: companies *O1* and *O2* are within item *I*. *P* represents the position of the screen where the correct company appears.
- *correctobject(I,O):* company *O* is the correct answer for item *I*.
- *selects(H,I,O):* heuristic *H* selects company *O* for item *I*.
- *only(I,O,C,V):* company *O* is the only within item *I* that has a value of *V* for cue *C*. For example, if item *i1* is formed by companies *o1* and *o2*, *only(i1,o1,c1,1)* represents that *o1* has a value of 1 for cue *c1*, while *o2* has a value of 0.
- *same(I,C):* both companies on item *I* have the same value for cue *C*.

Step 3. Data collection

We use the answers of 20 subjects from Study 1 of [Rieskamp and Otto, 2006][4]. For each subject, the results of the survey are represented as ground facts in a logic program, using the predicates of step 2. The following are examples of these:

```
show(t1,s1,i6,left). within(i6,left,o22,o24).
 selectedheuristic(t1,s1,wadd). nansweredcorrectly(t1,s1,d0).
 ...
```

The first fact represents that, at trial $t1$, subject $s1$ was shown item $i6$, with the correct object at the left of the computer screen. The second fact represents that item $i6$ is formed by companies $o22$ and $o24$. Finally, the last two facts represent the answer of the subject: he has used the WADD heuristic and thus answered incorrectly.

Step 4. Model construction

We use the system Iaction [Otero and Varela, 2006] to build an action theory representing how a subject selects heuristics. An action theory is built for each subject. The following is an action theory built by the system:

```
selectedheuristic(T,Sb,ttb):-show(T,Sb,I,P), prev(T,Pt),
        feedback(Ptr,Sb,wadd,incorrect,d0),
        showed(Ptr,Sb,I2,P2,d0),
        only(I2,O,c6,1).
selectedheuristic(T,Sb,ttb):-show(T,Sb,I,P), prev(Tr,Pt),
        nansweredcorrectly(Ptr,Sb,d0),
        csame(I,c6),
        itemselected(Ptr,Sb,I,O1), within(I,P,O1,O2).
selectedheuristic(Tr,Sb,wadd):-show(T,Sb,I,P), prev(Tr,Pt),
        answeredcorrectly(Pt,Sb,d0),
        only(I,O,c5,0),selects(ttb,I,O).
```

Trials		t0	t1		t2	
selectedheuristic		wadd	**wadd**		**ttb**	
answeredcorrectly		yes	**no**		**yes**	
actions (show)			i6,left		i1,right	
companies			o22	o24	o33	o23
cue values	c1		1	0	0	1
	c2		0	1	1	0
	c3		0	0	1	0
	c4		0	1	1	1
	c5		1	1	0	1
	c6		0	1	1	1

Table 3: Predicting heuristic selection problem.

This set of action laws represents what makes a subject select an heuristic.

The first action law states that, if 1) the subject has used the WADD heuristic on the last two trials and answered incorrectly on both cases, and 2) for the last shown item cue c6 in the screen was different for both companies, then she will use the TTB heuristic on the next trial.

The second action law states that, if 1) the subject has answered incorrectly to the last trial, 2) the subject is shown an item where both companies have the same value for cue c6, and 3) the last time this item appeared, she selected the object on the left of the computer screen, then she will select the TTB heuristic.

The third action law states that, if 1) the subject has answered correctly to the last trial, 2) the subject is shown an item where both companies have a different value for cue c5, and 3) the TTB heuristic would select the company with a value of 0 in this cue, then she will apply the WADD heuristic.

Step 5. Reasoning

The action theory built on the previous step is combined with the background defined on step 2. The resulting model can be used to predict, explain and plan for subject answers.

Prediction. Given a sequence of trials, the model can be used to predict which heuristic the subject will use on each of them. For example, consider the problem summarized in table 3. At trial 0 (*t0*), the subject has selected the *WADD* heuristic, and has *answered correctly*. We now know that the subject will have to answer to items *i6* and *i11*, and that the correct object appears in the *left* and *right* of the screen, respectively. Table 3 shows the *companies* within each item, their *cue values*, and which company would be *selected by* the TTB and WADD heuristic. With this information, the goal is to decide which heuristics the subject will use to answer to these two items. To solve this problem we add the following rules to the logic program:

```
selectedheuristic(t0,s1,wadd). answeredcorrectly(t0,s1).
 ....
show(t1,s1,i6,left). show(t2,s1,i1,right).
```

First, we specify the state of the subject at trial $t0$ as a set of ground facts. Then, we specify the sequence of actions that the subject will be shown. With these rules we get a single solution, the prediction shown in table 3:

[4] The authors wish to thank Dr. Jörg Rieskamp for providing the data used in this section.

```
selectedheuristic(t0,s1,wadd) selectedheuristic(t1,s1,wadd)
selectedheuristic(t2,s1,ttb)
```

Planning. In the planning task, the goal is to find a sequence of actions that would make the subject show a certain behavior, e.g. using an heuristic or answering a question incorrectly. As an example, consider the same problem shown in table 1. This time, however, we just know that the subject has used the WADD heuristic at trial 0, and that we want her to use the TTB heuristic on trial 2.

To solve this problem we add the following rules to the program:

```
selectedheuristic(t0,s1,wadd). answeredcorrectly(t0,s1).
....
1 { show(T,Sb,I,P) : item(I) : position(P) } 1 :-
        prev(T,Pt), subject(Sb).
:- selectedheuristic(T,Sb,H),
        selectedheuristic(T,Sb,H2), H!=H2.
:- not selectedheuristic(t2,s1,ttb).
```

First, we specify the state of the subject at trial t0 as in the prediction task. Then, we specify the planning problem using three rules. The first rule defines the set of possible solutions for the task, the second rule grants that the solutions found are consistent, and the last rule represents the goal of the problem. Running Clasp we get all possible solutions. For example:

```
show(t88,s107,i6,left) show(t89,s107,i1,right)
```

that is the same used in the prediction example.

Discussion.
To model the process of strategy selection [Rieskamp and Otto, 2006] have proposed the SSL theory. According to this theory subjects start with an initial preference for each heuristic. At each trial subjects use the most preferred heuristic, and they update their preferences depending on the performance of the heuristics (see [Rieskamp and Otto, 2006] for a formal definition). On a preliminary study SSL correctly predicted 80% of the trials where the result of TTB and WADD is different, and in the same setting the ILP+ASP method predicted correctly 88% of the trials. Note that with ILP+ASP the model is built automatically, without prior knowledge of how the process of strategy selection is done. And the resulting model can provide new insight on this process. For example, the action theory constructed by Iaction suggests that the cue $c6$, that appears in the bottom of the computer screen, could be relevant for the decisions of the subjects. Finally, we have seen how these models can be used in an ASP system to predict and plan about subject answers.

5 Conclusions

We have proposed a new method of ILP+ASP for Experimental Psychology. It is a correct and complete method for induction of logic programs that provides a general form of representation and can be used to solve relevant tasks like explanation and planning. We have applied the method for learning and reasoning in a real-world dynamic domain, thus improving the methods used in Experimental Psychology, that do not consider these problems. As of future work we will apply the method to other fields of Psychology where dynamic domains need to be modeled.

Acknowledgments. This research is partially supported by the Government of Spain, grant AP2008-03841, and in part by the Government of Galicia (Spain), under grant PGIDIT08-IN840C.

References

[Ajzen, 1985] I. Ajzen. From intentions to actions: a theory of planned behavior. *Action-control: from cognition to behavior*, pages 11–39, 1985.

[Balduccini and Girotto, 2010] M. Balduccini and S. Girotto. Formalization of psychological knowledge in answer set programming and its application. *Theory Pract. Log. Program.*, 10:725–740, 2010.

[Gelfond and Lifschitz, 1991] M. Gelfond and V. Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9:365–385, 1991.

[Gigerenzer and Selten, 2002] G. Gigerenzer and R. Selten. *Bounded Rationality: The Adaptive Toolbox*. The MIT Press, 2002.

[King *et al.*, 1996] R.D. King, S.H. Muggleton, A. Srinivasan, and M. Sternberg. Structure-activity relationships derived by machine learning: the use of atoms and their bond connectives to predict mutagenicity by inductive logic programming. *Proceedings of the National Academy of Sciences*, 93:438–442, 1996.

[Lifschitz, 2002] Vladimir Lifschitz. Answer set programming and plan generation. *Artificial Intelligence*, 138:39–54, 2002.

[McCarthy and Hayes, 1969] J. McCarthy and P. J. Hayes. Some philosophical problems from the standpoint of artificial intelligence. In *Machine Intelligence*, pages 463–502. Edinburgh University Press, 1969.

[Muggleton, 1995] S. H. Muggleton. Inverse entailment and progol. *New Generation Computing*, 13:245–286, 1995.

[Nogueira *et al.*, 2000] M. Nogueira, M. Balduccini, M. Gelfond, R. Watson, and M. Barry. An a-prolog decision support system for the space shuttle. In *In PADL 2001*, pages 169–183. Springer, 2000.

[Otero and Varela, 2006] R. Otero and M. Varela. Iaction, a system for learning action descriptions for planning. *Proceedings of the 16th Int. Conference on Inductive Logic Programming, ILP-06. LNAI*, 4455, 2006.

[Otero, 2003] R. Otero. Induction of the effects of actions by monotonic methods. *Proceedings of the 13th Int. Conference on Inductive Logic Programming, ILP-03. LNAI*, 2835:193–205, 2003.

[Rieskamp and Otto, 2006] J. Rieskamp and P. E Otto. Ssl: a theory of how people learn to select strategies. *Journal of Experimental Psychology: General*, 135(2):219–238, 2006.

Tractable Strong Outlier Identification

Fabrizio Angiulli
f.angiulli@deis.unical.it
ECSS Dept.
University of Calabria
Via P. Bucci, 41C, 87036
Rende (CS), Italy

Rachel Ben-Eliyahu–Zohary
rbz@jce.ac.il
Software Engineering Dept.
Jerusalem College of Engineering
Jerusalem, Israel

Luigi Palopoli
palopoli@deis.unical.it
ECSS Dept.
University of Calabria
Via P. Bucci, 41C, 87036
Rende (CS), Italy

Abstract

In knowledge bases expressed in default logic, outliers are sets of literals, or observations, that feature unexpected properties. This paper introduces the notion of *strong* outliers and studies the complexity problems related to outlier recognition in the fragment of *acyclic normal unary theories* and the related one of *mixed unary theories*. We show that recognizing strong outliers in acyclic normal unary theories can be done in polynomial time and, moreover, that this result is sharp, since switching to either general outliers, cyclic theories or acyclic mixed unary theories makes the problem intractable. This is the only fragment of default theories known so far for which the general outlier recognition problem is tractable. Based on these results, we have designed a polynomial time algorithm for enumerating all strong outliers of bounded size in an acyclic normal unary default theory. These tractability results rely on the Incremental Lemma which is also presented. This useful Lemma provides conditions under which a mixed unary default theory displays a *monotonic* reasoning behavior.

1 Introduction

Detecting outliers is a premiere task in data mining. Although there is no universal definition of outlier, it is usually referred to as an observation that appears to deviate markedly from the other observations or to be inconsistent with the remainder of the data (Hawkins, 1980). Applications of outlier detection include fraud detection, intrusion detection, activity and network monitoring, detecting novelties in various contexts, and many others (Hodge & Austin, 2004; Chandola, Banerjee, & Kumar, 2009).

Consider a rational agent acquiring information about the world stated in the form of a sets of facts. It is analogously relevant to recognize if some of these facts disagree with her own view of the world. Obviously such a view has to be encoded somehow using one of the several KR&R formalisms defined and studied in the Artificial Intelligence literature. In particular, for such a formalism to be suitable to attack interesting KR problems it must be nonmonotonic, so that it is possible to naturally exploit defeasible reasoning schemas. Among the nonmonotonic knowledge representation formalisms, Reiter's default logic (Reiter, 1980) occupies a preeminent and well-recognized role.

In a recent paper (Angiulli, Zohary, & Palopoli, 2008) formally defined the outlier detection problem in the context of Reiter's default logic knowledge bases and studied some associated computational problems. Following (Angiulli et al., 2008), this problem can be intuitively described as follows: *outliers* are sets of observations that demonstrate some properties contrasting with those that can be logically "justified" according to the given knowledge base. Thus, along with outliers, their *witnesses*, which are sets of observations encoding the unexpected properties associated with outliers, are singled out.

To illustrate this technique, consider a case where during the same day, a credit card number is used several times to pay for services provided through the Internet. This sounds normal enough, but add to that the interesting fact that the payment is done through different IPs, each of which is located in a different country! It might be the case that the credit card owner is traveling on this particular day, but if the different countries from which the credit card is used are located in different continents we might get really suspicious about who has put his hands on these credit card numbers. Another way to put it, is to say that the fact that the credit card number is used in different continents during the same day makes this credit card an outlier, and one of the probable explanations for such a phenomenon is that the credit card numbers have been stolen. This example is discussed further in Section 3.3.

As noted in (Angiulli et al., 2008),outlier detection problems are generally computationally quite hard, with their associated complexities ranging from D^{P}-complete to $\mathrm{D}_3^{\mathrm{P}}$-complete, depending on the specific form of problem one decides to deal with. For this reason, (Angiulli, Zohary, & Palopoli, 2010) singled out several cases where a very basic outlier detection problem, that is, the problem of recognizing an outlier set and its witness set, can be solved in polynomial time.

A cumulative look at the results presented in (Angiulli et al., 2008, 2010), provides an idea of the tractability frontier associated with outlier detection problems in default logic. In this paper, we continue along this line of research and attempt to draw, as precisely as possible, such a tractability frontier. We want to depict the contour of a tractability region for outlier detection problems that refers to the well-known fragment of unary propositional default theories. In particular, motivated by the intractability of the general outlier recognition problem in all the classes of theories considered thus far in the literature, we investigate this problem within further subsets of the classes already studied, such as the fragment of *acyclic normal unary theories* and the related one of *mixed unary theories*. We also introduce a new type of outliers which we will call *strong outliers*.

Informally speaking, acyclic normal unary theories are normal unary theories characterized by a bounded degree of cyclicity, while strong outliers are outliers characterized by a stronger relationship with their witness set than in the general case. In this context, we have been able to prove that recognizing strong outliers under acyclic normal unary theories can be done in polynomial time and, moreover, that this result is sharp, since switching either to general outliers, to cyclic theories or to acyclic mixed unary theories makes the problem intractable. Notably, this is the only only fragment of default theories known so far for which the general outlier recognition problem is tractable. Based on these results, we designed a polynomial time algorithm for enumerating all strong outliers of bounded size in an acyclic normal unary default theory.

This algorithm can also be employed to enumerate all strong outliers of bounded size in a general normal mixed unary theory and, with some minor modifications, all the general outliers and witness pairs of bounded size. However, in this latter case, since the problems at hand are NP-hard, its worst case running time is exponential, even if from a practical point of view it can benefit from some structural optimizations which allows the algorithm to reduce the size of the search space.

The rest of the paper is organized as follows. Section 2 recalls the definition of default logics and that of the outlier detection task in the framework of default reasoning. Section 3 introduces the definitions of mixed unary and acyclic unary default theories, the definition of strong outlier, and provides a roadmap of the technical results that will be presented in the rest of the paper. Section **??** presents intractability results while Section **??** presents some computational characterizations of mixed unary theories, the tractability result concerning strong outlier recognition that completes the layout of the tractability frontier, and the polynomial time strong outlier enumeration algorithm for acyclic unary default theories. To conclude, Section 4 runs through the complexity results presented in Sections **??** and **??** once more, this time focusing on their complementarity and commenting also upon the application of the outlier enumeration algorithm within more general scenarios. The section ends with our conclusions. Due to space constraints, many proofs are ommited. All the proofs can be found in the full version of the paper, see (Angiulli, Zohary, & Palopoli,).

2 Outlier Detection using Default Logic

Default logic was introduced by Reiter (Reiter, 1980). We first recall basic facts about its propositional fragment. For T, a propositional theory, and S, a set of propositional formulae, T^* denotes the logical closure of T, and $\neg S$ the set $\{\neg(s)|s \in S\}$. A set of literals L is *inconsistent* if $\neg\ell \in L$ for some literal $\ell \in L$. Given a literal ℓ, $letter(\ell)$ denotes the letter in the literal ℓ. Given a set of literals L, $letter(L)$ denotes the set $\{A \mid A = letter(\ell) \text{ for some } \ell \in L\}$.

2.1 Syntax

A *propositional default theory* Δ is a pair (D, W) where W is a set of propositional formulae and D is a set of default rules. We assume that both sets D and W are finite. A *default rule* δ is

$$\frac{\alpha : \beta_1, \ldots, \beta_m}{\gamma} \tag{1}$$

where α (called *prerequisite*), β_i, $1 \leq i \leq m$ (called *justifications*) and γ (called *consequent*) are propositional formulae. For δ a default rule, $pre(\delta)$, $just(\delta)$, and $concl(\delta)$ denote the prerequisite, justification, and consequent of δ, respectively. Analogously, given a set of default rules, $D = \{\delta_1, \ldots, \delta_n\}$, $pre(D)$, $just(D)$, and $concl(D)$ denote, respectively, the sets $\{pre(\delta_1), \ldots, pre(\delta_n)\}$, $\{just(\delta_1), \ldots, just(\delta_n)\}$, and $\{concl(\delta_1), \ldots, concl(\delta_n)\}$. The prerequisite may be missing, whereas the justification and the consequent are required (an empty justification denotes the presence of the identically true literal **true** specified therein).

Next, we introduce some well-known subsets of propositional default theories relevant to our purposes.

Normal theories. If the conclusion of a default rule is identical to the justification the rule is called *normal*. A default theory containing only normal default rules is called *normal*.

Disjunction-free theories. A propositional default theory $\Delta = (D, W)$ is *disjunction free* (DF for short) (Kautz & Selman, 1991), if W is a set of literals, and, for each δ in D, $pre(\delta)$, $just(\delta)$, and $concl(\delta)$ are conjunctions of literals.

Normal mixed unary theories. A DF default theory is *normal mixed unary* (NMU for short) if its set of defaults contains only rules of the form $\frac{\alpha:\beta}{\beta}$, where α is either empty or a single literal and β is a single literal.

Normal and dual normal unary theories. An NMU default theory is *normal unary* (NU for short) if the prerequisite of each default is either empty or positive. An NMU default theory is *dual normal* (DNU for short) unary if the prerequisite of each default is either empty or negative.

Figure 1 highlights the set-subset relationships between the above fragments of default logic.

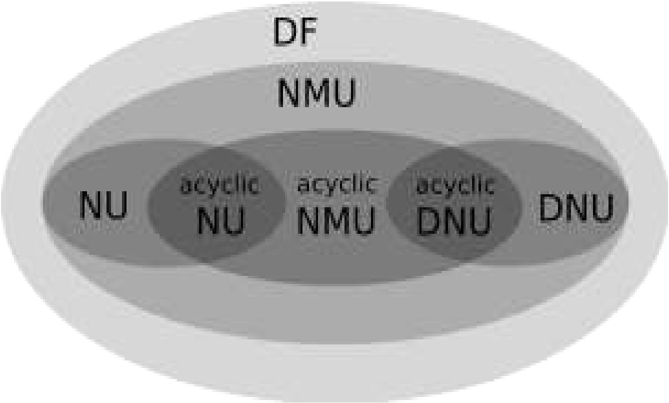

Figure 1: A map of the investigated fragments default theory

2.2 Semantics

The informal meaning of a default rule δ is as follows: If *pre*(δ) is known to hold and if it is consistent to assume *just*(δ), then infer *concl*(δ). The formal semantics of a default theory Δ is defined in terms of *extensions*. A set $\mathcal{E}$ is an extension for a theory $\Delta = (D, W)$ if it satisfies the following set of equations:

- $E_0 = W$,
- for $i \geq 0$, $E_{i+1} = E_i^* \cup \left\{\gamma \mid \frac{\alpha:\beta_1,\ldots,\beta_m}{\gamma} \in D, \alpha \in E_i, \neg\beta_1 \notin \mathcal{E}, \ldots, \neg\beta_m \notin \mathcal{E}\right\}$,
- $\mathcal{E} = \bigcup_{i=0}^{\infty} E_i$.

Given a default δ and an extension $\mathcal{E}$, we say that δ is applicable in $\mathcal{E}$ if $pre(\delta) \in \mathcal{E}$ and $(\nexists c \in just(\delta))(\neg c \in \mathcal{E})$.

It is well known that an extension $\mathcal{E}$ of a finite propositional default theory $\Delta = (D, W)$ can be finitely characterized through the set $D_{\mathcal{E}}$ of the *generating defaults* for $\mathcal{E}$ w.r.t. Δ (Reiter, 1980; Zhang & Marek, 1990).

Next we introduce a characterization of an extension of a finite DF propositional theory which is based on a lemma from (Kautz & Selman, 1991).

Lemma 2.1 *Let $\Delta = (D, W)$ be a DF default theory; then $\mathcal{E}$ is an extension of Δ if there exists a sequence of defaults $\delta_1, ..., \delta_n$ from D and a sequence of sets $E_0, E_1, ..., E_n$, such that for all $i > 0$:*

- $E_0 = W$,
- $E_i = E_{i-1} \cup concl(\delta_i)$,
- $pre(\delta_i) \subseteq E_{i-1}$,
- $(\nexists c \in just(\delta_i))(\neg c \in E_n)$,
- $(\nexists \delta \in D)(pre(\delta) \subseteq E_n \wedge concl(\delta) \not\subseteq E_n \wedge (\nexists c \in just(\delta))(\neg c \in E_n))$,
- *$\mathcal{E}$ is the logical closure of E_n,*

where E_n is called the signature set *of $\mathcal{E}$ and is denoted liter($\mathcal{E}$) and the sequence of rules $\delta_1, ..., \delta_n$ is the set $D_{\mathcal{E}}$ of generating defaults of $\mathcal{E}$. We assume that the set of generating defaults is maximal, that is, for every $\delta \in D$, if δ is applicable in $\mathcal{E}$ then, for some $1 \leq i \leq n$, $\delta = \delta_i$.*

Although default theories are *non-monotonic*, normal default theories satisfy the property of *semi-monotonicity* (see Theorem 3.2 of (Reiter, 1980)). Semi-monotonicity in default logic means the following: Let $\Delta = (D, W)$ and $\Delta' = (D', W)$ be two default theories such that $D \subseteq D'$; then for every extension $\mathcal{E}$ of Δ there is an extension $\mathcal{E}'$ of Δ' such that $\mathcal{E} \subseteq \mathcal{E}'$.

A default theory may not have any extensions (an example is the theory $(\{\frac{:\beta}{\neg\beta}\}, \emptyset)$. Then, a default theory is called *coherent* if it has at least one extension, and incoherent otherwise. Normal default theories are always coherent. A coherent default theory $\Delta = (D, W)$ is called *inconsistent* if it has just one extension which is inconsistent. By Theorem 2.2 of (Reiter, 1980), the theory Δ is inconsistent iff W is inconsistent. The theories examined in this paper are always coherent and consistent, since only normal default theories (D, W) with W a consistent set of literals are taken into account.

The *entailment problem* for default theories is as follows: Given a default theory Δ and a propositional formula ϕ, does every extension of Δ contain ϕ? In the affirmative case, we write $\Delta \models \phi$. For a set of propositional formulas S, we analogously write $\Delta \models S$ to denote $(\forall \phi \in S)(\Delta \models \phi)$.

2.3 Outliers in Default Logic

The issue of outlier detection in default theories is extensively discussed in (Angiulli et al., 2008). The formal definition of outlier there proposed is given as follows. For a given set W and a list of sets $S_1, \ldots, S_n$, $W_{S_1,\ldots,S_n}$ denotes the set $W \setminus (S_1 \cup S_2 \cup \ldots \cup S_n)$.

Definition 2.2 (Outlier and Outlier Witness Set) (Angiulli et al., 2008) Let $\Delta = (D, W)$ be a propositional default theory and let $L \subseteq W$ be a set of literals. If there exists a non-empty subset S of W_L such that:

1. $(D, W_S) \models \neg S$, and
2. $(D, W_{S,L}) \not\models \neg S$

then L is an *outlier* set in Δ and S is an *outlier witness* set for L in Δ.

The intuitive explanation of the different roles played by an outlier and its witness is as follows. Condition (i) of Definition 2.2 states that the outlier witness set S denotes something that does not agree with the knowledge encoded in the defaults. Indeed, by removing S from the theory at hand, we obtain $\neg S$. In other words, if S had not been observed, then, according to the given defaults, we would have concluded the exact opposite. Moreover, condition (ii) of Definition 2.2 states that the outlier L is a set of literals that, when removed from the theory, makes such a disagreement disappear. Indeed, by removing both S and L from the theory, $\neg S$ is no longer obtained. In other words, disagreement for S is a consequence of the presence of L in the theory. To summarize, the set S witnesses that the piece of knowledge denoted by L behaves, in a sense, exceptionally, tells us that L is an outlier set and S is its associated outlier witness set.

The intuition here is better illustrated by referring to the example on stolen credit card numbers given in the Introduction. A default theory $\Delta = (D, W)$ that describes such an episode might be as follows:

– $D = \left\{ \frac{CreditNumber : \neg MultipleIPs}{\neg MultipleIPs} \right\}$,

– $W = \{CreditNumber, MultipleIPs\}$.

Here, the credit card number might be stolen, for otherwise it wouldn't have been used over different continents during the same day. Accordingly, $L = \{CreditNumber\}$ is an outlier set here, and $S = \{MultipleIPs\}$ is the associated witness set. This reasoning agrees with our intuition that an outlier is, in some sense, abnormal and that the corresponding witness testifies to it.

Note that sets of outliers and their corresponding witness sets are selected among those explicitly embodied in the given knowledge base. Hence, outlier detection using default reasoning is essentially a knowledge discovery technique. As such, it can be very useful, to give one example, when applied to information systems for crime prevention and homeland security, because the outlier detection technique can be exploited in order to highlight suspicious individuals and/or events. Several examples for the usefulness of this approach are given in (Angiulli et al., 2008) and in Section 3.3 below.

3 Charting the Tractability Frontier of Outlier Detection

Subsection 3.1 recalls two main outlier detection tasks and the related complexity results known so far; Subsection 3.2 introduces *acyclic theories* and an interesting restriction of the above defined concept of outlier, that we call *strong outliers*, and finally, Subsection 3.5 presents the plan of a set of results that will allow us to chart the tractability frontier in the context of propositional normal mixed unary default theories.

3.1 Outlier Detection Problems

The computational complexity of discovering outliers in default theories under various classes of default logics has been previously investigated in (Angiulli et al., 2008). In particular, the two main recognition tasks in outlier detection are the *Outlier Recognition* and the *Outlier-Witness Recognition* problems (also called $Outlier(L)$ and $Outlier(S)(L)$, respectively, in (Angiulli et al., 2008)), and are defined as follows:

- **Outlier Recognition Problem:** *Given a default theory $\Delta = (D, W)$ and a set of literals $L \subseteq W$, is L an outlier set in Δ?*
- **Outlier-Witness Recognition Problem:** *Given a default theory $\Delta = (D, W)$ and two sets of literals $L \subset W$ and $S \subseteq W_L$, is L an outlier set with witness set S in Δ?*

Table 1 summarizes previous complexity results, together with the results that constitute the contributions of the present work that will be detailed later in this section.

In particular, the complexity of the *Outlier Recognition* and the *Outlier-Witness Recognition* problems has been studied in (Angiulli et al., 2008) for general and disjunction-free (DF) default theories and in (Angiulli et al., 2010) for normal unary (NU) and dual normal unary (DNU) default theories. The results there pointed out that the general problem of recognizing an outlier set is always intractable (see Theorem 4.3 in (Angiulli et al., 2008) and Theorem 3.6 in (Angiulli et al., 2010)). As for recognizing an outlier together with its witness, this problem is intractable for general and disjunction-free default theories (see Theorem 4.6 in (Angiulli et al., 2008)), but can be solved in polynomial time if either NU or DNU default theories are considered. Regarding the latter result, it is interesting to note that, while for both NU and DNU default theories the entailment of a literal can be decided in polynomial time, deciding the entailment in DF default theories is intractable.

Motivated by the intractability of the general *Outlier Recognition* problem on all classes of default theories considered so far, in this paper we take some further steps in analyzing the complexity of outlier detection problems in default logics in order to try to chart the associated tractability frontier. To this end, in the next sections we consider further subsets of the classes already mentioned, referred to as *Acyclic Normal Unary* and *Acyclic Dual Normal Unary* theories, and a specific kind of outlier, which we will call *Strong Outliers*. The latter, loosely speaking are characterized by a stronger relationship with their witness set than in the general case. Then, in Subsection 3.5, the main results of our complexity analysis are overviewed.

3.2 Strong Outliers and Acyclic Theories

Next, the definitions of strong outlier set (Section 3.3) and of acyclic default theory (Section 3.4) are given.

3.3 Strong Outliers

Recall the definition of outlier set already provided in Section 2.3 (see Definition 2.2).

Conditions 1 and 2 of the Definition 2.2 of outlier of Subsection 2.3 can be rephrased as follows:

1. $(\forall \ell \in S)(D, W_S) \models \neg \ell$, and
2. $(\exists \ell \in S)(D, W_{S,L}) \not\models \neg \ell$.

In other words, condition 1 states that the negation of every literal $\ell \in S$ must be entailed by (D, W_S) while, according to condition 2, it is sufficient for just one literal $\ell \in S$ to exist whose negation is not entailed by $(D, W_{S,L})$. Hence, there is a sort of "asymmetry" between the two conditions, which is the direct consequence of the semantics of the entailment established for sets of literals.

It is clear that, at least from a purely syntactic point of view, the relationship between the outlier set and its

Problem	*Outlier Type*	*General Default*	DF *Default*	(D)NU *Default*	*Acyclic* (D)NU *Default*
Outlier Recognition	*General*	Σ^{P}_3-c Th.4.3*	Σ^{P}_2-c Th.4.3*	NP-c Th.3.6**	$\underline{\text{NP-c}}$ Th.3.9
	Strong	NP-hard Th.3.13		$\underline{\text{NP-c}}$ Th.3.13	$\underline{\text{P}}$ Th.3.14
Outlier-Witness Recognition	*General*	$\mathrm{D}^{\mathrm{P}}_2$-c Th.4.6*	D^{P}-c Th.4.6*	P Th.3.1**	P Th.3.1**
	Strong	NP-hard Th.3.4		P Th.3.3	P Th.3.3

Table 1: Complexity results for outlier detection (*=reported in (Angiulli et al., 2008), **=reported in (Angiulli et al., 2010)).

witness set can be strengthened by replacing the existential quantifier in Condition 2 with the universal one, thus breaking the aforementioned asymmetry between the two conditions and obtaining the following definition of *strong outlier* set.

Definition 3.1 (Strong Outlier) *Let $\Delta = (D, W)$ be a propositional default theory and let $L \subset W$ be a set of literals. If there exists a non-empty subset S of W_L such that:*

1. $(\forall \ell \in S)(D, W_S) \models \neg\ell$*, and*
2. $(\forall \ell \in S)(D, W_{S,L}) \not\models \neg\ell$

then L is a strong outlier *set in Δ and S is a* strong outlier witness *set for L in Δ.*

The following proposition is immediately proved:

Proposition 3.2 *If L is a strong outlier set then L is an outlier set.*

Proof: Straightforward. □

Note that, in general the vice versa of Proposition 3.2 does not hold.

We study next the impact of restricting attention to strong outliers on the computational complexity of outlier detection problems. Before doing that, we first discuss the significance of the knowledge associated with strong outliers.

We begin with an example that is an extension of the credit card scenario presented in the Introduction. Recall that a credit card number is suspected to be stolen since it was used from several IPs in different continents during the same day. The example we give now is related to violating normal behavioral patterns in using a cellular phone. Normally, almost all the numbers that people call are from their contacts list. In addition, for each cell phone user, there are hours of the day during which she normally does not use the phone. For example, most users would not use the phone during the night hours. Finally, for a typical cellphone user, there is a list of locations from which she normally calls. The knowledge described above can be summarized using the following defaults:

1. $\frac{CreditNumber:\neg MultipleIPs}{\neg MultipleIPs}$ – Normally, credit card numbers are not used in different continents during the same day;
2. $\frac{CellUse:MfC}{MfC}$ – (MfC stands for "mostly from contacts") - Normally numbers dialed from a cell phone are mostly from the contact list;
3. $\frac{CellUse:\neg QuietTime}{\neg QuietTime}$ – Normally people do not use the phone during their "quiet time" - e.g. late at night;
4. $\frac{CellUse:\neg NewLocation}{\neg NewLocation}$ – Normally cell phones are used in locations in which the device was used in the past.

Now, suppose that a pickpocket stole Michelle's cellphone and purse from her handbag. She came home late and didn't notice the theft till morning. While she was sleeping, the pickpocket could broadcast her credit card numbers through malicious servers over the Internet and use her cellphone to make expensive phone calls. A sophisticated crime prevention information system could automatically notice exceptional behaviors, and make the following observations:

QuietTime – calls are made from the device during abnormal hours;

¬MfC – It is not the case that most of the calls' destinations are from the phone's contact list;

NewLocation – The device is in a location where it hasn't been before;

MultipleIPs – The credit card number is used in different continents during the last day.

Let us now consider the default theory $\Delta = (D, W)$, where D is the set of Defaults 1-4 introduced above, and $W = \{CreditNumber$, $CellUse$, $\neg MfC$, $QuietTime$, $NewLocation$, $MultipleIPs\}$. According to the definition of outlier given in (Angiulli et al., 2008) (see Definition 2.2), we get that $L = \{CreditNumber\}$ is an outlier and $S = \{\neg MfC$, $NewLocation$, $QuietTime$, $MultipleIPs\}$ is a possible witness set for L. This last witness set is also a witness set for the outlier $\{CellUse\}$. However, although the observations MfC and $QuietTime$ are in the witness set of the outlier $\{CreditNumber\}$, they do not explain why

$\{CreditNumber\}$ is an outlier. Similarly, the observation $MultipleIPs$ is in the witness set of the outlier $\{CellUse\}$ but it does not explain why $\{CellUse\}$ is an outlier.

One might suggest that in order to improve the behavior demonstrated above, we should look for a *minimal* witness set. However, it seems to us counter intuitive to look for a minimal witness set. If we identify an outlier, we would like to have a *maximal* set of the observations that support our suspicion. In the example above, $\{\neg MfC\}$ is a minimal witness set for the outlier $\{CellUse\}$, but its superset $\{\neg MfC$, $NewLocation$, $QuietTime\}$, which is also a witness set for the outlier $\{CellUse\}$, provides more information.

The notion of *strong* outlier presented above seems to adequately capture, in such scenarios as that depicted in this example situation, the notion of outlier and its witness set. If we use the definition of strong outlier we get that $S = \{\neg MfC$, $NewLocation$, $QuietTime$, $MultipleIPs\}$ is *neither* a witness set for the outlier $\{CreditNumber\}$ nor a witness set for the outlier $\{CellUse\}$. A witness set for the outlier $\{CellUse\}$ is, instead, the set $\{\neg MfC$, $NewLocation$, $QuietTime\}$ or any of its nonempty subsets, while a witness set for the outlier $\{CreditNumber\}$ is the set $\{MultipleIPs\}$.

We now turn to the complexity issues. In order to mark the tractability landscape of the new strong outlier detection problem, we provide two results, the former one regarding the tractability of the outlier-witness recognition problem and the latter one pertaining to its intractability.

Theorem 3.3 *Strong Outlier-Witness Recognition on propositional* NU *default theories is in* P.

Proof: The proof is immediate since the statement follows from the definition of strong outlier set (Definition 3.1) and the fact that the entailment problem on propositional NU default theories is polynomial time solvable (as proved in (Kautz & Selman, 1991; Zohary, 2002)). □

As for the complexity of the *Strong Outlier-Witness Recognition* problem on propositional DF and general default theories, the following statement holds.

Theorem 3.4 *Strong Outlier-Witness Recognition on propositional* DF *default theories is NP-hard.*

Proof: The statement follows from the reduction employed in Theorem 4.6 of (Angiulli et al., 2008), where it is proved that given two DF default theories $\Delta_1 = (D_1, \emptyset)$ and $\Delta_2 = (D_2, \emptyset)$, and two letters s_1 and s_2, the problem q of deciding whether $((\Delta_1 \models s_1) \wedge (\Delta_2 \models s_2))$ is valid can be reduced to the outlier-witness problem; that is, to the problem of deciding whether $L = \{s_2\}$ is an outlier having witness set $S = \{\neg s_1\}$ in the theory $\Delta(q)$, where $\Delta(q) = (D(q), W(q))$ is the propositional DF default theory with $D(q) = \{\frac{s_2 \wedge \alpha : \beta}{\beta} \mid \frac{\alpha : \beta}{\beta} \in D_1\} \cup D_2$ and $W(q) = \{\neg s_1, s_2\}$. Since the former problem is NP-hard, it follows from the reduction that the latter problem is NP-hard as well.

In order to complete the proof, we note that a singleton witness set is always a strong witness set and, hence, the above reduction immediately applies to strong outliers as well. □

3.4 Acyclic NU and DNU theories

In this section, acyclic normal mixed unary default theories are defined. We begin by introducing the notions of atomic dependency graph and that of tightness of a NMU default theory.

Definition 3.5 (Atomic Dependency Graph) Let $\Delta = (D, W)$ be a NMU default theory. The *atomic dependency graph* (V, E) of Δ is a directed graph such that

- $V = \{l \mid l$ is a letter occurring in $\Delta\}$, and
- $E = \{(x, y) \mid$ letters x and y occur respectively in the prerequisite and the consequent of a default in $D\}$.

Definition 3.6 (A set influences a literal) *Let* $\Delta = (D, W)$ *be an NMU default theory. We say that a set of literals* S influences *a literal* l *in* Δ *if for some* $t \in S$ *there is a path from* $letter(t)$ *to* $letter(l)$ *in the atomic dependency graph of* Δ.

Definition 3.7 (Tightness of an NMU theory) The *tightness* c of an NMU default theory is the size c (in terms of number of atoms) of the largest **strongly connected component** (SCC) of its atomic dependency graph.

Intuitively, an acyclic NMU default theory is a theory whose degree of cyclicity is fixed, where its degree of cyclicity is measured by means of its tightness, as formalized in the following definition.

Definition 3.8 (Acyclic NMU theory) Given a fixed positive integer c, a NMU default theory is said to be (c-)*acyclic*, if its tightness is not greater than c.

Figure 1 in Section 2 highlights the containment relationship among DF, NMU, NU, DNU, and acyclic default theories.

For the sake of simplicity, in the following sections we refer to c-acyclic theories simply as acyclic theories.

3.5 Main Results

It is clear from the definition of outlier that tractable subsets for outlier detection problems necessarily have to be singled out by considering theories for which the entailment operator is tractable. Thus, with the aim of identifying tractable fragments for the outlier recognition problem, we have investigated its complexity on acyclic (dual) normal unary default theories. These theories form a strict subset of normal unary default theories already considered in (Angiulli et al., 2010) (other than being a subset of acyclic NMU theories), for which the entailment problem is indeed polynomially time solvable (proved in (Kautz & Selman, 1991; Zohary, 2002)) and for which the outlier recognition problem is known to be NP-complete (Th. 3.6 of (Angiulli et al., 2010)).

Unexpectedly, it turns out that recognizing general outliers is intractable even in this rather restricted class of default theories, as accounted for in the theorem whose statement is reported below.

Theorem 3.9 *Outlier Recognition for NU acyclic default theories is* NP*-complete.*

Note that the results for NU (DNU, resp.) theories immediately apply to DNU theories, since given an NU (DNU, resp.) theory Δ, the dual theory $\overline{\Delta}$ of Δ is obtained from Δ by replacing each literal ℓ in Δ with $\neg\ell$ is a DNU (NU, resp.) theory that has the same properties of its dual.

Unfortunately, this result confirms that detecting outliers even in default theories as structurally simple as acyclic NU and DNU ones remains inherently intractable. Therefore, in order to chart the tractability frontier for this problem, we looked into the case of strong outliers. To characterize the complexity of this problem, a technical lemma, called the *incremental lemma*, is needed. The Incremental Lemma provides an interesting *monotonicity characterization in NMU theories* which is valuable on its own. The statement of the incremental lemma is reported next.

Lemma 3.10 *[The Incremental Lemma] Let (D, W) be an NMU default theory, q a literal and Sa set of literals such that $W \cup S$ is consistent and S does not influence q in (D, W). Then the following hold:*

Monotonicity of brave reasoning: If q is in some extension of (D, W) then q is in some extension of $(D, W \cup S)$.

Monotonicity of skeptical reasoning: If q is in every extension of (D, W) then q is in every extension of $(D, W \cup S)$.

This lemma helps us to state an upper bound on the size of any minimal outlier witness set in an acyclic NMU (and, hence, also NU and DNU) default theory.

Lemma 3.11 *Let (D, W) be a consistent NMU default theory and let L be a set of literals in W. If S is a minimal strong outlier witness set for L in (D, W), then $letter(S)$ is a subset of a SCC in the atomic dependency graph of (D, W).*

Taken together, the following tractability result can be proved.

Theorem 3.12 *Strong Outlier Recognition for NU acyclic default theories is in* P*.*

The proof is informally as follows. Since by Lemma 3.11 the size of a minimal strong outlier witness set is upper bounded by c, where c is the tightness of the theory, then the number of potential witness sets is polynomially bounded in the size of the theory. Moreover, checking conditions of Definition 2.2 can be done in polynomial time on NU theories, as the associated entailment is tractable. Based on these properties, a polynomial time procedure can be built that enumerates all the potential witness sets S for the outlier L and checks that S is actually a witness set for L.

The formal proofs of Theorem 3.14 and of Lemmas 3.10 and 3.11 are reported in the full paper (See (Angiulli et al.,). It is important to note that Lemma 3.11 cannot actually be exploited to prove the tractability of the *Strong Outlier Recognition* for NMU theories since for these theories the entailment problem remains intractable. Indeed, we show in the full paper that deciding the entailment is co-NP-complete even for NMU theories with tightness one.

This latter result is complemented by the following one.

Theorem 3.13 *Strong Outlier Recognition for NU cyclic default theories is* NP*-complete.*

In particular, both Theorems 3.9 and 3.13 make use of a lemma that informally speaking, establishes that, despite the difficulty to encode the conjunction of a set of literals using a NU theory, a *CNF formula can nonetheless be evaluated by means of condition* 1 *of Definition 2.2 applied to an acyclic NU theory, provided that the size of S is polynomial in the number of conjuncts in the formula.*

The tractability results complements the intractability results since Lemma 3.11 establishes that the size of a minimal strong outlier witness set is upper bounded by the tightness of the NU theory.

Recognizing strong outliers under acyclic (dual) normal default theories is the only outlier recognition problem known so far to be tractable; Furthermore, this result is indeed sharp, since switching either to general outliers, to cyclic theories, or to acyclic NMU theories makes the problem intractable.

Based on the above results, we designed a *polynomial time algorithm for enumerating all strong outliers of bounded size in an acyclic (dual) normal unary default theory.* The algorithm can also be employed to enumerate all strong outliers of bounded size in a general NMU theory and, with some minor modifications, all the general outliers and witness pairs of bounded size. However, in this latter case, since the problems at hand are NP-hard, its worst case running time will remain exponential, even if from a practical point of view it can benefit from some structural optimizations, based on Lemmas 3.10 and 3.11, which would allow it to reduce the size of the search space.

All complexity results presented inthis work, together with those already presented in the literature, are summarized in Table 1, where the problems lying on the tractability frontier are underlined.

Theorem 3.14 *Strong Outlier Recognition for NU acyclic default theories is in* P*.*

Proof: Given a NU default theory (D, W) of tightness c and a set of literals L from W, by Lemma 3.11 a minimal outlier witness set S for L in (D, W) has a size of at most

```
**Input:** $\Delta = (D, W)$ – a NU default theory.
**Output:** *Out* – the set of all strong outlier sets $L$ in $\Delta$ s.t. $|L| \leq k$.

let $C_1, \dots, C_N$ the ordered SCCs in the atomic dependency graph of $\Delta$;
set $Out$ to $\emptyset$;
for $i = 1..N$ do
  for all $S \subset W$ s.t. $letter(S) \subseteq C_i$ do
    if $(\forall \ell \in S)(D, W_S) \models \neg\ell$ then
      for all $L \subseteq W_S$ s.t. $|L| \leq k$ and $letter(S) \subseteq (C_1 \cup \ldots \cup C_i)$ do
        if $(\forall \ell \in S)(D, W_{S,L}) \not\models \neg\ell$ then
          set $Out$ to $Out \cup \{L\}$;
        end if
      end for
    end if
  end for
end for
```

Figure 2: Algorithm *Outlier Enumeration.*

c, where c is the maximum size of an SCC in the atomic dependency graph of (D, W).

Thus, the strong outlier recognition problem can be decided by solving the strong outlier-witness recognition problem for each subset S of literals in W_L having a size of at most c. Since the latter problem is polynomial time solvable (by Theorem 3.3) and since the number of times it has to be evaluated, that is $O(|W|^c)$, is polynomially in the size of the input, then the depicted procedure solves the strong outlier recognition problem in polynomial time. □

To take a step further and present an outlier enumeration algorithm, a known proposition is recalled.

Proposition 3.15 (proved in (Kautz & Selman, 1991; Zohary, 2002)) *Let Δ be an NU or a DNU propositional default theory and let L be a set of literals. Deciding whether $\Delta \models L$ is $\mathcal{O}(n^2)$, where n is the size of the theory Δ.*

Based on the above properties, we are now ready to describe the algorithm *Outlier Enumeration* which, for a fixed integer k, enumerates in polynomial time all the strong outlier sets of size at most k in an acyclic NU default theory.

The algorithm is presented in Figure 2. The SCCs $C_1, \dots, C_N$ of the atomic dependency graph of the theory are ordered such that there do not exist C_i and C_j with $i < j$ and two letters $l \in C_i$ and $q \in C_j$ such that there exists a path from $letter(q)$ to $letter(j)$.

By Theorem 3.15, the cost of steps 5 and 7 is $O(n^2)$. Thus, the cost of the algorithm is $O(2^c(n/c) \cdot (cn^2 + n^k cn^2)) = O(2^c n^{k+3})$. Since c and k are fixed, the algorithm enumerates the strong outliers in polynomial time in the size of (D, W). For example, all the singleton strong outlier sets can be enumerated in time $O(n^4)$.

4 Discussion and Conclusions

In this paper we have analyzed the tractability border associated with outlier detection in default logics. From Theorems 3.9 and 3.13, it is clear that neither acyclicity nor strongness alone are sufficient to achieve tractability. However, if both constraints are imposed together, the complexity of the outlier recognition problem falls below the tractability frontier, as shown in Theorem 3.14.

Overall, the results and arguments reported in this paper indicate that outlier recognition, even in its strong version, remains challenging and difficult on default theories. The tractability results we have provided nonetheless indicate that there are significant cases which can be efficiently implemented. A complete package for performing outlier detection in general default theories might therefore try to attain reasonable efficiency by recognizing such tractable fragments. Techniques by which the outlier detection task in default logics can be rendered practically affordable remain a major subject area for future research.

References

Angiulli, F., Zohary, R. B.-E., & Palopoli, L. Tractale strong outlier identification, submitted..

Angiulli, F., Zohary, R. B.-E., & Palopoli, L. (2008). Outlier detection using default reasoning. *Artificial Intelligence*, *172*(16-17), 1837–1872.

Angiulli, F., Zohary, R. B.-E., & Palopoli, L. (2010). Outlier detection for simple default theories. *Artificial Intelligence*, *174*(15), 1247–1253.

Chandola, V., Banerjee, A., & Kumar, V. (2009). Anomaly detection: A survey. *ACM Comput. Surv.*, *41*(3).

Hawkins, D. (1980). *Identification of Outliers.* Chapman and Hall, London, New York.

Hodge, V. J., & Austin, J. (2004). A survey of outlier detection methodologies. *Artif. Intell. Rev.*, *22*(2), 85–126.

Kautz, H. A., & Selman, B. (1991). Hard problems for simple default logics. *Artificial Intelligence*, *49*(1-3), 243–279.

Reiter, R. (1980). A logic for default reasoning. *Artificial Intelligence*, *13*(1-2), 81–132.

Zhang, A., & Marek, W. (1990). On the classification and existence of structures in default logic. *Fundamenta Informaticae*, *13*(4), 485–499.

Zohary, R. B.-E. (2002). Yet some more complexity results for default logic. *Artificial Intelligence*, *139*(1), 1–20.

Topics in Horn Contraction: Supplementary Postulates, Package Contraction, and Forgetting

James P. Delgrande
School of Computing Science,
Simon Fraser University,
Burnaby, B.C.,
Canada V5A 1S6.
jim@cs.sfu.ca

Renata Wassermann
Department of Computer Science
University of São Paulo
05508-090 São Paulo,
Brazil
renata@ime.usp.br

Abstract

In recent years there has been interest in studying belief change, specifically contraction, in Horn knowledge bases. Such work is arguably interesting since Horn clauses have found widespread use in AI; as well, since Horn reasoning is weaker than classical reasoning, this work also sheds light on the foundations of belief change. In this paper, we continue our previous work along this line. Our earlier work focussed on defining contraction in terms of *weak remainder sets*, or maximal subsets of an agent's belief set that fail to imply a given formula. In this paper, we first examine issues regarding the *extended contraction postulates* with respect to Horn contraction. Second, we examine *package contraction*, or contraction by a set of formulas. Last, we consider the closely-related notion of *forgetting* in Horn clauses. This paper then serves to address remaining major issues concerning Horn contraction based on remainder sets.

1 Introduction

Belief change addresses how a rational agent may alter its beliefs in the presence of new information. The best-known approach in this area is the AGM paradigm [Alchourrón *et al.*, 1985; Gärdenfors, 1988], named after the original developers. This work focussed on belief *contraction*, in which an agent may reduce its stock of beliefs, and belief *revision*, in which new information is consistently incorporated into its belief corpus. In this paper we continue work in belief contraction in the expressively weaker language of *Horn formulas*, where a Horn formula is a conjunction of Horn clauses and a Horn clause can be written as a rule in the form $a_1 \wedge a_2 \wedge \cdots \wedge a_n \rightarrow a$ for $n \geq 0$, and where a, a_i $(1 \leq i \leq n)$ are atoms. (Thus, expressed in conjunctive normal form, a Horn clause will have at most one positive literal.) Horn contraction has been addressed previously in [Delgrande, 2008; Booth *et al.*, 2009; Delgrande and Wassermann, 2010; Zhuang and Pagnucco, 2010b]. With the exception of the last reference, this work centres on the notion of a *remainder set*, or maximal subset of a knowledge base that fails to imply a given formula.

In this paper we continue work in Horn belief contraction, on a number of aspects; our goal is to essentially complete the overall framework of Horn contraction based on remainder sets. Previous work in this area has addressed counterparts to the *basic* AGM postulates; consequently we first examine prospects for extending the approach to counterparts of the *supplemental* AGM postulates. Second, we address *package contraction*, in which one may contract by a set of formulas, and the result is that no (contingent) formula in the set is believed. In the AGM approach, for a finite number of formulas this can be accomplished by contracting by the disjunction of the formulas. Since the disjunction of Horn formulas may not be in Horn form, package contraction then becomes an important accessory operation. Last we briefly examine a *forgetting* operator, in which one effectively reduces the language of discourse.

The next section introduces belief change while the third section discusses Horn clause reasoning, and previous work in the area. Section 4 examines the supplementary postulates; Section 5 addresses package contraction; and Section 6 covers forgetting. The last section contains a brief conclusion.

2 The AGM Framework for Contraction

As mentioned, the AGM approach [Alchourrón *et al.*, 1985; Gärdenfors, 1988] is the best-known approach to belief change. Belief states are modelled by deductively-closed sets of sentences, called *belief sets*, where the underlying logic includes classical propositional logic. Thus a belief set K satisfies the constraint:

$$\text{If } K \text{ logically entails } \phi \text{ then } \phi \in K.$$

The most basic operator is called *expansion*: For belief set K and formula ϕ, the expansion of K by ϕ, $K + \phi$, is the deductive closure of $K \cup \{\phi\}$. Of more interest are *contraction*, in which an agent reduces its set of beliefs, and *revision*, in which an agent consistently incorporates a new belief. These operators can be characterised by two means. First, a set of *rationality postulates* for a belief change function may be provided; these postulates stipulate constraints that should govern any rational belief change function. Second, specific constructions for a belief change function are given. *Representation results* can then be given (or at least are highly desirable) showing that a set of rationality postulates exactly captures the operator given by a particular construction.

Our focus in this paper is on belief contraction, and so we review these notions with respect to this operator. Informally, the contraction of a belief set by a formula is a belief set in which that formula is not believed. Formally, a contraction function $\dot{-}$ is a function from $2^{\mathcal{L}} \times \mathcal{L}$ to $2^{\mathcal{L}}$ satisfying the following postulates:

($K\dot{-}1$) $K\dot{-}\phi$ is a belief set.

($K\dot{-}2$) $K\dot{-}\phi \subseteq K$.

($K\dot{-}3$) If $\phi \not\in K$, then $K\dot{-}\phi = K$.

($K\dot{-}4$) If not $\vdash \phi$, then $\phi \not\in K\dot{-}\phi$.

($K\dot{-}5$) If $\phi \in K$, then $K \subseteq (K\dot{-}\phi) + \phi$.

($K\dot{-}6$) If $\vdash \phi \equiv \psi$, then $K\dot{-}\phi = K\dot{-}\psi$.

($K\dot{-}7$) $K\dot{-}\phi \cap K\dot{-}\psi \subseteq K\dot{-}(\phi \wedge \psi)$.

($K\dot{-}8$) If $\psi \not\in K\dot{-}(\psi \wedge \phi)$ then $K\dot{-}(\phi \wedge \psi) \subseteq K\dot{-}\psi$.

The first six postulates are called the *basic* contraction postulates, while the last two are referred to as the *supplementary* postulates. We have the following informal interpretations of the postulates: contraction yields a belief set ($K\dot{-}1$) in which the sentence for contraction ϕ is not believed (unless ϕ is a tautology) ($K\dot{-}4$). No new sentences are believed ($K\dot{-}2$), and if the formula is not originally believed then contraction has no effect ($K\dot{-}3$). The fifth postulate, the so-called *recovery* postulate, states that nothing is lost if one contracts and expands by the same sentence. This postulate is controversial; see for example [Hansson, 1999]. The sixth postulate asserts that contraction is independent of how a sentence is expressed. The last two postulates express relations between contracting by conjunctions and contracting by the constituent conjuncts. ($K\dot{-}7$) says that if a formula is in the result of contracting by each of two formulas then it is in the result of contracting by their conjunction. ($K\dot{-}8$) says that if a conjunct is not in the result of contracting by a conjunction, then contracting by that conjunct is (using ($K\dot{-}7$)) the same as contracting by the conjunction.

Several constructions have been proposed to characterise belief change. The original construction was in terms of *remainder sets*, where a remainder set of K with respect to ϕ is a maximal subset of K that fails to imply ϕ. Formally:

Definition 1 *Let $K \subseteq \mathcal{L}$ and let $\phi \in \mathcal{L}$.*

$K\downarrow\phi$ is the set of sets of formulas s.t. $K' \in K\downarrow\phi$ iff

1. *$K' \subseteq K$*
2. *$K' \not\vdash \phi$*
3. *For any K'' s.t. $K' \subset K'' \subseteq K$, it holds that $K'' \vdash \phi$.*

$X \in K\downarrow\phi$ is a remainder set of K wrt ϕ.

From a logical point of view, the remainder sets comprise equally-good candidates for a contraction function. *Selection functions* are introduced to reflect the extra-logical factors that need to be taken into account, to obtain the "best" or most plausible remainder sets. In *maxichoice contraction*, the selection function determines a single selected remainder set as the contraction. In partial meet contraction, the selection function returns a subset of the remainder sets, the intersection of which constitutes the contraction. Thus if the selection function is denoted by $\gamma(\cdot)$, then the contraction of K by formula ϕ can be expressed by

$$K\dot{-}\phi \;=\; \bigcap \gamma(K\downarrow\phi).$$

For arbitrary theory K and function $\dot{-}$ from $2^{\mathcal{L}} \times \mathcal{L}$ to $2^{\mathcal{L}}$, it proves to be the case that $\dot{-}$ is a partial meet contraction function iff it satisfies the basic contraction postulates ($K\dot{-}1$)–($K\dot{-}6$). Last, let $\preceq$ be a transitive relation on 2^K, and let the selection function be defined by:

$$\gamma(K\downarrow\phi) \;=\; \{K' \in K\downarrow\phi \mid \forall K'' \in K\downarrow\phi, K'' \preceq K'\}.$$

γ is a *transitively relational* selection function, and $\dot{-}$ defined in terms of such a γ is a *transitively relational partial meet contraction function*. Then we have:

Theorem 1 ([Alchourrón *et al.*, 1985]) *Let K be a belief set and let $\dot{-}$ be a function from $2^{\mathcal{L}} \times \mathcal{L}$ to $2^{\mathcal{L}}$. Then*

1. *$\dot{-}$ is a partial meet contraction function iff it satisfies the contraction postulates ($K\dot{-}1$)–($K\dot{-}6$).*
2. *$\dot{-}$ is a transitively relational partial meet contraction function iff it satisfies the contraction postulates ($K\dot{-}1$)–($K\dot{-}8$).*

The second major construction for contraction functions is called *epistemic entrenchment*. The general idea is that extralogic factors related to contraction are given by an ordering on formulas in the agent's belief set, reflecting how willing the agent would be to give up a formula. Then a contraction function can be defined in terms of removing less entrenched formulas from the belief set. It is shown in [Gärdenfors and Makinson, 1988] that for logics including classical propositional logic, the two types of constructions, selection functions over remainder sets and epistemic entrenchment orderings, capture the same class of contraction functions; see also [Gärdenfors, 1988] for details.

3 Horn Theories and Horn Contraction

3.1 Preliminary Considerations

Let $\mathbf{P} = \{a, b, c, \dots\}$ be a finite set of atoms, or propositional letters, that includes the distinguished atom $\bot$. $\mathcal{L}_H$ is the language of *Horn formulas*. That is, $\mathcal{L}_H$ is given by:

1. Every $p \in \mathbf{P}$ is a Horn clause.
2. $a_1 \wedge a_2 \wedge \cdots \wedge a_n \rightarrow a$, where $n \geq 0$, and a, a_i $(1 \leq i \leq n)$ are atoms, is a Horn clause.
3. Every Horn clause is a Horn formula.
4. If ϕ and ψ are Horn formulas then so is $\phi \wedge \psi$.

For a rule r as in 2 above, $head(r)$ is a, and $body(r)$ is the set $\{a_1, a_2, \dots, a_n\}$. Allowing conjunctions of rules, as given in 4, adds nothing of interest to the expressivity of the language with respect to reasoning. However, it adds to the expressibility of contraction, as we are able to contract by more than a single Horn clause. For convenience, we use $\top$ to stand for some arbitrary tautology.

An *interpretation* of $\mathcal{L}_H$ is a function from $\mathbf{P}$ to $\{true, false\}$ such that $\bot$ is assigned $false$. Sentences of $\mathcal{L}_H$ are *true* or *false* in an interpretation according to the standard rules in propositional logic. An interpretation M is a *model* of a sentence ϕ (or set of sentences), written $M \models \phi$, just if M makes ϕ true. $Mod(\phi)$ is the set of models of formula (or set of formulas) ϕ; thus $Mod(\top)$ is the set of interpretations of $\mathcal{L}_H$. An interpretation is usually identified with the atoms true in that interpretation. Thus, for $\mathbf{P} = \{p, q, r, s\}$ the interpretation $\{p, q\}$ is that in which p and q are true and r and s are false. For convenience, we also express interpretations by juxtaposition of atoms. Thus the interpretations $\{\{p, q\}, \{p\}, \{\}\}$ will usually be written as $\{pq, p, \emptyset\}$.

A key point is that Horn theories are characterised semantically by the fact that the models of a Horn theory are closed under intersections of positive atoms in an interpretation. That is, Horn theories satisfy the constraint:

$$\text{If } M_1, M_2 \in Mod(H) \text{ then } M_1 \cap M_2 \in Mod(H).$$

This leads to the notion of the *characteristic models* [Khardon, 1995] of a Horn theory: M is a characteristic model of theory H just if for every $M_1, M_2 \in Mod(H)$, $M_1 \cap M_2 = M$ implies that $M = M_1$ or $M = M_2$. E.g. the theory expressed by $\{p \land q \to \bot, r\})$ has models $\{pr, qr, r\}$ and characteristic models $\{pr, qr\}$. Since $pr \cap qr = r$, r isn't a characteristic model of H.

A Horn formula ψ is entailed by a set of Horn formulas A, $A \vdash_H \psi$, just if any model of A is also a model of ψ. For simplicity, and because we work exclusively with Horn formulas, we drop the subscript and write $A \vdash \psi$. If $A = \{\phi\}$ is a singleton set then we just write $\phi \vdash \psi$. A set of formulas A is *inconsistent* just if $A \vdash \bot$. We use $\phi \leftrightarrow \psi$ to represent logical equivalence, that is $\phi \vdash \psi$ and $\psi \vdash \phi$.

Notation: We collect here notation that is used in the paper. Lower-case Greek characters ϕ, ψ, $\ldots$, possibly subscripted, denote arbitrary formulas of $\mathcal{L}_H$. Upper case Roman characters A, B, $\ldots$, possibly subscripted, denote arbitrary sets of formulas. H (H_1, H', etc.) denotes Horn belief sets, so that $\phi \in H$ iff $H \vdash_H \phi$.

$Cn^h(A)$ is the deductive closure of a Horn formula or set of formulas A under Horn derivability. $|\phi|$ is the set of maximal, consistent Horn theories that contain ϕ. m (and subscripted variants) represents a maximum consistent set of Horn formulas.

M (M_1, M', etc.) denote interpretations over some fixed language. $Mod(A)$ is the set of models of A. Arbitrary sets of interpretations will be denoted $\mathcal{M}$ ($\mathcal{M}'$ etc.). $Cl_\cap(\mathcal{M})$ is the intersection closure of a set of interpretations $\mathcal{M}$;[1] that is, $Cl_\cap(\mathcal{M})$ is the least set such that $\mathcal{M} \subseteq Cl_\cap(\mathcal{M})$ and M_1, $M_2 \in Cl_\cap(\mathcal{M})$ implies that $M_1 \cap M_2 \in Cl_\cap(\mathcal{M})$. Note that M denotes an interpretation expressed as a set of atoms, while m denotes a maximum consistent set of Horn formulas. Thus the logical content is the same, in that an interpretation defines a maximum consistent set of Horn formulas, and vice versa. We retain these two interdefinable notations, since each is useful in the subsequent development. Similar comments apply to $Mod(\phi)$ vs. $|\phi|$.

Since **P** is finite, a (Horn or propositional logic) belief set may be finitely represented, that is, for X a belief set, there is a formula ϕ such that $Cn^h(\phi) = X$. As well, we make use of the fact that there is a 1-1 correspondence between elements of $|\phi|$ and of $Mod(\phi)$.

counter-model	induced models	resulting KB	r.s.
a		$a \land (c \to b)$	$\surd$
ac	a	a	
b		$b \land (c \to a)$	$\surd$
bc	b	b	
$\emptyset$		$(a \to b) \land (b \to a) \land (c \to a \land b)$	$\surd$
c	$\emptyset$	$(a \to b) \land (b \to a)$	

Figure 1: Example: Candidates for Horn contraction

3.2 Horn Contraction

The last few years have seen work on Horn contraction. Delgrande [2008] addressed *maxichoice Horn belief set contraction* based on (Horn) remainder sets, called *e-remainder sets*. The definition of *e-remainder sets* for Horn clause belief sets is the same as that for a remainder set (Definition 1) but with respect to Horn clauses and Horn derivability. For H a Horn belief set and $\phi \in \mathcal{L}_H$, the set of *e-remainder sets* with respect to H and ϕ is denoted by $H \downarrow_e \phi$.

Booth, Meyer, and Varzinczak [2009] subsequently investigated this area by considering partial meet contraction, as well as a generalisation of partial-meet, based on the idea of *infra-remainder sets* and package contraction. In [Booth *et al.*, 2009], an *infra remainder sets* is defined as follows:

Definition 2 *For belief sets H and X, $X \in H \Downarrow_e \phi$ iff there is some $X' \in H \downarrow_e \phi$ such that $(\bigcap H \downarrow_e \phi) \subseteq X \subseteq X'$. The elements of $H \Downarrow_e \phi$ are the* infra e-remainder sets *of H with respect to ϕ.*

All e-remainder sets are infra e-remainder sets, as is the intersection of any set of e-remainder sets. It proved to be the case that e-remainder sets (and including the infra-remainder sets of [Booth *et al.*, 2009]) are not sufficiently expressive for contraction.

The problem arises from the relation between remainder sets on the one hand, and their counterpart in terms of interpretations on the other. In the classical AGM approach, a remainder set is characterised semantically by a minimal superset of the models of the agent's belief set such that this superset does not entail the formula for contraction. As a result, the models of a remainder set consist of the models of a belief set H together with a countermodel of the formula ϕ for contraction. With Horn clauses, things are not quite so simple, in that for a countermodel M of ϕ, there may be no Horn remainder set that has M as a model.

To see this, consider the following example, adapted from [Delgrande and Wassermann, 2010].

Example 1 *Let $\mathbf{P} = \{a, b, c\}$ and $H = Cn^h(a \land b)$. Consider candidates for $H \dot{-} (a \land b)$. There are three remainder sets, given by the Horn closures of $a \land (c \to b)$, $b \land (c \to a)$, and $(a \to b) \land (b \to a) \land (c \to a \land b)$). Any infra-remainder set contains the closure of $(c \to a) \land (c \to b)$.*

See Figure 1. In the first line of the table, we have that a (viz. $\{a, \neg b, \neg c\}$) is a countermodel of $a \land b$. Adding this model to the models of H yields the models of the formula $a \land (c \to b)$. This characterises a remainder set, as indicated in the last column. In the second line, we have that ac (viz.

[1] Recall that an interpretation is represented by the set of atoms true in the interpretation.

$\{a, \neg b, c\}$) is another countermodel of H. However, since H has a model ab, the intersection of these models, $ab \cap ac = a$ must also be included; this is the item in the second column. The resulting belief set is characterised by the interpretations $Mod(H) \cup \{ac, a\} = \{abc, ab, ac, a\}$, which is the set of models of formula a, as given in the third column. However, the result isn't a remainder set, since $Cn^h(a \wedge (c \rightarrow b))$ is a logically stronger belief set than $Cn^h(a)$, which also fails to imply $a \wedge b$.

This result is problematic for both [Delgrande, 2008] and [Booth *et al.*, 2009]. For example, in none of the approaches in these papers is it possible to obtain $H \dot{-}_e (a \wedge b) \leftrightarrow a$, nor $H \dot{-}_e (a \wedge b) \leftrightarrow (a \equiv b)$. But presumably these possibilities are desirable as *potential* contractions. Thus, in all of the approaches developed in the cited papers, it is not possible to have a contraction wherein $a \wedge \neg b \wedge c$ corresponds to a model of the contraction.

This issue was addressed in [Delgrande and Wassermann, 2010]. There the *characteristic models* of maxichoice candidates for $H \dot{-}_e \phi$ consist of the *characteristic models* of H together with a single interpretation from $Mod(\top) \setminus Mod(\phi)$. The resulting theories, called *weak remainder sets*, corresponded to the theories given in the third column in Figure 1.

Definition 3 ([Delgrande and Wassermann, 2010]) *Let H be a Horn belief set, and let ϕ be a Horn formula.*

$H \Downarrow_e \phi$ is the set of sets of formulas s.t. $H' \in H \Downarrow_e \phi$ iff $H' = H \cap m$ for some $m \in |\top| \setminus |\phi|$.

$H' \in H \Downarrow_e \phi$ is a weak remainder set *of H and ϕ.*

The following characterizations were given for maxichoice and partial meet Horn contraction:

Theorem 2 ([Delgrande and Wassermann, 2010]) *Let H be a Horn belief set. Then $\dot{-}_w$ is an operator of maxichoice Horn contraction based on weak remainders iff $\dot{-}_w$ satisfies the following postulates.*

($H \dot{-}_w 1$) *$H \dot{-}_w \phi$ is a belief set.* *(closure)*

($H \dot{-}_w 2$) *If not $\vdash \phi$, then $\phi \notin H \dot{-}_w \phi$.* *(success)*

($H \dot{-}_w 3$) *$H \dot{-}_w \phi \subseteq H$.* *(inclusion)*

($H \dot{-}_w 4$) *If $\phi \notin H$, then $H \dot{-}_w \phi = H$.* *(vacuity)*

($H \dot{-}_w 5$) *If $\vdash \phi$ then $H \dot{-}_w \phi = H$* *(failure)*

($H \dot{-}_w 6$) *If $\phi \leftrightarrow \psi$, then $H \dot{-}_w \phi = H \dot{-}_w \psi$.* *(extensionality)*

($H \dot{-}_w 7$) *If $H \neq H \dot{-}_w \phi$ then $\exists \beta \in \mathcal{L}_H$ s.t. $\{\phi, \beta\}$ is inconsistent, $H \dot{-}_w \phi \subseteq Cn^h(\{\beta\})$ and $\forall H'$ s.t $H \dot{-}_w \phi \subset H' \subseteq H$ we have $H' \not\subseteq Cn^h(\{\beta\})$.* *(maximality)*

Theorem 3 ([Delgrande and Wassermann, 2010]) *Let H be a Horn belief set. Then $\dot{-}_w$ is an operator of partial meet Horn contraction based on weak remainders iff $\dot{-}_w$ satisfies the postulates ($H \dot{-}_w 1$) – ($H \dot{-}_w 6$) and:*

($H \dot{-}_{pm} 7$) *If $\beta \in H \backslash (H - \alpha)$, then there is some H' such that $H - \alpha \subseteq H'$, $\alpha \notin Cn^h(H')$ and $\alpha \in Cn^h(H' \cup \{\beta\})$ (weak relevance)*

More recently, [Zhuang and Pagnucco, 2010b] have addressed Horn contraction from the point of view of epistemic entrenchment. They compare AGM contraction via epistemic entrenchment in classical propositional logic with contraction in Horn logics. A postulate set is provided and shown to characterise entrenchment-based Horn contraction. The fact that AGM contraction refers to disjunctions of formulas, which in general will not be Horn, is handled by considering *Horn strengthenings* in their postulate set, which is to say, logically weakest Horn formulas that subsume the disjunction. In contrast to earlier work, their postulate set includes equivalents to the supplemental postulates, and so goes beyond the set of basic postulates.

For a given clause φ, the set of its Horn strengthenings $(\varphi)_H$ is the set such that $\psi \in (\varphi)_H$ if and only if ψ is a Horn clause and there is no Horn clause ψ' such that $\psi \subset \psi' \subseteq \varphi$.

Of the set of ten postulates given in [Zhuang and Pagnucco, 2010b], five correspond to postulates characterizing partial meet contraction based on weak remainders as defined in [Delgrande and Wassermann, 2010] and two correspond to the supplementary postulates $(K \dot{-} 7)$ and $(K \dot{-} 8)$. The three new postulates are:

$(H \dot{-} 5)$ If $\psi \in H \dot{-} \varphi \wedge \psi$ then $\psi \in H \dot{-} \varphi \wedge \psi \wedge \delta$

$(H \dot{-} 9)$ If $\psi \in H \setminus H \dot{-} \varphi$ then $\forall \chi \in (\varphi \vee \psi)_H, \chi \notin H \dot{-} \varphi$

$(H \dot{-} 10)$ If $\forall \chi \in (\varphi \vee \psi)_H, \chi \notin H \dot{-} \varphi \wedge \psi$ then $\psi \notin H \setminus H \dot{-} \varphi$

While there has been other work on belief change and Horn logic, such work focussed on specific aspects of the problem, rather than a general characterisation of Horn clause belief change. For example, Eiter and Gottlob [1992] address the complexity of specific approaches to revising knowledge bases, including the case where the knowledge base and formula for revision are conjunctions of Horn clauses. Not unexpectedly, results are generally better in the Horn case. Liberatore [2000] considers the problem of compact representation for revision in the Horn case. Basically, given a knowledge base K and formula ϕ, both Horn, the main problem addressed is whether the knowledge base, revised according to a given operator, can be expressed by a propositional formula whose size is polynomial with respect to the sizes of K and ϕ. [Langlois *et al.*, 2008] approaches the study of revising Horn formulas by characterising the existence of a complement of a Horn consequence; such a complement corresponds to the result of a contraction operator. This work may be seen as a specific instance of a general framework developed in [Flouris *et al.*, 2004]. In [Flouris *et al.*, 2004], belief change is studied under a broad notion of *logic*, where a logic is a set closed under a Tarskian consequence operator. In particular, they give a criterion for the existence of a contraction operator satisfying the basic AGM postulates in terms of *decomposability*.

4 Supplementary postulates

In this section we investigate how the different proposals for Horn contraction operations behave with respect to the supplementary postulates (K-7) and (K-8). Throughout the section, we consider all selection functions to be transitively relational.

First we consider the operation of Horn Partial Meet e-Contraction as defined in [Delgrande, 2008]. The following example shows that, considering $\downarrow_e$ as defined in [Delgrande, 2008], Horn Partial Meet e-Contraction does not satisfy ($K\dot{-}7$):

Example 2 *Let $H = Cn^h(\{a \to b, b \to c, a \to d, d \to c\})$. We then have*

$H \downarrow_e a \to c = \{H_1, H_2, H_3, H_4\}$
$H \downarrow_e b \to c = \{H_5\}$

where:
$H_1 = Cn^h(\{a \to b, a \to d\})$,
$H_2 = Cn^h(\{a \to b, a \wedge c \to d, d \to c\})$,
$H_3 = Cn^h(\{b \to c, a \wedge c \to b, a \to d\})$,
$H_4 = Cn^h(\{a \wedge c \to b, b \to c, a \wedge c \to d, d \to c, a \wedge d \to b, a \wedge b \to d\})$*, and*
$H_5 = Cn^h(\{a \to b, a \to d, d \to c\})$

Note that the two first elements of $H \downarrow_e a \to c$ are subsets of the single element of $H \downarrow_e b \to c$ and hence, cannot belong to $H \downarrow_e a \to c \wedge b \to c$.

$H \downarrow_e a \to c \wedge b \to c = \{H_3, H_4, H_5\}$

If we take a selection function based on a transitive relation between remainder sets that gives priority in the order in which they appear in this example, i.e., $H_5 \prec H_4 \prec H_3 \prec H_2 \prec H_1$, we will have:

$H - a \to c = H_1$
$H - b \to c = H_5$
$H - a \to c \wedge b \to c = H_3$

And we see that $H - a \to c \cap H - b \to c = H_1 \not\subseteq H_3 = H - a \to c \wedge b \to c$

The same example shows that the operation does not satisfy ($K\dot{-}8$):
$a \to c \notin H - a \to c \wedge b \to c$, but $H - a \to c \wedge b \to c \not\subseteq H - a \to c$.

If there are no further restrictions on the selection function, the same example also shows that contraction based on infra-remainders does not satisfy the supplementary postulates. Note that each remainder set in the example is also an infra-remainder and that the selection function always selects a single element. It suffices to assign all the remaining infra-remainders lower priority.

Now we can show that the operation of partial meet based on weak remainders (PMWR) has a better behaviour with respect to the supplementary postulates:

Proposition 1 *Partial meet based on weak remainders and a transitive relational selection function satisfies ($K\dot{-}7$) and ($K\dot{-}8$).*

We have seen that Epistemic Entrenchment Horn Contraction (EEHC) is characterized by a set of ten postulates. In [Zhuang and Pagnucco, 2010a], it is shown that transitively relational PMWR as defined above is more general than EEHC. This means that any operation satisfying their set of 10 postulates (which include ($K\dot{-}7$) and ($K\dot{-}8$)) is a PMWR. We have seen that PMWR satisfies ($K\dot{-}7$) and ($K\dot{-}8$), hence, in order to compare PMWR and EEHC, we need to know whether PMWR satisfies ($H\dot{-}5$), ($H\dot{-}9$) and ($H\dot{-}10$).

Proposition 2 *PMWR satisfies* ($H\dot{-}5$).

Proposition 3 *PMWR satisfies* ($H\dot{-}9$)

PMWR in general does not satisfy ($H\dot{-}10$), as the following example shows.
Let $H = Cn^h(\{a, b\})$. Then
$H \downdownarrows_e a = \{H_1, H_3\}$ and
$H \downdownarrows_e a \wedge b = \{H_1, H_2, H_3\}$, where
$H_1 = Cn^h(\{a \vee \neg b, b \vee \neg a\})$,
$H_2 = Cn^h(\{a\})$ and
$H_3 = Cn^h(\{b\})$.
Assuming a selection function based on a transitive relation such that $H_1 \prec H_2$ and $H_1 \prec H_3$ (and $H_2 \preceq H_3$ and $H_3 \preceq H_2$), we have
$H - a = H_3$ and $H - a \wedge b = H_2 \cap H_3$
Since $(a \vee b)_H = \{a, b\}$, we have that for any $\chi \in (a \vee b)_H$, $\chi \notin H - a \wedge b$, but $b \in H - a$.

In order to finish the comparison between the sets of postulates, it is interesting to note the following:

Observation 1 ($H\dot{-}9$) *implies weak relevance.*

5 Package Contraction

In this section we consider *Horn package contraction*. For belief set H and a set of formulas Φ, the package contraction $H \dot{-}_p \Phi$ is a form of contraction in which no member of Φ is in $H \dot{-}_p \Phi$. As [Booth *et al.*, 2009] points out, this operation is of interest in Horn clause theories given their limited expressivity: in order to contract by ϕ and ψ simultaneously, one cannot contract by the disjunction $\phi \vee \psi$, since the disjunction is generally not a Horn clause. Hence, one expresses the contraction of both ϕ and ψ as the package contraction $H \dot{-}_p \{\phi, \psi\}$.

We define the notion of Horn package contraction, and show that it is in fact expressible in terms of maxichoice Horn contraction.

Definition 4 *Let H be a Horn belief set, and let $\Phi = \{\phi_1, \ldots, \phi_n\}$ be a set of Horn formulas.*
$H \downdownarrows_p \Phi$ is the set of sets of formulas s.t. $H' \in H \downdownarrows_p \Phi$ iff $\exists m_1, \ldots, m_n$ such that, for $1 \leq i \leq n$:
$m_i \in |\top| \setminus |\phi_i|$ if $\nvdash \phi_i$, otherwise $m_i = \mathcal{L}_H$
and $H' = H \cap \bigcap_{i=1}^n m_i$.

Definition 5 *Let γ be a selection function on H such that $\gamma(H \downdownarrows_p \Phi) = \{H'\}$ for some $H' \in H \downdownarrows_p \Phi$.*
The (maxichoice) package Horn contraction *based on weak remainders is given by:*

$$H \dot{-}_p \Phi = \gamma(H \downdownarrows_p \Phi)$$

if $\emptyset \neq \Phi \cap H \not\subseteq Cn^h(\top)$; and H otherwise.

The following result relates elements of $H \downdownarrows_p \Phi$ to weak remainders.

Proposition 4 *Let H be a Horn belief set and let $\Phi = \{\phi_1, \ldots, \phi_n\}$ be a set of Horn formulas where for $1 \leq i \leq n$ we have $\nvdash \phi_i$.*
Then $H' \in H \downdownarrows_p \Phi$ iff for $1 \leq i \leq n$ there are $H_i \in H \downdownarrows_e \phi_i$ and $H' = \bigcap_{i=1}^n H_i$.

It follows immediately from this that any maxichoice Horn contraction defines a package contraction, and vice versa.

Example 3 *Consider the Horn belief set $H = Cn^h(\{a, b\})$ over $\mathbf{P} = \{a, b, c\}$. We want to determine elements of*

$$H \Downarrow_p \Phi \;=\; Cn^h(\{a, b\}) \Downarrow_p \{a, b\}.$$

It proves to be the case that there are a total of 14 elements in $H \Downarrow_p \Phi$ and so 14 candidate package contractions. We have the following.

1. *There are 4 countermodels of a, given by:*

   $$A = \{bc, b, c, \emptyset\}.$$

   Thus there are four weak remainders corresponding to these countermodels, and so four candidates for maxichoice Horn contraction by a.

2. *Similarly there are 4 countermodels of b:*

   $$B = \{ac, a, c, \emptyset\}.$$

3. *Members of $H \Downarrow_p \Phi$ are given by*

   $$Cl_\cap(Mod(H) \cup \{x\} \cup \{y\})$$

   for $x \in A$ and $y \in B$.

For example, for $x = bc$, $y = \emptyset$, we have that $Cl_\cap(Mod(H) \cup \{x\} \cup \{y\}) = \{abc, ab, bc, b, \emptyset\}$, which is the set of models of $(c \to b) \wedge (a \to b)$.

For $x = bc$, $y = ac$, we have that $Cl_\cap(Mod(H) \cup \{x\} \cup \{y\}) = Cn^h(\top)$; this holds for no other choice of x and y.

What this example indicates informally is that there is a great deal of scope with respect to candidates for package contraction. To some extent, such a combinatorial explosion of possibilities is to be expected, given the fact that a formula will in general have a large number of countermodels, and that this is compounded by the fact that each formula in a package contraction does not hold in the result. However, it can also be noted that some candidate package contractions appear to be excessively weak; for example it would be quite drastic to have $Cn^h(\top)$ as the result of such a contraction. As well, some candidate package contractions appear to contain redundancies, in that a selected countermodel of a may also be a countermodel of b, in which case there seems to be no reason to allow the possible incorporation of a separate countermodel of b. Consequently, we also consider versions of package contraction that in some sense yield a maximal belief set. However, first we provide results regarding package contraction.

We have the following result:

Theorem 4 *Let H be a Horn belief set. Then if $\dot{-}_p$ is an operator of maxichoice Horn package contraction based on weak remainders then $\dot{-}_p$ satisfies the following postulates.*

($H\dot{-}_p 1$) $H\dot{-}_p \Phi$ *is a belief set.* *(closure)*

($H\dot{-}_p 2$) *For $\phi \in \Phi$, if not $\vdash \phi$, then $\phi \notin H\dot{-}_p \Phi$* *(success)*

($H\dot{-}_p 3$) $H\dot{-}_p \Phi \subseteq H$ *(inclusion)*

($H\dot{-}_p 4$) $H\dot{-}_p \Phi = H\dot{-}_p (H \cap \Phi)$ *(vacuity)*

($H\dot{-}_p 5$) $H\dot{-}_p \Phi = H\dot{-}_p (\Phi \setminus Cn^h(\top))$ *(failure)*

($H\dot{-}_p 5b$) $H\dot{-}_p \emptyset = H$ *(triviality)*

($H\dot{-}_p 6$) *If $\phi \leftrightarrow \psi$, then*
$H\dot{-}_p (\Phi \cup \{\phi\}) = H\dot{-}_p (\Phi \cup \{\psi\})$ *(extensionality)*

($H\dot{-}_p 7$) *If $H \neq H\dot{-}_p \Phi$ then for*

$$\Phi' \;=\; (\Phi \setminus Cn^h(\top)) \cap H \;=\; \{\phi_1, \ldots, \phi_n\}$$

there is $\{\beta_1, \ldots, \beta_n\}$ s.t. $\{\phi_i, \beta_i\} \vdash \bot$ and $H\dot{-}_p \Phi \subseteq Cn^h(\beta_i)$ for $1 \leq i \leq n$;
and $\forall H'$ s.t $H\dot{-}_p \Phi \subset H' \subseteq H$, $\exists \beta_i$ s.t. $H' \not\subseteq Cn^h(\beta_i)$.
(maximality)

The following result, which shows that package contraction generalises maxichoice contraction, is not surprising, nor is the next result, which shows that a maxichoice contraction defines a package contraction.

Proposition 5 *Let $\dot{-}_p$ be an operator of maxichoice Horn package contraction. Then*

$$H\dot{-}\phi = H\dot{-}_p \Phi \quad \textit{for } \Phi = \{\phi\}$$

is an operator of maxichoice Horn contraction based on weak remainders.

Proposition 6 *Let $\dot{-}$ be an operator of maxichoice Horn contraction based on weak remainders. Then*

$$H\dot{-}_p \Phi = \bigcap_{\phi \in \Phi} H\dot{-}\phi$$

is an operator of maxichoice Horn package contraction.

As described, a characteristic of maxichoice package contraction is that there are a large number of members of $H \Downarrow_p \Phi$, some of which may be quite weak logically. Of course, a similar point can be made about maxichoice contraction, but in the case of package contraction we can eliminate some candidates via pragmatic concerns. We have that a package contraction $H\dot{-}_p \Phi$ is a belief set $H' \in H \Downarrow_p \Phi$ such that, informally, models of H' contain a countermodel for each $\phi_i \in \Phi$ along with models of H. In general, some interpretations will be countermodels of more than one member of Φ, and so pragmatically, one can select minimal sets of countermodels. Hence in the case that $\bigcap_i (Mod(\top) \setminus Mod(\phi_i)) \neq \emptyset$, a single countermodel, that is some $m \in \bigcap_i (Mod(\top) \setminus Mod(\phi_i))$, would be sufficient to yield a package contraction.

Now, it may be that $\bigcap_i (Mod(\top) \setminus Mod(\phi_i))$ is empty. A simple example illustrates this case:

Example 4 *Let $H = Cn^h(a \to b, b \to a)$ where $\mathbf{P} = \{a, b\}$. Then $H\dot{-}_p \{a \to b, b \to a\} = Cn^h(\top)$. That is, the sole countermodel of $a \to b$ is $\{a\}$ while that of $b \to a$ is $\{b\}$. The intersection closure of these interpretations with those of H is $\{ab, a, b, \emptyset\} = Mod(\top)$.*

Informally then one can select a minimal set of models such that a countermodel of each member of Φ is in the set. These considerations yield the following definition:

Definition 6 *Let H be a Horn belief set, and let $\Phi = \{\phi_1, \ldots, \phi_n\}$ be a set of Horn formulas.*

$HS(\Phi)$, the set of (minimal) hitting sets *of interpretations with respect to Φ, is defined by:*

$S \in HS(\Phi)$ iff

1. $S \subseteq |\top|$
2. $S \cap (|\top| \setminus |\phi_i|) \neq \emptyset$ *for* $1 \leq i \leq n$.
3. *For* $S' \subset S$, $S' \cap (|\top| \setminus |\phi_i|) = \emptyset$ *for some* $1 \leq i \leq n$.

Thus we look for sets of sets of interpretations, elements of such a set S are interpretations represented as maximum consistent sets of formulas (Condition 1). As well, this set S contains a countermodel for each member of Φ (2) and moreover S is a subset-minimal set that satisfies these conditions (3). The notion of a hitting set is not new; see [Garey and Johnson, 1979] and see [Reiter, 1987] for an early use in AI. Thus $S \in HS(\Phi)$ corresponds to a minimal set of countermodels of members of Φ.

Definition 7 $H \Downarrow_p \Phi$ *is the set of sets of formulas s.t.*
$H' \in H \Downarrow_p \Phi$ *iff* $H' = H \cap \bigcap_{m \in S}$ *for some* $S \in HS(\Phi)$.

Proposition 7 *For* $H' \in H \Downarrow_p \Phi$*,* H' *is an operator of maxichoice Horn package contraction.*

Example 5 *Consider where* $H = Cn^h(a, b)$, $\mathbf{P} = \{a, b, c\}$.

1. *Let* $\Phi = \{a, b\}$. *We obtain that*

$$\begin{aligned} H \Downarrow_p \Phi \quad = \quad & \{\, Cn^h(\top),\ Cn^h(c \to a),\ Cn^h(c \to b), \\ & Cn^h(c \to a, c \to b), \\ & Cn^h(a \to b, b \to a), \\ & Cn^h(a \to b, b \to a, c \to a, c \to b)\ \}. \end{aligned}$$

Compare this with Example 3, where we have 14 candidate package contractions.

2. *Let* $\Phi = \{a, a \wedge b\}$. *We obtain that*

$$\begin{aligned} H \Downarrow_p \Phi \quad = \quad & \{\, Cn^h(b),\ Cn^h(b \wedge (c \to a)), \\ & Cn^h(a \to b, b \to a), \\ & Cn^h(a \to b, b \to a, c \to a, c \to b)\ \}. \end{aligned}$$

Any set of formulas that satisfies Definition 7 clearly also satisfies Definition 5. One can further restrict the set of candidate package contractions by replacing $S' \subset S$ by $|S'| < |S|$ in the third part of Definition 7. As well, of course, one could continue in the obvious fashions to define a notion of partial meet Horn package contraction.

6 Forgetting in Horn Formulas

This section examines another means of removing beliefs from an agent's belief set, that of *forgetting* [Lin and Reiter, 1994; Lang and Marquis, 2002]. Forgetting is an operation on belief sets and atoms of the language; the result of forgetting an atom can be regarded as decreasing the language by that atom.

In general it will be easier to work with a set of Horn clauses, rather than Horn formulas. Since there is no confusion, we will freely switch between sets of Horn clauses and the corresponding Horn formula comprising the conjunction of clauses in the set. Thus any time that a set appears as an element in a formula, it can be understood as standing for the conjunction of members of the set. Thus for sets of clauses S_1 and S_2, $S_1 \vee S_2$ will stand for the formula $(\bigwedge_{\phi \in S_1} \phi) \vee (\bigwedge_{\phi \in S_2} \phi)$. Of course, all such sets will be guaranteed to be finite.

We introduce the following notation for this section, where S is a set of Horn clauses.

- $S[p/t]$ is the result of uniformly substituting $t \in \{\bot, \top\}$ for atom p in S.
- $S_{\downarrow p} = \{\phi \in S \mid \phi \text{ does not mention } p\}$

Assume without loss of generality that for $\phi \in S$, that $head(\phi) \notin body(\phi)$.

The following definition adapts the standard definition for forgetting to Horn clauses.

Definition 8 *For set of Horn clauses* S *and atom* p*, define* $forget(S, p)$ *to be* $S[p/\bot] \vee S[p/\top]$.

This is not immediately useful for us, since a disjunction is generally not Horn. However, the next result shows that this definition nonetheless leads to a Horn-definable forget operator. Recall that for clauses c_1 and c_2, expressed as sets of literals where $p \in c_1$ and $\neg p \in c_2$, that the resolvent of c_1 and c_2 is the clause $(c_1 \setminus \{p\}) \cup (c_2 \setminus \{\neg p\})$. As well, recall that if c_1 and c_2 are Horn, then so is their resolvent.

In the following, $Res(S, p)$ is the set of Horn clauses obtained from S by carrying out all possible resolutions with respect to p.

Definition 9 *Let* S *be a set of Horn clauses and* p *an atom. Define*

$$\begin{aligned} Res(S, p) = \ & \{\phi \mid \exists \phi_1, \phi_2 \in S \text{ s.t. } p \in body(\phi_1), \\ & \quad p = head(\phi_2), \text{ and} \\ & \quad \phi = (body(\phi_1) \setminus \{p\} \cup body(\phi_2)) \to head(\phi_1)\} \end{aligned}$$

Theorem 5 $forget(S, p) \ \leftrightarrow\ S_{\downarrow p} \cup Res(S, p)$.

Corollary 1 *Let* S *be a set of Horn clauses and* p *an atom. Then* $forget(S, p)$ *is equivalent to a set of Horn clauses.*

Corollary 2 *Let* S_1 *and* S_2 *be sets of Horn clauses and* p *an atom. Then* $S_1 \leftrightarrow S_2$ *implies that* $forget(S_1, p) \leftrightarrow forget(S_2, p)$.

There are several points of interest about these results. The theorem is expressed in terms of arbitrary sets of Horn clauses, and not just deductively-closed Horn belief sets. Hence the second corollary states a principle of irrelevance of syntax for the case for forgetting for belief bases. As well, the expression $S_{\downarrow p} \cup Res(S, p)$ is readily computable, and so the theorem in fact provides a means of computing $forget$. Further, the approach clearly iterates for more than one atom. We obtain the additional result:

Corollary 3

$$forget(forget(S, p), q) \equiv forget(forget(S, q), p).$$

(In fact, this is an easy consequence of the definition of *forget*.) Given this, we can define for set of atoms A, $forget(S, A) = forget(forget(S, a), A \setminus \{a\})$ where $a \in A$. On the other hand, forgetting an atom may result in a quadratic blowup of the knowledge base.

Finally, it might seem that the approach allows for the definition of a revision operator – and a procedure for computing a revision – by using something akin to the Levi Identity. Let $\mathcal{A}(\phi)$ be the set of atoms appearing in (formula or set of formulas) ϕ. Then:

$$FRevise(S, \phi) \overset{def}{=} forget(S, \mathcal{A}(S) \cap \mathcal{A}(\phi)) + \phi.$$

In fact, this *does* yield a revision operator, but an operator that in general is far too drastic to be useful. To see this, consider a taxonomic knowledge base which asserts that whales are fish, $whale \rightarrow fish$. Of course, whales are mammals, but in using the above definition to repair the knowledge base, one would first forget *all* knowledge involving whales. Such an example doesn't demonstrate that there are no reasonable revision operators definable via *forget*, but it does show that a naïve approach is problematic.

7 Conclusions

This paper has collected various results concerning Horn belief set contraction. Earlier work has established a general framework for maxichoice and partial meet Horn contraction. The present paper then extends this work in various ways. We examined issues related to supplementary postulates, developed an approach to package contraction, and explored the related notion of forgetting. For future work, it would be interesting to investigate relationships between remainder-based and entrenchment-based Horn contraction, as well as to explore connections to constructions for (Horn) belief revision.

References

[Alchourrón *et al.*, 1985] C.E. Alchourrón, P. Gärdenfors, and D. Makinson. On the logic of theory change: Partial meet functions for contraction and revision. *Journal of Symbolic Logic*, 50(2):510–530, 1985.

[Booth *et al.*, 2009] Richard Booth, Thomas Meyer, and Ivan José Varzinczak. Next steps in propositional Horn contraction. In *Proceedings of the International Joint Conference on Artificial Intelligence*, Pasadena, CA, 2009.

[Delgrande and Wassermann, 2010] James Delgrande and Renata Wassermann. Horn clause contraction functions: Belief set and belief base approaches. In Fangzhen Lin and Uli Sattler, editors, *Proceedings of the Twelfth International Conference on the Principles of Knowledge Representation and Reasoning*, pages 143–152, Toronto, 2010. AAAI Press.

[Delgrande, 2008] J.P. Delgrande. Horn clause belief change: Contraction functions. In Gerhard Brewka and Jérôme Lang, editors, *Proceedings of the Eleventh International Conference on the Principles of Knowledge Representation and Reasoning*, pages 156–165, Sydney, Australia, 2008. AAAI Press.

[Eiter and Gottlob, 1992] T. Eiter and G. Gottlob. On the complexity of propositional knowledge base revision, updates, and counterfactuals. *Artificial Intelligence*, 57(2-3):227–270, 1992.

[Flouris *et al.*, 2004] Giorgos Flouris, Dimitris Plexousakis, and Grigoris Antoniou. Generalizing the AGM postulates: Preliminary results and applications. In *Proceedings of the 10th International Workshop on Non-Monotonic Reasoning (NMR-04)*, pages 171–179, Whistler BC, Canada, June 2004.

[Gärdenfors and Makinson, 1988] P. Gärdenfors and D. Makinson. Revisions of knowledge systems using epistemic entrenchment. In *Proc. Second Theoretical Aspects of Reasoning About Knowledge Conference*, pages 83–95, Monterey, Ca., 1988.

[Gärdenfors, 1988] P. Gärdenfors. *Knowledge in Flux: Modelling the Dynamics of Epistemic States*. The MIT Press, Cambridge, MA, 1988.

[Garey and Johnson, 1979] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Co., New York, 1979.

[Hansson, 1999] S. O. Hansson. *A Textbook of Belief Dynamics*. Applied Logic Series. Kluwer Academic Publishers, 1999.

[Khardon, 1995] Roni Khardon. Translating between Horn representations and their characteristic models. *Journal of Artificial Intelligence Research*, 3:349–372, 1995.

[Lang and Marquis, 2002] J. Lang and P. Marquis. Resolving inconsistencies by variable forgetting. In *Proceedings of the Eighth International Conference on the Principles of Knowledge Representation and Reasoning*, pages 239–250, San Francisco, 2002. Morgan Kaufmann.

[Langlois *et al.*, 2008] M. Langlois, R.H. Sloan, B. Szörényi, and G. Turán. Horn complements: Towards Horn-to-Horn belief revision. In *Proceedings of the AAAI National Conference on Artificial Intelligence*, Chicago, Il, July 2008.

[Liberatore, 2000] Paolo Liberatore. Compilability and compact representations of revision of Horn knowledge bases. *ACM Transactions on Computational Logic*, 1(1):131–161, 2000.

[Lin and Reiter, 1994] F. Lin and R. Reiter. Forget it! In *AAAI Fall Symposium on Relevance*, New Orleans, November 1994.

[Reiter, 1987] R. Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32(1):57–96, 1987.

[Zhuang and Pagnucco, 2010a] Z. Zhuang and Maurice Pagnucco. Two methods for constructing horn contractions. *AI 2010: Advances in Artificial Intelligence*, pages 72–81, 2010.

[Zhuang and Pagnucco, 2010b] Zhi Qiang Zhuang and Maurice Pagnucco. Horn contraction via epistemic entrenchment. In Tomi Janhunen and Ilkka Niemelä, editors, *Logics in Artificial Intelligence - 12th European Conference (JELIA 2010)*, volume 6341 of *Lecture Notes in Artificial Intelligence*, pages 339–351. Springer Verlag, 2010.

A Selective Semantics for Logic Programs with Preferences

Alfredo Gabaldon
CENTRIA – Center for Artificial Intelligence
Universidade Nova de Lisboa
ag@di.fct.unl.pt

Abstract

Agents in complex domains need to be able to make decisions even if they lack complete knowledge about the state of their environment. One approach that has been fairly successful is to use logic programming with answer set semantics (ASP) to represent the beliefs of the agent and solve various reasoning problems such as planning. The ASP approach has been extended with preferences and several semantics have been proposed for selecting preferred answer sets of a logic program. Among the available semantics, one proposed by Brewka and Eiter has been shown to be more permissive than others in the sense of allowing the selection of a larger number of answer sets as the preferred ones. Although the semantics is permissive enough to allow multiple preferred answer sets even for some fully prioritized programs, there are on the other hand programs that have answer sets but not preferred ones. We consider a semantics that selects at most one answer set as the preferred one for fully prioritized (propositional) programs, and show that programs in a large class guaranteed to have an answer set (negative-cycle-free, head-consistent) are also guaranteed to have a preferred answer set.

1 Introduction

Reasoning with preferences is widely recognized as an important problem. Many knowledge representation formalisms have been extended to represent preferences as priorities between propositions in a knowledge base. In particular, prioritized versions of logic programming with answer set semantics [Gelfond and Lifschitz, 1991; Gelfond, 2008] have been studied and various semantics have been proposed [Sakama and Inoue, 1996; Gelfond and Son, 1997; Zhang and Foo, 1997; Brewka and Eiter, 1999; Delgrande *et al.*, 2000; Wang *et al.*, 2000]. Some of these approaches [Brewka and Eiter, 1999; Delgrande *et al.*, 2000; Wang *et al.*, 2000] are classified as being *selective* in the sense that the preferences are effectively used as a selection mechanism for choosing among the answer sets of a logic program. That is, the preferred answer sets are always chosen from the collection of standard answer sets of the logic program. Another characteristic of the selective approaches is that most of them extend logic programs with preferences without increasing computational complexity.

In this work we focus on a selective approach and propose a new semantics for answer set programming with preferences. The main motivation for introducing a new semantics is that all of the existing selective approaches seem to be too strong in the sense that there are programs that possess answer sets but not preferred answer sets. At the same time, the same approaches seem to be weak in the sense that there are programs that possess multiple answers sets that cannot be distinguished apart even by a full prioritization. Our proposed semantics yields at most one preferred answer set when a complete set of priorities is specified. Moreover, for a large class of propositional logic programs (called *negative-cycle-free* and *head-consistent*) that are guaranteed to have answer sets, we show that a preferred answer set always exists under our proposed semantics. In the case of logic programs without classical negation, this is the most general known class of programs guaranteed to have answer sets [Baral, 2003].

Our starting point of reference is the preferred answer set semantics introduced by Brewka and Eiter [1999] (for brevity, we will refer to this semantics as the BE semantics). Among the selective semantics within the NP complexity class, the BE semantics is the least restrictive in the sense that, for a given logic program with a fixed set of preferences, it selects a collection of preferred answer sets that is a superset of those selected by the other approaches, as shown in [Schaub and Wang, 2001]. In other words, if a program does not have preferred answer sets under the BE semantics, neither does it have any preferred answer sets under the other selective semantics. Since our aim is a semantics that always assigns a preferred answer set to a large class of logic programs, the BE semantics seems to be a good point of reference and comparison.

2 Prioritized Extended Logic Programs

In this section we give an overview of the answer set semantics and of the BE preferred answer set semantics.

2.1 Extended Logic Programs

We start with the syntax of Extended Logic Programs (elps) [Gelfond and Lifschitz, 1991; Gelfond, 2008]. A *literal* is an atom p or its negation $\neg p$. Literals p and $\neg p$ are called *contrary* and $\bar{l}$ denotes the literal contrary to l. If the language of an elp is not explicitly defined then it is understood to consist of all the atoms that appear in the program. Lit denotes the set of all literals in the language of an elp.

A rule r is an expression of the form

$$l_0 \leftarrow l_1, \ldots, l_n, not\, l_{n+1}, \ldots, not\, l_m \qquad (1)$$

where $l_0, \ldots, l_m$ are literals and not denotes negation-as-failure (or *default negation*). Expressions $not\, l$ are called *extended literals*. For a rule r of the form (1), the head, l_0, is denoted by $head(r)$, the set of literals $\{l_1, \ldots, l_n\}$ by $body^+(r)$ and the set of literals $\{l_{n+1}, \ldots, l_m\}$ by $body^-(r)$. An *extended logic program* is a finite set of rules.

A set of literals $S \subseteq Lit$ is called a *partial interpretation*. A rule r is said to be *defeated by a literal* l if $l \in body^-(r)$. A partial interpretation S *defeats a rule* r if there is a literal $l \in S$ that defeats r. S *satisfies the body* of a rule r if $body^+(r) \subseteq S$ and S does not defeat r. S *satisfies* r if $head(r) \in S$ or S does not satisfy the body of r.

The answer sets of an elp whose rules do not contain not is defined as follows.

Definition 1. Let P be an elp without default negation. A partial interpretation S is an *answer set* of P if S is minimal (wrt set inclusion) among the partial interpretations that satisfy the rules of P, and S is logically closed, i.e. if S contains contrary literals then S is Lit.

For arbitrary programs, the definition is extended by introducing the Gelfond-Lifschitz reduct: let S be a partial interpretation and P be an elp. The *reduct*, P^S, of P relative to S is the set of rules $l_0 \leftarrow l_1, \ldots, l_n$ for all rules (1) in P that are not defeated by S.

Definition 2. (Answer Set) A partial interpretation S is an *answer set* of an elp P if S is an answer set of P^S.

2.2 Prioritized extended logic programs

We now turn to *prioritized elps*, adapting the definitions from [Brewka and Eiter, 1999]. Let us start with the syntax.

An elp rule r of the form (1) is called *prerequisite-free* if $body^+(r) = \emptyset$ and an elp P is *prerequisite-free* if all its rules are prerequisite-free.

Definition 3. A *prioritized elp* is a pair $\mathcal{P} = (P, <)$ where P is an elp and $<$ is a strict partial order on the rules of P.

The answer sets of a prioritized elp $\mathcal{P} = (P, <)$ are defined as the answer sets of P and are denoted by $AS(\mathcal{P})$.

Definition 4. A *full prioritization* of a prioritized elp $\mathcal{P}$ is any pair $\mathcal{P}' = (P, <')$ where $<'$ is a total order on P that is compatible with $<$, i.e., $r_1 < r_2$ implies $r_1 <' r_2$ for all r_1, r_2 in P.

The total ordering in a fully prioritized elp induces an enumeration $r_1, r_2, \ldots$ of its rules with r_1 having the highest priority. Throughout the paper, we use such an enumeration in examples and write

$$r_i : l \leftarrow l_1, \ldots, l_n, not\, l_{n+1}, \ldots, not\, l_m$$

to denote the ith rule in such an enumeration.

Let us next look at the BE preferred answer set semantics. We will refer to preferred answers sets under the BE semantics as BE-preferred answer sets. These definitions are simplified versions of those in [Brewka and Eiter, 1999] as we focus in this work on propositional programs.

Definition 5. Let $\mathcal{P} = (P, <)$ be a fully prioritized elp where P is a set of n prerequisite-free rules and let S be a set of literals. The sequence of sets $S_0, S_1, \ldots, S_n$ is defined as follows: $S_0 = \emptyset$ and for $0 < i \leq n$,

$$S_i = \begin{cases} S_{i-1}, & \text{if } r_i \text{ is defeated by } S_{i-1} \text{ or} \\ & head(r_i) \in S \text{ and} \\ & \quad r_i \text{ is defeated by } S, \\ S_{i-1} \cup \{head(r_i)\}, & \text{otherwise.} \end{cases}$$

The set $C_{\mathcal{P}}(S)$ is defined to be the smallest set of literals such that $S_n \subseteq C_{\mathcal{P}}(S)$ and $C_{\mathcal{P}}(S)$ is logically closed (consistent or equal to Lit).

Definition 6. (BE-preferred Answer Sets) Let $\mathcal{P} = (P, <)$ be a fully prioritized elp with prerequisite-free P and let A be an answer set of P. Then A is the *BE-preferred answer set* of $\mathcal{P}$ iff $A = C_{\mathcal{P}}(A)$.

As the definition suggests, a fully prioritized, prerequisite-free elp has at most one BE-preferred answer set.

For non prerequisite-free prioritized elps, a transformation is applied similar to the Gelfond-Lifschitz reduct but that produces rules without prerequisites. The precise definition is as follows.

Definition 7. Let $\mathcal{P} = (P, <)$ be a fully prioritized elp and let S be a set of literals. Define ${}^S\mathcal{P} = ({}^SP, {}^S{<})$ to be the fully prioritized elp such that ${}^SP$ is the set of rules obtained from P by

1. deleting every rule $r \in P$ s.t. $body^+(r) \not\subseteq S$, and
2. deleting $body^+(r)$ from every remaining rule r;

and ${}^S{<}$ is inherited from $<$ by the mapping $f : {}^SP \mapsto P$ where $f(r')$ is the first rule in P wrt $<$ such that r' results from r by step (2) above. In other words, for every $r_1, r_2 \in P$, $r_1' \;{}^S{<}\; r_2'$ iff $f(r_1') < f(r_2')$.

Definition 8. A set of literals A is a BE-preferred answer set of a fully prioritized elp $\mathcal{P} = (P, <)$ iff A is a BE-preferred answer set of ${}^A\mathcal{P}$.

In the next section we present some examples illustrating this semantics, including programs without preferred answer sets and fully prioritized programs with multiple ones.

3 Limitations of existing semantics

As we discussed in the introduction, the motivation for proposing a new semantics for prioritized logic programs is twofold. First, there are elps that while containing no discernible conflict or contradiction and indeed possessing answer sets, have no BE-preferred answer sets and therefore no preferred answer sets under the other semantics which are more restrictive. Second, there are programs that even by providing a full prioritization of its rules, it is not possible to select only one of the answer sets as the preferred one. Most of the following examples are from [Brewka and Eiter, 1999]. Recall that r_i means the rule is the ith rule in the enumeration of the rules by priority.

Example 1. Consider the program $\mathcal{P}_1$ with rules

$$\begin{array}{l} r_1: \ c \leftarrow \ not\, b. \\ r_2: \ b \leftarrow \ not\, a. \end{array}$$

This program has one answer set, $A = \{b\}$. Since $c \notin A$ nor is r_1 defeated by $\emptyset$, $c \in C_{\mathcal{P}_1}(A)$. Therefore this program has no BE-preferred answer sets.

Brewka and Eiter's approach to preferences is based on the notion (which we follow as well) that preferences are introduced in order to "solve potential conflicts...to conclude more than in standard answer semantics." [Brewka and Eiter, 1999]. Since the rules in the above program show no apparent conflict between them and in fact the program has only one answer set, it seems reasonable that it should have a preferred answer set.

The following example shows this shortcoming even more directly, since one of the rules is a fact, i.e. does not involve defaults at all.

Example 2. Consider the program $\mathcal{P}_2$ with rules

$$\begin{array}{l} r_1: \ a \leftarrow \ not\, b. \\ r_2: \ b. \end{array}$$

This program has one answer set, $A = \{b\}$. By a similar argument as in the previous example, we have that $a \in C_{\mathcal{P}_2}(A)$ and so the program has no BE-preferred answer sets.

The above examples seem to show that the semantics is in some sense too strong (and so are other proposed selective semantics which have been shown to be even stronger). On the other hand, as already mentioned, in some cases this semantics assigns multiple preferred answer sets to programs that are fully prioritized. In other words, under the BE-preferred answer set semantics there are cases where it is not possible to "solve potential conflicts" completely even with a full prioritization. Consider the following example.

Example 3. Consider the program $\mathcal{P}_3$ with rules

$$\begin{array}{l} r_1: \ b \leftarrow \ not\, \neg b, \ a. \\ r_2: \ c \leftarrow \ not\, b. \\ r_3: \ a \leftarrow \ not\, c. \end{array}$$

This fully prioritized elp has two answer sets: $A_1 = \{c\}$ and $A_2 = \{a, b\}$. They are both BE-preferred answer sets. Consider A_1. Rule r_1 does not belong to the reduct of the program since prerequisite $a \notin A_1$. Then rule r_2 is not defeated by $\emptyset$ nor by A_1, so we get c which then defeats rule r_3 and we have $A_1 = C_{\mathcal{P}_3}(A_1)$. Now consider A_2. Prerequisite a is removed from r_1 in the reduct of the program. Then we have that rule r_1 is not defeated by $\emptyset$ nor by A_2, so we get b which then defeats rule r_2 allowing r_3 to fire. Thus we have $A_2 = C_{\mathcal{P}_3}(A_2)$.

This program is already fully prioritized. It is not possible to use further priorities to select one of the two answer sets as the preferred one. Ideally, a fully prioritized elp should have at most one preferred answer set.

4 A new semantics for prioritized elps

Our proposed new semantics is intuitively based on the view, following Brewka and Eiter, that priorities are used to resolve conflicts. Intuitively, it is also based on the idea of taking conflicts between rules somewhat more literally by appealing to a notion of "attack" that is to some degree inspired by argument attacks in argumentation. Here, by an attack of a rule on another we simply mean that if the attacking rule fires, it will defeat the other one. We then consider rules to be in conflict when they attack each other, as in the program:

$$\begin{array}{l} a \leftarrow \ not\, b. \\ b \leftarrow \ not\, a. \end{array}$$

But these attacks can be indirect, through a chain of other rules, as in the program:

$$\begin{array}{l} a \leftarrow \ c. \\ b \leftarrow \ not\, a. \\ c \leftarrow \ not\, b. \end{array}$$

Here, the second rule attacks the first indirectly through the third rule.

In order to simplify the development of our semantics for prioritized logic programs, we appeal to a well know unfolding operation which would transform the above program into the program:

$$\begin{array}{l} a \leftarrow \ not\, b. \\ b \leftarrow \ not\, a. \\ c \leftarrow \ not\, b. \end{array}$$

The formal definitions follow.

Definition 9. (Unfolding [Aravindan and Dung, 1995]) Let P_i be an elp and r be a rule in P_i of the form $H \leftarrow L, \Gamma$ where L is a literal different from H and Γ is the rest of the rule's body. Suppose that $r_1, \ldots, r_k$ are all the rules in P_i such that each r_j is of the form $L \leftarrow \Gamma_j$ such that $L \notin body^+(r_j)$. Then $P_{i+1} = (P_i \setminus \{r\}) \cup \{H \leftarrow \Gamma_j, \Gamma : 1 \leq j \leq k\}$. This operation is called *unfolding* r in P_i (or unfolding in general), r is called the *unfolded* rule, and L is called the *selected* literal in the unfolding.

The answer set semantics is one of the logic programming semantics that satisfies the *Generalized Principle of Partial Evaluation* [Aravindan and Dung, 1995; Dix, 1995; Brass and Dix, 1999], which means that the unfolding transformation above results in a program that has exactly the same answer sets as the original program.

Let us define the unfolding operation for prioritized elps. For our purposes it suffices to define it for fully prioritized elps.

An *unfolding operation for fully prioritized elps* is defined as follows. $\mathcal{P}_{i+1} = (P_{i+1}, <_{i+1})$ is the result of applying an unfolding on $\mathcal{P}_i = (P_i, <_i)$ if

1. P_{i+1} is the result of unfolding $r \in P_i$ such that r is replaced with rules $r'_1, \ldots, r'_k$, and
2. for each rule r'_j obtained in the previous step, if $r'_j \in P_i$, i.e. an identical rule was already in the program, then

(a) if $r'_j <_i r$ then let $r'_j <_{i+1} r^*$ (resp. $r^* <_{i+1} r'_j$) for every rule r^* such that $r'_j <_i r^*$ (resp. $r^* <_i r'_j$), i.e. r'_j retains the same priority, since it has higher priority in $\mathcal{P}_i$ than the unfolded rule r.

(b) if $r <_i r'_j$ then let $r'_j <_{i+1} r^*$ (resp. $r^* <_{i+1} r'_j$) for every rule r^* such that $r <_i r^*$ (resp. $r^* <_i r$), i.e. r'_j now has the same priority r has in $\mathcal{P}_i$, since r had higher priority.

3. for each rule r'_j obtain in step one such that $r'_j \notin P_i$, i.e. it is a new rule, $<_{i+1}$ extends $<_i$ with the priorities $r'_j <_{i+1} r^*$ (resp. $r^* <_{i+1} r'_j$) if $r <_i r^*$ (resp. $r^* <_i r$), i.e. these new rules are assigned the same priority r has in $\mathcal{P}_i$.

It is easy to see that applying an unfolding operation results in a fully prioritized elp.

Definition 10. A *transformation sequence* is a sequence of fully prioritized elps $\mathcal{P}_0, \ldots, \mathcal{P}_n$ such that each $\mathcal{P}_{i+1}$ is obtained by applying an unfolding operation on $\mathcal{P}_i$.

Definition 11. The *unfolding of* a fully prioritized $\mathcal{P}$, denoted $\overline{\mathcal{P}}$, is the fully prioritized elp $\mathcal{P}_n$ such that there is a transformation sequence $\mathcal{P}_0, \ldots, \mathcal{P}_n$ where $\mathcal{P}_0 = \mathcal{P}$ and there is no rule in P_n that can be unfolded.

Example 4. Consider again the program

$$\begin{array}{lcl} r_1 : b & \leftarrow & not\, \neg b,\ a. \\ r_2 : c & \leftarrow & not\, b. \\ r_3 : a & \leftarrow & not\, c. \end{array}$$

The unfolding of this program consists of the following rules:

$$\begin{array}{lcl} r'_1 : b & \leftarrow & not\, \neg b,\ not\, c. \\ r_2 : c & \leftarrow & not\, b. \\ r_3 : a & \leftarrow & not\, c. \end{array}$$

Here, the unfolding helps reveal more directly that there is a conflict between the first two rules: in the unfolded program the head of one rule appears negated in the body of the other and vice-versa.

Let us now proceed to define our proposed preferred answer set semantics, starting with the semantics for unfolded, fully prioritized elps. We start with some terminology.

Let P be an elp and X be a set of literals. We say a *literal* l *holds in* X if $l \in X$. An *extended literal* $not\, l$ *is defeated by* X if l holds in X. Obviously, it is possible for $not\, l$ not to hold nor be defeated in X. A *rule* r *is defeated by* X if there is a literal $l \in body^-(r)$ such that $not\, l$ is defeated by X. An *extended literal* $not\, l$ *holds in* X if $\overline{l}$ holds in X or if every rule $r \in P$ whose $head(r) = l$ is defeated in X. For a rule r, the *body,* $body(r)$*, holds in* X if $body^+(r)$ holds in X and $not\, l$ holds in X for each $l \in body^-(r)$. A *rule* r *is active in* X if neither $head(r)$ nor $\overline{head(r)}$ holds, $body^+(r)$ holds and r is not defeated, in X.[1]

We will use the notation $r \to r'$ to mean that $head(r) \in body^-(r')$, and $r \twoheadrightarrow r'$ to mean that there is a sequence $r \to r_1,\ r_1 \to r_2, \ldots, r_k \to r'$ where k is an odd number. We say that r *attacks* r' *in* X if r is active in X and $r \to r'$.

Definition 12. Let $\mathcal{P} = (P, <)$ be an unfolded, fully prioritized elp. We define the sequence $X_0, X_1, \ldots$ satisfying the the following conditions: $X_0 = \emptyset$ and $X_{i+1} = X_i \cup \{head(r)\}$ such that r is active in X_i and

1. $body(r)$ holds in X_i; or
2. there is no active r s.t. $body(r)$ holds in X_i (the previous case does not apply for any rule) and for all r', if r' attacks r then $r \twoheadrightarrow r'$ and $r < r'$.[2]

Intuitively, in each iteration we first check if there are any rules whose body is definitely satisfied by X_i. If so, we add the heads of those rules and skip to the next iteration. If there are no rules whose body is satisfied, then the second case adds the heads of all rules r which are not attacked by any rule, or if they are attacked by a rule r', the rules are in an even cycle and r has higher priority.

Proposition 1. *There exists n such that for all* $m > n$, $X_m = X_n$*, i.e., the sequence reaches a fixpoint.*

Let $I_\mathcal{P}$ be the fixpoint set X_n as defined above if X_n does not contain contrary literals, or Lit otherwise.

Definition 13. Let $\mathcal{P}$ be an unfolded, fully prioritized elp. The set $I_\mathcal{P}$ is a *preferred answer set* of $\mathcal{P}$ if $I_\mathcal{P}$ is an answer set of $\mathcal{P}$.

It trivially follows from this definition that all preferred answer sets of a prioritized elp $\mathcal{P}$ are answer sets of $\mathcal{P}$. It is also easy to see that according to these definitions, if $\mathcal{P}$ has a preferred answer set at all, it has one: $I_\mathcal{P}$.

The computation of $I_\mathcal{P}$ may fail to produce one of the answer sets of the program, hence the test done afterwards to check whether it is one them. For programs that are not negative-cycle-free, the computation may reach a fixpoint prematurely. The computation may also fail by producing a set that contains contrary literals. Later we show that for negative-cycle-free, head-consistent programs, this computation is guaranteed to produce one of the answer sets, making the check unnecessary.

For an arbitrary prioritized elp, preferred answer sets are defined as follows.

Definition 14. For an arbitrary prioritized elp $\mathcal{P}$, A is a preferred answer set of $\mathcal{P}$ if A is a preferred answer set of the unfolding $\overline{\mathcal{P}'}$ of one of the full prioritizations $\mathcal{P}'$ of $\mathcal{P}$.

The set of all preferred answer sets of $\mathcal{P}$ will be denoted by $PAS(\mathcal{P})$.

Given the above definition, the most interesting programs to analyze are the fully prioritized programs. Therefore all our examples use the latter.

[1] A similar notion of "active rule" is used in [Schaub and Wang, 2001]

[2] According to this definition, the rules in the odd path need not be active. But when a rule in the cycle from r to r' is not active then there is no longer a conflict to be resolved by priorities.

Example 5. Consider the program from Example 1:

$$\begin{array}{l} r_1 : \; c \; \leftarrow \; \mathit{not}\, b. \\ r_2 : \; b \; \leftarrow \; \mathit{not}\, a. \end{array}$$

Since there is no rule with head a, $\mathit{body}(r_2)$ holds in $X_0 = \emptyset$, so $X_1 = \{b\}$ which is the fixpoint. Since $\{b\}$ is also an answer set, it is the preferred answer set.

Example 6. Consider next the program from Example 3 which has the following unfolding (also shown in Example 4):

$$\begin{array}{l} r_1' : \; b \; \leftarrow \; \mathit{not}\, \neg b, \; \mathit{not}\, c. \\ r_2 : \; c \; \leftarrow \; \mathit{not}\, b. \\ r_3 : \; a \; \leftarrow \; \mathit{not}\, c. \end{array}$$

Extended literal $\mathit{not}\, \neg b$ holds in $X_0 = \emptyset$ since there are no rules with head $\neg b$. Also, r_2 attacks r_1' but $r_1' \twoheadrightarrow r_2$ and $r_1' < r_2$. So r_1' fires. On the other hand, both r_2, r_3 are attacked by higher priority rules that are active in X_0. Hence $X_1 = \{b\}$. Then r_2 is defeated in $\{b\}$ so it is no longer active. Hence $X_2 = \{a, b\}$, which is the fixpoint, an answer set, and therefore the (only) preferred answer set.

The following example shows that there are programs that have no preferred answer sets.

Example 7. Consider the program:

$$\begin{array}{l} r_1 : \; p \; \leftarrow \; \mathit{not}\, p, \; a. \\ r_2 : \; a \; \leftarrow \; \mathit{not}\, b. \\ r_3 : \; b \; \leftarrow \; \mathit{not}\, a. \end{array}$$

which unfolds into the program consisting of

$$r_1' : \; p \; \leftarrow \; \mathit{not}\, p, \; \mathit{not}\, b.$$

and r_2, r_3.

In this case r_1 attacks itself so it never fires. The resulting fixpoint is $\{a\}$ which is not an answer set and therefore not a preferred answer set. This program then has an answer set, $\{b\}$, but no preferred answer sets according to our semantics.

The above program does not have BE-preferred answer sets either. The fact that it does not have preferred answer sets is not really surprising. The program essentially has two parts, one part consisting of rules r_2, r_3 which intuitively generates a choice between a and b, and rule r_1 which says that answer sets that contain a must be ruled out. But the priorities on the bottom two rules say to prefer a which conflicts with the constraint represented by r_1. Note that if the priorities on r_2, r_3 are relaxed, i.e. the priority $r_2 < r_3$ is removed, then $\{b\}$ is a preferred answer set.

It is well know in answer set programming that cycles with an odd number of negative edges in the dependency graph, such as the cycle involving p and rule r_1 above, are used as constraints that eliminate answer sets. In the following section we show that absent such constraints, a prioritized elp is guaranteed to have a preferred answer set according to our semantics.

5 Properties

We start with a result establishing that prioritized elps with our proposed semantics are a conservative extension of elps.

Given an elp P, a set of literals S is said to be *generated by R* if R is the set of all the rules $r \in P$ whose bodies are satisfied by S and $\mathit{head}(r) \in S$.

Theorem 1. *Let $\mathcal{P} = (P, <)$ be a prioritized elp with empty $<$, that is, without priorities. Then $AS(\mathcal{P}) = PAS(\mathcal{P})$.*

Proof. By definition $PAS(\mathcal{P}) \subseteq AS(\mathcal{P})$. We show that $AS(\mathcal{P}) \subseteq PAS(\mathcal{P})$. Note that, by the equivalence preserving property of the unfolding operation, $AS(\mathcal{P}) = AS(\overline{\mathcal{P}})$. Thus it suffices to show that $AS(\overline{\mathcal{P}}) \subseteq PAS(\overline{\mathcal{P}})$. Note that the preference relation is empty so $\overline{\mathcal{P}}$ is defined in terms the original unfolding without priorities. Let $A \in AS(\overline{\mathcal{P}})$. Suppose $A = \mathit{Lit}$, then there are two rules r_1, r_2 in $\overline{P}^A$ s.t. $\mathit{head}(r_1) = \overline{\mathit{head}(r_2)}$ and s.t. the smallest set S closed under the rules of $\overline{P}^A$ satisfies the bodies of r_1, r_2. Since $r_1, r_2 \in \overline{P}^A$, $\mathit{body}^-(r_1) = \mathit{body}^-(r_2) = \emptyset$. Also, since r_1, r_2 are unfolded and their bodies satisfied by S, $\mathit{body}^+(r_1) = \mathit{body}^+(r_2) = \emptyset$. Therefore, for any full prioritization of $\mathcal{P}$, $I_{\mathcal{P}} = \mathit{Lit}$. Suppose $A \neq \mathit{Lit}$ and let R be the generating rules of A. Consider a full prioritization $\mathcal{P}' = (P', <')$ of $\overline{\mathcal{P}}$ where for every rule $r_1 \in R$ and $r_2 \in (P' \setminus R)$, $r_1 <' r_2$. Since rules R are generating rules, there are no rules $r_1, r_2 \in R$ s.t. $r_1 \rightarrow r_2$, i.e. rules in R never attack each other. Consider a rule $r \notin R$. Since this rule is not generating, then either there is a literal $l \in \mathit{body}^+(r)$ s.t. there is no rule $r' \in P'$ with $\mathit{head}(r') = l$, or there is a rule $r' \in R$ s.t. $r \rightarrow r'$. In the former case, r is never active. In the latter case, since $r' < r$, r is first attacked and then defeated. Thus $\mathit{head}(r) \notin I_{\mathcal{P}'}$. On the other hand, every rule $r' \in R$ is attacked only by rules in $(P' \setminus R)$ which are defeated in $I_{\mathcal{P}'}$. Therefore, $\mathit{head}(r') \in I_{\mathcal{P}'}$. We conclude that $I_{\mathcal{P}'} = A$ and therefore that A is a preferred answer set. □

As we discussed in the introduction, one of the motivations for a new semantics is the intention that by adding priorities it should be possible to select only one of the multiple answer sets of a program. Example 7 shows that the existence of answer sets does not guarantee the existence of preferred answer sets, although this seems to occur only in programs involving implicit constraints. The following theorem establishes the existence of preferred answer sets for a class of programs where constraints are ruled out. An elp P is said to be *head-consistent* if the set of literals $\{\mathit{head}(r) : r \in P\}$ is consistent, i.e. does not contain contrary literals. It is said to be *negative-cycle-free* if the dependency graph of P does not contain cycles with an odd number of negative edges.

Theorem 2. *Let $\mathcal{P} = (P, <)$ be a fully prioritized elp s.t. P is negative-cycle-free and head-consistent. Then $\mathcal{P}$ has a preferred answer set.*

Proof. Consider the program P' of the unfolding $\overline{\mathcal{P}}$. Unfolding cannot add negative cycles, so P' is negative-cycle-free. Let X denote the set $I_{\overline{\mathcal{P}}}$. We show that X is an answer set of P'. Let A be the answer set of the reduct P'^X. For any rule $r \in P'$, by r'^X we denote the rule that results from r in computing the reduct P'^X. If r is deleted from the reduct, we will write $r'^X \notin P'^X$.

$X \subseteq A$: Assume $X \not\subseteq A$. Let $X_0, X_1, \ldots, X_n = X$ be the sequence of $I_{\overline{\mathcal{P}}}$. Let i be the smallest such that $X_i \not\subseteq A$. Let l be any literal such that $l \in X_i$ but $l \notin A$. There must be a rule $r \in P'$ such that $l = \mathit{head}(r)$ and r is active in X_{i-1}. Then either $\mathit{body}(r)$ holds in X_{i-1} or for every rule r' that attacks r in X_{i-1}, $r < r'$ and $r \twoheadrightarrow r'$. In case $\mathit{body}(r)$ holds in $X_{i-1} \subseteq A$, then $l \in A$. Contradiction. Consider otherwise any literal $b \in \mathit{body}^-(r)$ and any rule $r' \in P'$ with $b = \mathit{head}(r')$. Then 1) r' is not active in X_{i-1} or 2) $r < r'$ and $r \twoheadrightarrow r'$. In case (1), we have three possibilities: i) $b \in \mathit{body}^+(r')$; ii) $\neg b \in X_{i-1}$ hence $\neg b \in X$ and $\neg b \in A$, therefore $b \notin A$ unless $A = \mathit{Lit}$ which is a contradiction; iii) r' is defeated in X_{i-1} and hence defeated in A. We conclude that for any rule r' not active in X_{i-1}, if $r'^X \in P'^X$ then $b \in \mathit{body}^+(r')$. In case (2), we have that $l \in \mathit{body}^-(r')$ hence $r'^X \notin P'^X$. We conclude that the only rules in P'^X with head b have b in the positive body. Therefore $b \notin A$. Since this holds for any $b \in \mathit{body}^-(r)$ we have that $\mathit{body}^-(r^X) \cap A = \emptyset$. Since r is active in X_{i-1}, $\mathit{body}^+(r)$ holds in X_{i-1} and in A. Since P' is head-consistent, $X \neq \mathit{Lit}$ and $r^X \in P'^X$. Therefore $l \in A$. Contradiction.

$A \subseteq X$: Assume $A \not\subseteq X$. Let l be any literal such that $l \in A$ but $l \notin X$. There must be a rule $r \in P$ such that $l = \mathit{head}(r)$ and $r^X \in P'^X$. Since $l \in A$ and P' is unfolded, $\mathit{body}^+(r) = \emptyset$ and $\mathit{body}^-(r) \cap X = \emptyset$. This means that r is active in X. Since $l \notin X$, there is at least one rule that attacks r in X. Let r' be any rule that attacks r in X. Since $r^X \in P'^X$, $\mathit{head}(r') \notin X$, and since r' attacks r in X, r' is active in X. Since $\mathit{head}(r') \notin X$, r' must be attacked in X by some other rule whose head is not in X either. This implies that there is a set of rules active in X that includes r, r' and that are in a cycle of attacks. Consider the largest such cycle. Since P is negative-cycle-free, the number of rules in the cycle is even. Let r_1 be the rule in the cycle with the highest priority. For any rule r_2 that attacks r_1 in X it must be the case that $r_1 < r_2$ and $r_1 \twoheadrightarrow r_2$. But this implies that $\mathit{head}(r_1) \in X$ and therefore that either $l \in X$ or $\mathit{head}(r') \in X$. Contradiction. □

Corollary 1. *If $\mathcal{P}$ is a fully prioritized elp with negative-cycle-free and head-consistent P, then the set $I_{\overline{\mathcal{P}}}$ is the preferred answer set.*

The main point of this corollary is that for a prioritized elp $\mathcal{P}$ with negative-cycle-free, head-consistent program, the computation of the sequence $X_0, \ldots, X_n$ of Definition 12 is guaranteed to produce an answer set, making the check stipulated in Definition 13 unnecessary.

In [Brewka and Eiter, 1999] one can find multiple examples illustrating how the BE-preferred answer set semantics overcomes some of the shortcomings of previous approaches. Based on their study of these shortcomings and the development of their semantics, Brewka and Eiter proposed two principles that ought to be satisfied by any system based on prioritized defeasible rules. We reproduced them here in their particular form for prioritized elps.

The first principle is a postulate meant as a minimal requirement on the treatment of $<$ as a preference relation.

Principle 1. Let A_1, A_2 be two answer sets of a prioritized elp $\mathcal{P} = (P, <)$ generated by the rules $R \cup \{r_1\}$ and $R \cup \{r_2\}$, respectively, where $r_1, r_2 \notin R$. If $r_1 < r_2$ then A_2 is not a preferred answer set of $\mathcal{P}$.

The second principle is about relevance. It says that a preferred answer set A should not become non-preferred after adding a rule with a prerequisite not in A while keeping all preferences intact.

Principle 2. Let A be a preferred answer set of $(P, <)$ and r be a rule such that at least one prerequisite of r is not in A. Then A is a preferred answer set of $(P \cup \{r\}, <')$ whenever $<'$ agrees with $<$ on the rules in P.

Let us show that our proposed semantics satisfies the first principle.

Theorem 3. *The preferred answer set semantics based on the computation of the set $I_{\overline{\mathcal{P}}}$ satisfies Principle 1.*

Proof. Let A_1, A_2 be answer sets of a fully prioritized, unfolded $\overline{\mathcal{P}} = (P, <)$ generated by $R \cup \{r_1\}$ and $R \cup \{r_2\}$, respectively, where $r_1, r_2 \notin R$ and $r_1 < r_2$. Since r_1, r_2 are unfolded and satisfied by A_1, A_2, we have $\mathit{body}^+(r_1) = \mathit{body}^+(r_2) = \emptyset$. Assume that A_2 is a preferred answer set of $\overline{\mathcal{P}}$. Since r_1 is not a generating rule of A_2, r_1 is defeated in A_2. Then there exists $b \in \mathit{body}^-(r_1)$ s.t. $b \in A_2$. Hence there is a rule $r' \in P$ s.t. $b = \mathit{head}(r')$. There are two cases: a) $r' \in R$. Then $b \in A_1$ and r_1 is defeated in A_1. Contradiction. b) $r' \notin R$. Then r' must be r_2. Since $r_1 < r_2$, it must be the case that $\mathit{body}(r_2)$ holds in A_2 (by case 1 of Definition 12). But this implies that $\mathit{body}(r_2)$ holds in A_1 and hence that $b \in A_1$ and that r_1 is defeated in A_1. Contradiction. Therefore A_2 is not a preferred answer set. □

Principle 2, which is about relevance, is not satisfied by our semantics. We use the program from Example 3 to make some observations.

Example 8. Consider again the program $\mathcal{P}$ from Example 3:

$$\begin{array}{rrcl} r_1: & b & \leftarrow & \mathit{not}\,\neg b,\ a. \\ r_2: & c & \leftarrow & \mathit{not}\,b. \\ r_3: & a & \leftarrow & \mathit{not}\,c. \end{array}$$

Consider the program $\mathcal{P}'$ consisting only of r_2, r_3 with $r_2 < r_3$. This program has one answer set $A_1 = \{c\}$ which is also preferred according to our semantics. The full program $\mathcal{P}$ has two answer sets, A_1 and the now preferred $A_2 = \{a, b\}$. $\mathcal{P}'$ has no BE-preferred answer sets while for $\mathcal{P}$ both A_1, A_2 are BE-preferred.

According to Principle 2, $\{c\}$ should remain a preferred answer set because r_1 is not applicable in A_1 (not relevant). But in terms of attacks, r_1 seems relevant since it can attack r_2 and has higher priority. Moreover, if we replace r_1 with the unfolded version $b \leftarrow \mathit{not}\,\neg b,\ \mathit{not}\,c$, which results in an equivalent program, Principle 2 no longer says anything about it since it defines relevance in terms of prerequisites (literals in body^+). In other words, applying an unfolding operation allows switching from violating to satisfying Principle 2, even though unfolding has no effect on a program's semantics.

The example above also shows that satisfying Principle 2 necessarily requires that some programs have no preferred answer sets or to have multiple ones. In the above example, $\mathcal{P}'$ has one answer set, $\{c\}$, which is not the same as the intuitively preferred answer set of $\mathcal{P}$. But Principle 2 requires that if $\{c\}$ is a preferred answer set, it must remain one after adding r_1. For these reasons we believe that Principle 2 as stated is not entirely suitable.

It is worth mentioning that Brewka and Eiter [1999] define another semantics, called *weakly preferred semantics*, which assigns preferred answer sets to programs that do not have one under the BE semantics. However, this semantics is based on quantitative measures of relative satisfaction of the preferences, which is very different to the style of semantics we propose here and of the BE semantics. Moreover, the weakly preferred semantics does not satisfy either of the above principles.

6 Conclusions

We have proposed a new (selective) semantics for prioritized extended logic programs. In contrast to previously proposed semantics, ours selects at most one preferred answer set for all programs that are fully prioritized. Furthermore, for a large class of programs guaranteed to have an answer set, the existence of a preferred answer set is also guaranteed. We have also shown that our semantics captures the intended meaning of preferences as postulated by Principle 1 from [Brewka and Eiter, 1999].

Future work includes looking at whether the set of preferred answer sets of a program under our semantics is a subset of the BE-preferred answer sets when the latter exist. Another is to generalize the results to programs with variables.

References

[Aravindan and Dung, 1995] Chandrabose Aravindan and Phan Minh Dung. On the correctness of unfold/fold transformation of normal and extended logic programs. *Journal of Logic Programming*, 24(3):201–217, 1995.

[Baral, 2003] Chitta Baral. *Knowledge representation, reasoning and declarative problem solving*. Cambridge University Press, 2003.

[Brass and Dix, 1999] Stefan Brass and Jürgen Dix. Semantics of (disjunctive) logic programs based on partial evaluation. *Journal of Logic Programming*, 40(1):1–46, 1999.

[Brewka and Eiter, 1999] Gerhard Brewka and Thomas Eiter. Preferred answer sets for extended logic programs. *Artificial Intelligence*, 109(1–2):297–356, 1999.

[Delgrande *et al.*, 2000] James P. Delgrande, Torsten Schaub, and Hans Tompits. Logic programs with compiled preferences. In Werner Horn, editor, *Proceedings of the 14th European Conference on Artificial Intelligence (ECAI'00)*, pages 464–468, 2000.

[Dix, 1995] Jürgen Dix. A classification theory of semantics of normal logic programs: II. Weak properties. *Fundamenta Informaticae*, 22(3):257–288, 1995.

[Gelfond and Lifschitz, 1991] Michael Gelfond and Vladimir Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9(3-4):365–386, 1991.

[Gelfond and Son, 1997] Michael Gelfond and Tran Cao Son. Reasoning with prioritized defaults. In Jurgen Dix, Luis Moniz Pereira, and Teodor C. Przymusinski, editors, *Selected Papers of the Third International Workshop on Logic Programming and Knowledge Representation*, number 1471 in LNCS, pages 164–223. Springer, 1997.

[Gelfond, 2008] Michael Gelfond. Answer sets. In F. van Harmelen, V. Lifschitz, and B. Porter, editors, *Handbook of Knowledge Representation*, chapter 7, pages 285–316. Elsevier, 2008.

[Sakama and Inoue, 1996] Chiaki Sakama and Katsumi Inoue. Representing priorities in logic programs. In M. Maher, editor, *Proceedings of the Joint International Conference and Symposium on Logic Programming*, pages 82–96, 1996.

[Schaub and Wang, 2001] Torsten Schaub and Kewen Wang. A comparative study of logic programs with preference. In Bernhard Nebel, editor, *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI'01)*, pages 597–602, 2001.

[Wang *et al.*, 2000] Kewen Wang, Lizhu Zhou, and Fangzhen Lin. Alternating fixpoint theory for logic programs with priority. In John W. Lloyd, Veronica Dahl, Ulrich Furbach, Manfred Kerber, Kung-Kiu Lau, Catuscia Palamidessi, Luis M. Pereira, Yehoshua Sagiv, and Peter J. Stuckey, editors, *Proceedings of the First International Conference on Computational Logic*, number 1861 in LNCS, pages 164–178. Springer, 2000.

[Zhang and Foo, 1997] Yan Zhang and Norman Y. Foo. Answer sets for prioritized logic programs. In Jan Maluszynski, editor, *Proceedngs of the International Symposium on Logic Programming (ILPS'97)*, pages 69–83, 1997.