IJCAI-05 Workshop on Nonmonotonic Reasoning, Action, and Change
(NRAC’05)

Edinburgh, Scotland, August 1, 2005

Working Notes

Edited by

Leora Morgenstern and Maurice Pagnucco
Sixth Workshop on Nonmonotonic Reasoning, Action, and Change

Welcome to NRAC’05, the Sixth Workshop on Nonmonotonic Reasoning, Action, and Change. This year marks the tenth anniversary of the start of this highly popular workshop series, which has been held in conjunction with the International Joint Conference on Artificial Intelligence (IJCAI) since 1995. The workshop aims to bring together researchers who, using logic-based AI, are investigating how rational agents use commonsense methods to reason about their world and determine how their actions change their environment.

This volume contains papers selected for presentation at NRAC’05, to be held on August 1, 2005 in Edinburgh, Scotland.

June 2005

Leora Morgenstern and Maurice Pagnucco
Workshop Chairs
NRAC’05

Workshop Chairs

Leora Morgenstern
IBM T.J. Watson Research Center (USA)
Maurice Pagnucco
University of New South Wales (Australia)

Program Committee

Gerhard Brewka
University of Leipzig (Germany)
Norman Foo
University of New South Wales (Australia)
Michael Gelfond
Texas Tech University (USA)
Gerhard Lakemeyer
Aachen University of Technology (Germany)
Jérôme Lang
Université Paul Sabatier (France)
Vladimir Lifschitz
University of Texas at Austin (USA)
Fangzhen Lin
Hong Kong Univ. of Science & Tech (China)
Pavlos Peppas
Patras University (Greece)
Erik Sandewall
Linköping University (Sweden)
Michael Thielscher
Dresden University of Technology (Germany)
Mary-Anne Williams
University of Technology, Sydney (Australia)

Additional Referees

Thomas Meyer
Richard Watson
# Table of Contents

Decision Theoretic Contraction and Value Indeterminacy: Maximizing rather than Optimizing ................................................................. 1  
*Horacio Arló-Costa (Carnegie Mellon University)*

Planning with Programs that Sense ....................................................... 7  
*Jorge Baier and Sheila McIlraith (University of Toronto)*

Restrained Revision .............................................................................. 15  
*Richard Booth (Macquarie University), Samir Chopra (Brooklyn College, CUNY) and Thomas Meyer (National ICT Australia)*

Modeling the Role of (ab)normality in the Ascription of Causality Judgments by Agents ................................................................. 22  
*Didier Dubois and Henri Prade (Université Paul Sabatier)*

Cumulative Effects of Concurrent Actions on Numeric-Valued Fluents ... 28  
*Esra Erdem (Vienna University of Technology) and Alfredo Gabaldon (National ICT Australia)*

Actions as Special Cases (Preliminary Report) ......................................... 34  
*Selim T. Erdoğan and Vladimir Lifschitz (University of Texas at Austin)*

Negotiating Logic Programs .................................................................. 39  
*Norman Foo, Thomas Meyer (National ICT Australia), Yan Zhang and Dongmo Zhang (University of Western Sydney)*

Compiling Qualitative Preferences into Decision-Theoretic Golog Programs 45  
*Christian Fritz and Sheila McIlraith (University of Toronto)*

Action Progression and Revision in Multiagent Belief Structures ............ 53  
*Andreas Herzig, Jérôme Lang (Université Paul Sabatier) and Pierre Marquis (Université d’Artois)*

Using Ranking Functions to Determine Plausible Action Histories ......... 59  
*Aaron Hunter and James P. Delgrande (Simon Fraser University)*

Improving Recovery for Belief Bases ..................................................... 65  
*Frances L. Johnson and Stuart C. Shapiro (The State University of New York at Buffalo)*

Actions as a Basic Software Concept in the **Leonardo** Computation System 71  
*Erik Sandewall (Linköping University)*
<table>
<thead>
<tr>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action, Belief Change and the Frame Problem: A Fluent Calculus</td>
<td>78</td>
</tr>
<tr>
<td>Richard B. Scherl (Monmouth University)</td>
<td></td>
</tr>
<tr>
<td>Belief Change with Noisy Sensing and Introspection</td>
<td>84</td>
</tr>
<tr>
<td>Steven Shapiro (University of Leipzig)</td>
<td></td>
</tr>
<tr>
<td><strong>Special Theme: Wumpus World</strong></td>
<td></td>
</tr>
<tr>
<td>The Wumpus World in IndiGolog: A Preliminary Report</td>
<td>90</td>
</tr>
<tr>
<td>Sebastian Sardina, Stavros Vassos (University of Toronto)</td>
<td></td>
</tr>
<tr>
<td>A SNePS Approach to The Wumpus World Agent or Cassie Meets the Wumpus</td>
<td>96</td>
</tr>
<tr>
<td>Stuart C. Shapiro and Michael Kandefer (The State University of New York)</td>
<td></td>
</tr>
<tr>
<td>A FLUX Agent for the Wumpus World</td>
<td>104</td>
</tr>
<tr>
<td>Michael Thielescher (Dresden University of Technology)</td>
<td></td>
</tr>
</tbody>
</table>
Schedule

8:20 - 8:25  **Opening**

**Session 1: Reasoning About Actions and Change**

8:25 – 8:50 Richard B. Scherl
Action, Belief Change and the Frame Problem: A Fluent Calculus Approach

8:50 – 9:15 Steven Shapiro
Belief Change with Noisy Sensing and Introspection

9:15 – 9:40 Aaron Hunter and James P. Delgrande
Using Ranking Functions to Determine Plausible Action Histories

9:40 – 10:05 Jorge Baier and Sheila McIlraith
Planning with programs that sense

10:05 – 10:30 Christian Fritz and Sheila McIlraith
Compiling Qualitative Preferences into Decision-Theoretic Golog Programs

10:30 – 11:00 **Coffee/Tea Break**

**Session 2: Belief Change**

11:00 – 11:25 Richard Booth, Samir Chopra and Thomas Meyer
Restrained Revision

11:25 – 11:50 Frances L. Johnson and Stuart C. Shapiro
Improving Recovery for Belief Bases

11:50 – 12:15 Horacio Arló-Costa
Decision Theoretic Contraction and Value Indeterminacy

12:15 – 12:40 Didier Dubois and Henri Prade
Modeling the Role of (ab)normality in the Ascription of Causality Judgments by Agents

12:40 – 14:00 **Lunch**

**Session 3: Special theme panel: Wumpus World**

14:00 – 15:00 Special theme panel: Wumpus World

  Sebastian Sardina and Stavros Vassos
  The Wumpus World in IndiGolog: A Preliminary Report

  Stuart C. Shapiro and Michael Kandefer
  A SNePS Approach to The Wumpus World Agent or Cassie Meets the Wumpus

  Michael Thielscher
  A FLUX Agent for the Wumpus World
Session 4: Reasoning About Actions

15:00 – 15:30 Coffee/Tea Break

15:30 – 15:55 Erik Sandewall
Actions as a Basic Software Concept in the Leonardo Computation System

15:55 – 16:20 Andreas Herzig, Jérôme Lang and Pierre Marquis
Action progression and revision in multiagent belief structures

16:20 – 16:45 Esra Erdem and Alfredo Gabaldon
Cumulative Effects of Concurrent Actions on Numeric-Valued Fluents

16:45 – 17:10 Selim Erdoğan and Vladimir Lifschitz
Actions as Special Cases (Preliminary Report)

17:10 – 17:35 Norman Foo, Thomas Meyer, Yan Zhang and Dongmo Zhang
Negotiating Logic Programs

17:35 – 17:40 Closing
Decision Theoretic Contraction and Value Indeterminacy: Maximizing rather than Optimizing

Horacio Arló Costa
Carnegie Mellon University
Philosophy
161B Baker Hall, Pittsburgh, PA 15231
hcosta@andrew.cmu.edu

Abstract

When indexes of information value are indeterminate or vague the usual techniques of optimization cannot be applied in order to compute optimal contractions. But in cases of this sort it is still rational to deem an option as choosable when it is not known to be worse that any other. Recent advances in the study of binary relations and choice functions permit to tackle this maximization problem with new tools [Banerjee and Pattanaik, 1996], [Suzumura and Xu, 2003], [Sen, 1997]. When only ordinal information is available, maximizing solutions yield options that are recommended by robust decision theories (or by Levi’s decision-theoretic account of contraction). Moreover bounded methods of choice, like Simon’s satisficing can also be seen as forms of maximization. So the application of maximization methods to study contraction is both normatively more adequate as well as descriptively more realistic.

1 Introduction

The classical framework of optimization used in standard choice theory recommends choosing, among the feasible options, a best alternative. So, if \( S \) is the feasible set and \( R \) is a weak preference relation over \( S \) optimization recommends focusing on the following set of maximal best elements of \( S \):

\[
G(S, R) = \{ Y \in S: \text{ for all } Z \in S, ZRY \}
\]

But many economists have recently pointed out that this stringent form of maximization might not be the kind of maximization that one can apply in practical problems where information is usually incomplete and sometimes scarce. For example the Nobel-winner Amartya Sen has recently pointed out that:

The general discipline of maximization differs from the special case of optimization in taking an alternative as choosable when it is not known to be worse that any other. [...] The basic contrast between maximization and optimization arises from the possibility that the preference ranking may be incomplete ([Sen, 1997], section 5).

To define a maximal set we can use the asymmetric part \( P \) of the weak ordering \( R \) as follows:

\[
M(S, R) = \{ Y \in S: \text{ for no } Z \in S, ZPY \}
\]

It is easy to see that in general (for any binary relation \( R \) and any non-empty feasible set \( S \)) we have that \( G(S, R) \subseteq M(S, P) \), where the equality holds in case that \( R \) is complete. Moreover a maximal set \( M(S, P) \) can always be replicated by optimizing a complete relation \( R+ \) obtained from \( P \) by transforming incomparabilities into indifferences. Obviously this new relation \( R+ \) has to be complete but it might fail to be transitive. In addition, although this new relation can mimic the maximizing behavior of \( P \) it is clear that it should not be used for representing knowledge. Sen warns against using this kind of relations in representing economic knowledge in particular, but it is clear that the problem is more general.

With the sole exception perhaps of the writings of Isaac Levi, the discipline of belief change has been dominated by the use of optimization techniques. This is particularly transparent in the recent work of Hans Rott who has shown how some of the central axioms of contraction arise from constraints in G-functions.

In order to study the contraction of a theory \( K \) with a sentence \( A \) the so-called AGM framework [Carlos Alchourron and Makinson, 1985] has proposed to focus on the set \( (K \downarrow A) \) of maximal subsets of \( K \) that fail to entail \( A \) as the feasible set from which one make choices. Then the idea is to utilize a selection function \( \gamma \) that when applied to \( (K \downarrow A) \) selects a non-empty subset of \( (K \downarrow A) \). In particular partial meet contraction focuses on selection functions that are relational, i.e. selection functions for which there is a binary relation \( \leq \) such that:

\[
\gamma(K \downarrow A, \leq) = \{ Y \in K \downarrow A: \text{ for all } Z \in K \downarrow A, Z \leq Y \}
\]
then \( K \div A \) is defined as the intersection of the elements of \( \gamma(K \downarrow A, \leq) \). Obviously this definition relies on a process of optimization of the sort discussed above. Two main criticisms can be raised against this way of articulating contraction. One concerns the feasible set, which many see as too restrictive. We will consider this problem in the final sections of this article. The second criticism concerns the use of optimization techniques. In many applications one might not have access to the binary relation needed to optimize, a relation that imposes strong demands, like completeness. In particular one might face cases of indeterminacy, which can be caused, for example, by lack of information or, alternatively, by the existence of conflicting standards of valuation.

Situations of this sort can be modeled by considering a set of binary relations that the agent considers permissible. Some researchers have proposed therefore that contractions are not functions but relations. This seems an extreme solution that does not seem to capture the fact that in contracting one does not merely want counseling about possible options but a concrete output.

When an agent faces indeterminacy between a set of permissible orderings one standard solution is to consider the compromises between them, that is the categorical relation obtained by considering all ordered pairs shared by all the permissible orderings. This seems a sensible solution but, of course, the resulting categorical preference need not be complete. It has to be a quasi-ordering, but not necessarily complete. So, optimizing this relation might not be possible. But, of course, one can maximize the resulting quasi-ordering.

Very recent advances on the study of binary relations and choice functions permit to tackle this maximization problem with new tools. In particular Banerjee and Pattanaik [Banerjee and Pattanaik, 1996] recently showed that the maximal set with respect to a quasi ordering can be fully recovered by defining the greatest sets with respect to each and every ordering extension thereof and taking their union. Even more recently Suzumura and Xu [Suzumura and Xu, 2003] extended this result by relaxing the axiom of transitivity to the axiom that Suzumura calls consistency [Suzumura, 1983]. These results permit to study contraction in the more general and liberal framework of maximization rather than optimization. This abstract focuses on presenting the bases for this study. The resulting theory is not only theoretically more adequate, but it is also considerably more realistic in considering that the sources of valuation guiding contraction might be indeterminate or vague.

2 Recoverability of choice functions and binary relations

Let \( P(R) \) denote the asymmetric part of a binary relation \( R \). Given a binary relation \( R \) on a menu \( X \) we call a relation \( Q \) on \( X \) satisfying \( Q \subset R, P(Q) \subset P(R) \) and \( Q \neq R \) a strict sub-relation of \( R \) and \( R \) is then called a strict extension of \( Q \).

For a given binary relation \( Q \) on a menu \( X \), let \( \Xi(Q) \) be the set of all strict ordering extensions of \( Q \).

**DEFINITION 2.1** Let \( Q \) be a binary relation on \( X \). \( Q \) is choice-functionally recoverable if and only if \( M(S, Q) = \bigcup_{R \in \Xi(Q)} G(S, R) \) holds for all \( S \in X \).

The following theorem, due to Banerjee and Pattanaik, is the first important result in the theory of recoverability functions:

**Theorem 2.1** A quasi-ordering \( Q \) is choice-functionally recoverable if and only if \( \Sigma(Q) \neq \emptyset \).

Suzumura and Xu have recently generalized this result, by making it applicable to a certain class of reflexive and consistent binary relations. They focus on relations obeying:

**DEFINITION 2.2** (Transitive Closure) Let a binary relation \( Q \) on \( X \) be given. For all \( x, y \in X \), if \( (x, y) \in P(T(Q)) \) holds, then \( (x, y) \in Q \), where \( T(Q) \) is the transitive closure of the binary relation \( Q \).

With the help of this definition Suzumura and Xu establish the following generalization of the result by Banerjee and Pattanaik:

**Theorem 2.2** A reflexive and consistent \( Q \) is choice-functionally recoverable if and only if \( \Sigma(Q) \neq \emptyset \) and it obeys the Transitive Closure condition.

When the menu \( X \) is finite Suzumura and Xu present the following important result, which is established via a constructive method and which relates to well-known results in the literature of rational choice and revealed preference:

**Theorem 2.3** Let \( X \) be finite. Assume that \( \Sigma(Q) \neq \emptyset \) and that the Transitive Closure condition hold. Then the \( Q \)-maximal sets satisfy the following conditions:

1. For all \( S, T \in X \), if \( S \subset T \), then \( S \cap M(T, Q) \subset M(S, Q) \).
2. For all \( S, T \in X \), if \( S \subset T \), and \( M(T, Q) \subset S \), then \( M(S, Q) \subset M(T, Q) \).

Suzumura and Xu’s results can be extended as follows:

**Lemma 2.1** Let \( X \) be finite. Assume that \( \Sigma(Q) \neq \emptyset \) and that the Transitive Closure condition hold. Then the \( Q \)-maximal sets satisfy:

1. For all \( S, T \in X \), such that \( S \cup T \in X \), then \( M(S, Q) \cap M(T, Q) \subset M(T \cup S, Q) \).
2. For all \( S, T \in X \), such that \( S \cup T \in X \), if \( M(S \cup T, Q) \cap T = \emptyset \), then \( M(S, Q) \subset M(T \cup S, Q) \).

**Sen’s \( \gamma \)** follows immediately from Aizerman and Sen’s \( \gamma \) can be established independently. On the other hand it is also useful to see that some important properties of choice functions are not satisfied in this setting. This is particularly important in the case of Arrow’s axiom:

**Arrow’s Axiom** For all \( S, T \in X \), such that \( S \cup T \in X \), if \( M(S \cup T, Q) \cap S = \emptyset \), then \( M(S, Q) \subset M(T \cup S, Q) \).

3 Maximizing categorical preference

We will assume a classical propositional language \( L \) as a representational tool. We assume that \( L \) contains the classical connectives. The underlying logic will be identified with its Tarskian consequence operator \( Cn_2 : 2^L \rightarrow 2^L \) which is assumed to obey for all subsets \( X \) and \( Y \) of \( L \):

**Inclusion** \( X \subset Cn(X) \)

**Monotony** If \( X \subset Y \), then \( Cn(X) \subset Cn(Y) \)
Some basic properties of ‘remainder sets’ of the form $(K \perp A)$ are useful at this juncture. In particular $K \perp (A \land B) = (K \perp A) \cup (K \perp B)$, for $A, B \in K - Cn(\emptyset)$. In addition it is useful to notice that for $A, B \in K - Cn(\emptyset)$ we have $M((K \perp A), <_c) \cap (K \perp B) = \emptyset$ if and only if $B \in \bigcap M((K \perp A), <_c)$. These are properties of remainder sets that do not depend on maximizing or optimizing.

Now the methods of proof used in [Rott, 1993] can be easily adapted to establish the following results involving how to contract conjunctions and other Boolean compounds. We assume in all results that the choice menu is finite:

$(\ast 7)$ $K \div A \cap K \div B \subseteq K \div (A \land B)$ [conjunctive overlap]

The completion $M^*$ of a choice function $M((K \perp A), <_c)$ is defined as $M^* = \{(K \perp A), <_c\} = \{M \in (K \perp A) : \bigcap M((K \perp A), <_c) \subseteq M\}$. A choice function is complete if and only if $M = M^*$.

**Lemma 3.1** If $\div$ is defined via $(C)$ for some choice function $M((K \perp A), <_c)$, then if $M$ satisfies Chernoff (Sen’s $\alpha$) satisfies postulates $(\ast 1)$ to $(\ast 7)$. Moreover if $\div$ satisfies postulates $(\ast 1)$ to $(\ast 7)$ and the choice function that determines $\div$ is complete, then $M$ satisfies Chernoff.

Now we can focus on some properties first analyzed in [Rott, 1993]. These properties have not been usually considered in the theory of belief change. They were first considered because of their obvious formal connections with well-known postulates used in the theory of choice functions.

$(\ast 8r)$ $K \div (A \land B) \subseteq Cn(K \div A \cup K \div B)$

The ‘r’ used in the name of this property stands for ‘relational’. The analysis that follows shows the relevance of the name.

**Lemma 3.2** If $\div$ is defined via $(C)$ for some complete choice function $M((K \perp A), <_c)$, $M$ satisfies Sen’s $\gamma$ if and only if $\div$ satisfies postulates $(\ast 1)$ to $(\ast 6)$ and $(\ast 8r)$.

The next property was also considered in [Rott, 1993]. The ‘c’ in its name stands for ‘cumulativity’ (the name should be obvious for readers familiar with the literature on non-monotonic logic).

$(\ast 8c)$ If $B \in K \div (A \land B)$, then $K \div (A \land B) \subseteq K \div A$.

**Lemma 3.3** If $\div$ is defined via $(C)$ for some choice function $M((K \perp A), <_c)$, then if $M$ satisfies Sen’s $\epsilon$ satisfies postulates $(\ast 1)$ to $(\ast 6)$ and $(\ast 8c)$. Moreover if $\div$ satisfies postulates $(\ast 1)$ to $(\ast 6)$ and $(\ast 8c)$ and the choice function that determines $\div$ is complete, then $M$ satisfies Sen’s $\epsilon$.

The choice-theoretical model proposed by Rott in [Rott, 1993], which depends on optimization, also considers the following important axiom of AGM:

$(\ast 8)$ If $A \notin K \div (A \land B)$, then $K \div (A \land B) \subseteq K \div A$.

This postulate naturally corresponds to the so-called Arrow Axiom (or Sen’s property $\beta^*$). But we have seen above that Arrow’s Axiom does not hold for the maximizing choice functions we are considering here. So, the postulate does not hold either.

It is important to notice that Azizerman is stronger than most of the aforementioned conditions (or is independent of them).
So, we have the following additional postulate not previously considered in the literature:

\((\star 8a)\) If \(A \notin K \land (A \land B)\), then \(K \div (A \land B) \subseteq K \div A\).

**Lemma 3.4** If \(\div\) is defined via \((C)\) for some choice function \(M((K \land A), <_c)\), then if \(M\) satisfies the Aizerman property then \((\star 8a)\) is satisfied.

**Proof 3.1** The above proposition states Aizerman's useful here; 
If \(M((S \cup T), <_c) \subseteq S\), then \(M(S, <_c) \subseteq M((S \cup T), <_c)\). In virtue of aforementioned properties of remainder sets we also have therefore that: If \(M((K \land (A \land B)), <_c) \subseteq K \land A\), then \(M((K \land A) \subseteq M((K \land (A \land B)))\). The lemma follows then immediately from this instance of Aizerman.

The resulting theory of contraction is obviously weaker than the AGM theory. The eight AGM axiom has been criticized by many authors before the use of intuitive examples or by pointing out that it imposes unreasonable requirements on modeling contraction (John Pollock provided similar examples in conditional logic). It is interesting to see that the axiom is in principle not valid when we want to maximize rather than optimizes.

We can call the contraction functions obeying \((\star 1)\) to \((\star 7)\), plus \((\star 8r)\), \((\star 8c)\), and \((\star 8a)\), liberal contraction, given that it has been obtained by employing a process of liberal maximization as opposed to the stringent process called optimization – \((\star 8c)\) is redundant in this presentation of liberal contraction.

### 4 Optimizing models for liberal contraction and binariness

It is important to recall here that every maximizing process can always be replicated by optimizing the augmentation of the maximized relation.

**Example:** Consider the following two initial rankings: \(\leq_1\) and \(\leq_2\) over the feasible set \(S = \{a, b, c\}\). \(\leq_1: (a, a), (b, b), (c, c), (c, b), (b, a), \leq_2: (a, a), (b, b), (c, c), (a, c), (b, a).\) The categorical preference for these rankings is \(<_{c}: (a, a), (b, b), (c, c), (b, a)\).

Let’s now consider the augmentation of \(<_{c},<^+_c: (a, a), (b, b), (c, b), (b, a), (c, a), (c, a), (c, b), (b, c)\). Obviously \(<^+_c\) is complete but it is not transitive. But the augmented relation can mimic the process of maximizing \(<_c\). In fact, we have that \(M(S, <_c) = G(S_{<^+_c}) = \{a, c\}\).

The example shows that the augmented relations used in order to replicate maximizing process tend not to be transitive. So, we must expect that an optimizing model of liberal contraction cannot require the transitivity of the binary relation used in the optimizing model. This is indeed the case given that \((\star 8)\) plays a crucial role in establishing representation results of contraction functions requiring the transitivity of the optimizing relation used to define contraction.

But some of the results proved by Rott characterizing contractions that obey \((\star 1)\) to \((\star 7)\), plus \((\star 8r)\), \((\star 8c)\), do apply to liberal contraction. According to these results we should expect that the optimizing relation used to characterize liberal contraction should be at least negatively transitive relational, and therefore relational.

Moreover the choice functions used in order to characterize liberal contraction are binary and menu independent. The terminology is Sen’s and the properties in question are quite important in order to understand the behavior of a choice function in general. Let \(Q\) be a categorical preference, which we can assume is a quasi ordering or the asymmetric part of a reflexive and consistent preference relation obeying the Transitive Closure condition.

**(Revealed Preference)** \(x_{R,y}\) if and only if \(\exists S: \{x \in M(S, Q) and y \in S\}\)

**(Binariness of Choice)** \(M(S, Q) = G(S, R_y)\)

The original choice function \(M(S, Q)\) is binary if and only if the revealed preference relation \(R_y\) generated by the choice function, if used as the basis of choice, will, in turn, regenerate the choice function itself.

Amartya Sen proved that a choice function is binary if and only if Properties \(\alpha\) (Chernoff) and Property \(\gamma\) hold. Since our choice functions obey these properties they are binary and therefore menu independent. Nevertheless, it is easy to see that the revealed preference \(R_y\) extracted by Revealed Preference is not the categorical preference \(Q\) whose maximization yield the choice functions in the first place. The revealed preference \(R_y\) is actually the augmentation of \(Q, Q^+\). Again this relation has little epistemological interest aside from mimicking the behavior of the original \(Q\).

**(Strong Revealed Preference)** \(x_{P,y}\) if and only if \(\exists S: \{x \in M(S, Q) and y \in S \setminus M(S, Q)\}\)

**(Weak Binariness of Choice)** \(M(S, Q) = M(S, P_c)\)

This strengthening of Revealed Preference has the capacity of re-generating the categorical preference that actually generated the \(M\) function to begin with. This notion of binariness holds for choice functions that obey Sen’s \(\alpha\), and it is more adequate for our present purposes.

### 5 Feasibility and Cognitive Choice

In all the previous sections we have assumed that in order to determine the content of a contraction \(K \div A\) it is adequate to maximize over a feasible set determined by the contents of the remainder set \((K \land A)\). But this is a controversial issue in the foundations of belief change. Isaac Levi has proposed in various writings that one should focus instead on the larger family of saturatable contractions removing \(A\).

**Definition 5.1** Let \(S(K, A)\) be the family of feasible sets of \(K\). I.e. if \(K\) is a theory, \(X \in S(K, A)\) if and only if \(X \subseteq K\), \(X\) is closed, and \(Cn(X \cup \{\neg A\})\) is a maximal and consistent set.

So, the proposal in many of the previous articulations on the so-called ‘Levi contractions’ [Levi, 1991] is to widen the scope of the choice function used in order to define contraction. The idea is that these choice functions take saturatable families as arguments (we follow in this abstract the presentation of Levi contractions presented in [Hansson, 1999] - a
more liberal presentation can be found in [Arló-Costa and Levi, 2005]). These choice functions should be such that when applied to a family $S(K, A)$, return a non-empty subset of $S(K, A)$.

**DEFINITION 5.2** $\vdash$ is a Levi-contraction of a theory $K$ if and only if there exists a choice function $G$ for $K$ such that for all sentences $A$: if $A \in K$, then $K \vdash A = \bigcap G(S(K, A))$, and if $A \notin K$, $K \vdash A = K$.

A contraction operator of this kind does not obey the controversial postulate of Recovery, presented above. Unlike other presentations of contraction this kind of contraction is decision theoretically motivated, and it is usually complemented by the explicit introduction of a value function $V$ on the set of logically closed subsets of a theory of reference $K$. If we have a set of value functions (due either to indeterminacy or to the presence of various sources of epistemic value), each one of them will induce an ordering $\leq$. Moreover if each value function is weakly monotonic we have that categorial preference should satisfy:

(Weak Monotony) For any two sets $X, Y$ in the range of $V$, such that $X \subseteq Y$, $V(X) \leq V(Y)$.

A more robust notion of contraction can be then introduced by further constraining the choice function $G$ in such a way that it optimizes the underlying value function:

$$G(S(K, A)) = \{X \in S(K, A) : V(X) \leq V(Y) \text{ for all } Y \in S(K, A)\}$$

The corresponding notion of contraction can be further enriched by adding constraints on the function $V$ reflecting the idea that the index that has to be minimized in contraction is the loss of information value (as opposed to just information loss). In [Arló-Costa and Levi, 2005] a complete representation for this notion of contraction is presented. But this is not the object of the present study. Indeterminacy can arise for this notion of contraction in the same way in which it can arise for the notion of contraction studied previously (AGM contraction). When this happens the agent might have at his disposal only the ordinalized version of a set of value functions. As before he can extract a categorial preference from it and proceed in the same way that we proceeded before (this issue is considered in passing in [Levi, 2004] - we are exploring here one of the options suggested in the book).

Much of the structural results depending on the work of Suzumura and Xu continue to hold here as well. Nevertheless the logic of saturatable families differs from the logic of remainder sets, and this generates some differences.

The first important difference is that we only have that: $S(K, (A \land B)) \subseteq S(K, A) \cup S(K, B)$. But the converse does not necessarily hold. This and other logical asymmetries put constraints on immediate applications of the results on liberal contraction presented in previous sections. Some results hold nevertheless in spite of these asymmetries. We need first some definitions constraining the notion of categorial preference:

(WMc) For any two sets $X, Y$ in the power set of $S(K, A)$, such that $X \subseteq Y$, $X \leq_c Y$.

**DEFINITION 5.3** $\vdash$ is a liberal contraction of a theory $K$ if and only if there exists a choice function $M$ for $K$ such that for all sentences $A$: if $A \in K$, then $K \vdash A = \bigcap M(S(K, A), \leq_c)$, and if $A \notin K$, $K \vdash A = K$. The notion $\leq_c$ of categorical preference is assumed here to be a quasi-ordering.

Here is a previous lemma useful in some of the proofs that follow:

**Lemma 5.1** Let $M$ be a maximizing function for $\leq_c$ obeying WMc. If $Z \in M(S(K, A), \leq_c)$, and $Z \subseteq Z' \in K \land A$, then $Z' \in M(S(K, A), \leq_c)$.

**Proof 5.1** Assume that $Z \in M(S(K, A), \leq_c)$. Then by the so-called upper-bound property (see [Carlos Alchourrón and Makinson, 1985]) there is $Z'$ extending $Z$ such that $Z' \in K \land A$. By WMc we have that $Z \leq_c Z'$. Now, since $Z \in M(S(K, A), \leq_c)$, and $Z' \in S(K, A)$ we have that it is not the case that $Z <_c Z'$. Therefore $Z$ and $Z'$ are equi-preferred.

**Lemma 5.2** Let $\vdash$ a notion of liberal contraction on a theory $K$. Then $\vdash$ satisfies ($\vdash 7$).

**Proof 5.2** The non-trivial case is when both $A, B$ are non-tautological members of $K$. Assume then that $D \in K \vdash A \land B$. Take now an arbitrary member $M$ of $M(S(K, (A \land B)), \leq_c)$. It follows by well-known properties of saturatable sets that either $Y \in S(K, A)$, or $Y \in S(K, B)$. Wlog assume that $Y \in S(K, A)$. We will show also that $Y \in M(S(K, A), \leq_c)$. Since we assumed that $D \in K \vdash A \land K \vdash B$, it follows that if $Y \in M(S(K, A), \leq_c)$, or $Y \in M(S(K, B), \leq_c)$, then $D \in Y$, and we would be done.

Since we assumed that $Y \in S(K, A)$ this means that the upper-bound property guarantees that there is $Y'$ extending $Y$ and belonging to $K \land A$. Now, since $K \land (A \land B) = (K \land A) \cup (K \land B)$, for $A, B \in K - Cn(\emptyset)$, we have also that $Y' \in K \land (A \land B)$. This, in turn, guarantees that $Y' \in M(S(K, (A \land B)), \leq_c)$.

We need to establish now that $Y \in M(S(K, A), \leq_c)$. Assume by contradiction that there is $Z \in S(K, A)$ such that $Y <_c Z$. By the upper bound property we also have that there is $Z'$ extending $Z$ and belonging to $K \land A$. Moreover, as before this means that $Z' \in K \land (A \land B) \subseteq S(K, (A \land B))$. But then the transitivity of categorial preference yields that $Y' <_c Z'$, which is impossible because $Y' \in M(S(K, (A \land B)), \leq_c)$.

We can turn now to the postulates ($\vdash 8c$), ($\vdash 8c$), and ($\vdash 8r$).

**Lemma 5.3** Let $\vdash$ a notion of liberal contraction on a theory $K$. Then $\vdash$ satisfies ($\vdash 8r$).

**Proof 5.3** The non-trivial case is when both $A, B$ are non-tautological members of $K$. Assume then that $D \in \cap M(S(K, (A \land B)), \leq_c)$. To prove the lemma is enough to prove that: $M(S(K, (A \land B)), \leq_c) \subseteq M(S(K, A), \leq_c) \cup M(S(K, B), \leq_c)$, for $A, B \in K - Cn(\emptyset)$.

Assume that $Y \in M(S(K, (A \land B)), \leq_c)$. Assume also by contradiction that $Y \notin M(S(K, A), \leq_c)$, and $Y \notin M(S(K, B), \leq_c)$. We then conclude by well-known properties of...
M(S(K, B), ≤c). A contradiction follows in each case. For if \( Y \notin M(S(K, A), ≤c) \) there is \( R \in S(K, A) \) and \( Y <c R \). Now, we also know that in this case the upper bound property guarantees that there is \( R' \) extending \( R \) and a member of \( K.A \). Moreover, as in the previous proof \( R' \in K, (A \land B) \subseteq S(K, (A \land B)) \). Transitivity of categorical preference and \( WMC \) yield that \( Y <c R' \) contradicting the fact that \( Y \in M(S(K, (A \land B)), ≤c) \).

It seems, nevertheless, that counterexamples can be established in the case of the other two postulates. We will consider briefly here (8c). Assume that \( B \in K \div (A \land B) \). In view of previous results we know also that if \( X \in M(S(K, (A \land B)), ≤c) \) then \( X \in M(S(K, A), ≤c) \). Assume that \( Y' \in M(S(K, A), ≤c) \). Either \( B \in Y \) or \( B \notin Y \). The problematic case occurs when \( B \notin Y \), in virtue of the upper-bound property we have that there is a set \( Y' \) extending \( Y \) such that \( Y' \in K \div A \) and therefore \( Y'' \in M(S(K, A)) \). Notice that \( B \notin Y'' \). And we also have that \( Y'' \in K \div (A \land B) \subseteq S(K, (A \land B)) \). In this situation we could have \( Y'' \in S(K, (A \land B)) \) such that \( Y' <c Y'' \). Notice that \( Y'' \) cannot belong to \( S(K, A) \), given that \( Y'' \in M(S(K, A)) \). But nothing precludes that \( Y'' \in S(K, B) \) with \( A \in Y'' \).

The former results depend on the fact that the categorical preference is a quasi ordering. In particular transitivity is used various times. It would be nice though to extend the previous results by only assuming the Transitive Closure condition of Suzumura and Xu. A completeness result seems to depend on a previous characterization of choice functions used to maximize quasi orderings and the weaker relations used by Suzumura and Xu.

6 Conclusion

The appeal to maximization has many applications beyond cases where there is genuine indeterminacy. Some bounded methods of choice, like Simon’s satisficing [Simon, 1982] can also be seen as forms of maximization (in spite superficial impressions to the contrary). Sen has argued forcefully for this view. The businessman who is willing to settle for \( x = 5 \) million without concerns about raising it to \( y = 1.01 \) million, regards both \( x \) and \( y \) as acceptable, but this does not mean that he sees as ‘equally good’. With respect to his welfare function the businessman might place \( y \) over \( x \). On the other hand, given the bounded nature of his choice behavior he is ready to settle for either \( x \) or \( y \). So, given his goals neither \( x \) is placed over \( y \) nor vice versa. Nor there is a decision to accept the options as equally good given the agent’s goals. This has lead Sen [Sen, 1997] to conclude that:

So, in terms of the goal function (as opposed to his welfare function) there is a ‘tentative incompleteness’ here and both \( x \) and \( y \) can be seen as ‘maximal’ in terms of the operational goals. Thus interpreted satisficing corresponds entirely to maximizing behavior.

Sen recognizes that in view of these arguments satisficing can also be seen as an as if optimizing exercise (by using the completed extension of the goal function). But he also correctly warns against this as if account:

But, as we discussed earlier, the use of this as if preference is interpretatively quite different. Thus the substantive gap between satisficing and optimizing remains (closable only in a purely formal way), whereas the gap between satisficing and maximizing is both formally and substantively absent.

The insistence on transitorily relational accounts of optimizing contraction in the AGM tradition can therefore be seen as a tacit dismissal of indeterminacy and therefore also of the use of bounded methods of choice of the sort that Simon has in mind. Transitivity can be retained by focusing on maximizing accounts of contraction based on quasi orderings, but in this case the as if account of these functions in terms of optimization needs to drop transitivity. The use of maximization techniques seems therefore more promising than its optimizing counterpart, both in order to accommodate bounded methods and to study the indeterminacy given by either ignorance or by the multiplicity of conflicting standards of valuation that might arise in cognitive decision problems.

References


Planning with programs that sense

Jorge Baier and Sheila McIlraith
Department of Computer Science
University of Toronto
{jabaij, sheila}@cs.toronto.edu

Abstract
In this paper we address the problem of planning by composing programs, rather than or in addition to primitive actions. The programs that form the building blocks of such plans can, themselves, contain both sensing and world-altering actions. Our work is primarily motivated by the problem of automated Web service composition, since Web services are programs that can sense and act. Our further motivation is to understand how to exploit macro-actions in existing operator-based planners that plan with sensing. We study this problem in the language of the situation calculus, appealing to Golog to represent our programs. To this end, we propose an offline execution semantics for Golog programs with sensing. We then propose a compilation method that transforms our action theory with programs into a new theory where programs are replaced by primitive actions. This enables us to use traditional operator-based planning techniques to plan with programs that sense for a restricted but compelling class of problems. We conclude by discussing the applicability of these results to existing operator-based planners that allow sensing.

1 Introduction
Classical planning takes an initial state, a goal state and an action theory as input and generates a sequence of actions that, when performed starting in the initial state, will terminate in a goal state. Typically, actions are primitive and are described in terms of their precondition, and (conditional) effects. Classical planning has been extended to planning with sensing actions. In most instances the planners are propositional and the generated plans are conditional. Our interest here is in using programs, rather than or in addition to primitive actions, as the building blocks for plans. The programs that we wish to consider may both sense and act in the world. We study this problem in the language of the situation calculus, appealing to Golog to represent our programs.

Our primary motivation for investigating this topic is to address the problem of automated Web service composition (e.g., [13]). Web services are self-contained Web-accessible computer programs, such as the airline ticket service at www.aircanada.com, or the weather service at www.weather.com. These services are indeed programs that sense and/or act in the world – e.g., determining flight costs or credit card approval, arranging for the delivery of goods and the debiting of accounts, etc. As such, the task of automated Web service composition (WSC) can be conceived as the task of planning with programs, or as a specialized version of a program synthesis task. While space precludes detailed discussion of the WSC task, this paper addresses some key remaining challenges to achieving it.

A secondary motivation for this work is to improve the efficiency of planning with sensing by representing useful (conditional) plan segments as programs. Though we do not study its effectiveness in this paper, planning with some form of macro-actions (e.g., [19; 9; 6; 14; 5]) can dramatically improve the efficiency of plan generation by reducing the search space size and the length of a plan. This is of particular importance in planning problems that involve sensing actions.

Levesque argued in [11] that when planning with sensing, the outcome of the planning process should be a plan which the executing agent knows at the outset will lead to a final situation in which the goal is satisfied. Even in cases where we assume no uncertainty in the outcome of actions, and no exogenous actions, this remains challenging because of incomplete information about the initial state. To plan effectively with programs, we must consider whether we have the knowledge to actually execute the program prior to using it in a plan. To that end, in Section 3 we propose an offline execution semantics for Golog programs with sensing that enables us to determine that we know how to execute a program. We prove the equivalence of our semantics to the original Golog semantics, under certain conditions. Then, in Section 4 we propose a compilation method that transforms our action theory with programs into a new theory where programs are replaced by primitive actions. This enables us to use traditional operator-based planning techniques to plan with programs that sense in a restricted but compelling set of cases. We conclude by briefly discussing the applicability of these results to existing operator-based planners that allows sensing.

2 Preliminaries
In the two subsections that follow we briefly review the situation calculus [18], including a treatment of sensing actions and knowledge [21]. We also review the transition seman-
tics for Golog, a high-level agent programming language that we employ to represent the programs we are composing. For those familiar with the situation calculus and Golog, we draw your attention to the decomposition of successor state axioms for the \( K \) fluent leading to Proposition 2.1 and the perhaps less familiar distinction of deterministic tree programs found in Section 2.2.

### 2.1 The situation calculus

The situation calculus [12; 18] is a second-order language for specifying and reasoning about dynamical systems. In the situation calculus, the world changes as the result of actions. A situation is a term denoting the history of actions performed from an initial distinguished situation, \( S_0 \). The function \( do(a, s) \) denotes the situation that results from performing action \( a \) in situation \( s \).

Relational fluents (resp. functional fluents) are situation-dependent predicates (resp. functions) that capture the changing state of the world. The distinguished predicate \( Poss(a, s) \) is used to express that it is possible to execute action \( a \) in situation \( s \). Following Scherl and Levesque [21], we use the distinguished fluent \( K \) to capture the knowledge of an agent in the situation calculus. The \( K \) fluent reflects a first-order abstraction of Moore’s possible-world semantics for knowledge and action [15]. \( K(s', s) \) holds iff when the agent is in situation \( s \), she considers it possible to be in \( s' \). Thus, we say that a first-order formula \( \phi \) is known in a situation \( s \) if \( \phi \) holds in every situation that is \( K \)-accessible from \( s \). For notational convenience, we adopt the abbreviations\(^2\) \( \text{Knows}(\phi, s) \equiv (\forall s').K(s', s) \supset \phi[s'] \), and \( KW\text{Whether}(\phi, s) \equiv \text{Knows}(\phi, s) \lor \text{Knows}(\neg \phi, s) \). To define properties of the knowledge of agents we can define restrictions over the \( K \) fluent. One common restriction is reflexivity (i.e., \((\forall s)K(s, s)\) which implies that everything that is known in \( s \) is also true in \( s \).

A situation calculus theory of action. \( D \) logically describes the dynamics of a domain. Following the axiomatization of [18], the theories of action we consider comprise at least the following:

- \( \Sigma \), a set of foundational axioms.
- \( D_s \), a set of successor state axioms (SSAs). The set of SSAs can be compiled from a set of effect axioms, \( D_{eff} \) [17]. An effect axiom describes the effect of an action on the truth value of certain fluents, e.g., \( a = \text{startCar} \supset \text{engineStarted}(do(a, s)) \).
- \( D_{ap} \), a set of action precondition axioms, one for each action. They are usually compiled from a set \( D_{nc} \) of necessary conditions on the fluent Poss, e.g., \( \text{Poss} \text{(startCar, s)} \supset \text{batteryOK}(s) \).
- \( D_{un} \), the set of unique names axioms for actions.
- \( D_{re} \), a set that describes the initial state of the world.
- \( D_{init} \), a set that defines the properties of the \( K \) fluent in the initial situations and preserved in all situations.
- \( D_{golog} \) a set of axioms for Golog’s semantics.

\( do\left([a_1, \ldots, a_n], s\right) \) abbreviates \( do(a_0, do(\ldots, do(a_1, s)\ldots)) \).

\( ^1 \)We assume \( \phi \) is a situation-suppressed formula (i.e. a situation calculus formula whose situation terms are suppressed). \( \phi[s] \) denotes the formula that restores situation arguments in \( \phi \) by \( s \).

Agents can gather information from the world using sensing actions. A sensing action results in the agent knowing whether a particular property of the world is true, or knowing the value of a particular term. In [21] sensing actions do not alter the state of the world. They only alter the agent’s state of knowledge. [21] introduces a standard SSA for the \( K \) fluent. Given sensing actions \( a_1, \ldots, a_n \) such that \( a_i \) (\( 1 \leq i \leq n \)) senses whether or not formula \( \psi_i \) is true, the SSA for \( K \) is:

\[
K(s', do(a, s)) \equiv (\exists s'').s'' = do(a, s') \land K(s'', s) \land \bigwedge_{i=1}^{n} \{ a = a_i \supset (\psi_i(s) \equiv \psi_i(s'')) \}. \quad (1)
\]

Intuitively, when performing a non-sensing action \( a \) in \( s \), if \( s'' \) was \( K \)-accessible from \( s \) then so is \( do(a, s'') \) from \( do(a, s) \). However, if sensing action \( a_i \) is performed in \( s \) and \( s'' \) was \( K \)-accessible from \( s \) then \( do(a_i, s'') \) is \( K \)-accessible from \( do(a_i, s) \) only if \( s \) and \( s'' \) agree upon the truth value of \( \psi_i \).

Since \( a_i \) is not world-altering this means that in all situations reachable from \( do(a_i, s) \) either \( \psi_i \) or \( \neg \psi_i \) holds, i.e. the agent knows whether \( \psi_i \) holds.

In contrast to [21], we assume that the SSA for \( K \) is compiled from a set of sufficient condition axioms, \( \Sigma_K \), rather than simply given. We do this to be able to cleanly modify the SSA for \( K \) without appealing to syntactic manipulations. To model an agent with sensing actions \( a_1, \ldots, a_n \) such that each action \( a_i \) formula \( \psi_i \), the axiomatizer must generate the following sufficient condition axioms for each \( a_i \):

\[
K(s'', s) \land a = a_i \land \Bigl( \psi_i(s) \equiv \psi_i(s'') \Bigr) \supset K(do(a, s''), do(a, s)), \quad (2)
\]

which intuitively express the same dynamics of the \( K \)-reachability for situations as (1) but with one axiom for each action. Furthermore, in order to model the dynamics of the \( K \)-reachability for the remaining non-sensing actions, the following axiom must be added:

\[
s' = do(a, s'') \land K(s'', s) \land \bigwedge_{i=1}^{n} \{ a \neq a_i \supset K(s', do(a, s)) \}. \quad (3)
\]

(2) and (3) can be shown to be equivalent to the SSA of \( K \) when one assumes that all necessary conditions are also sufficient.

**Proposition 2.1** Predicate completion on axioms of the form (2) and (3) is equivalent to the SSA for \( K \) defined in (1).

### 2.2 Golog’s syntax and semantics

Golog is a high-level agent programming language whose semantics is based on the situation calculus [18]. A Golog program is a complex action\(^3\) potentially composed from:

- \( \text{nil} \) – the empty program
- \( a \) – primitive action
- \( \phi? \) – test action
- \( \delta_1, \delta_2 \) – sequences
- \( \text{while } \phi \text{ do } \delta \text{ endW} \) – loop
- \( \text{if } \phi \text{ then } \delta_1 \text{ else } \delta_2 \) – conditional

\( ^3 \)Henceforth, we use the symbol \( \delta \) to denote complex actions. \( \phi \) is a situation-suppressed formula.
In Section 4.1 we will propose a compilation algorithm for Golog programs that are deterministic tree programs. A Golog tree program is one that does not contain loops. A Golog program is deterministic if it does not contain non-deterministic constructs. The restriction to tree programs may seem strong. Nevertheless, in practical applications most loops in terminating programs can be replaced by a bounded loop (i.e., a loop that is guaranteed to end after a certain number of iterations). Thus, following [14], we extend the Golog language with the bounded loop construct, while \( @ \) do \( \delta \) endW defined equal to if \( @ \) then \( \{ \delta; \text{while}_{k-1} \) do \( \delta \) endW \} else nil endif, for \( k > 0 \) and equal to nil if \( k = 0 \). We include this as an admissible construct for a tree program.

Golog has both an evaluation semantics [18] and a transition semantics [7]. The transition semantics is defined in terms of single steps of computation, using two predicates Trans and Final. \( \text{Trans}(\delta, s, \delta', s') \) is true iff when a single step of program \( \delta \) is executed in \( s \), it ends in the situation \( s' \), with program \( \delta' \) remaining to be executed, and \( \text{Final}(\delta, s) \) is true if program \( \delta \) terminates in \( s \). Using the transitive closure of Trans, \( \text{Trans}^* \), the predicate Do(\( \delta, s, s' \)) such that it is true iff program \( \delta \) terminates in situation \( s' \) if executed in situation \( s \). Some axioms for Trans and Final are shown below.

\[
\text{Trans}(a, s, \delta', s') \equiv \text{Poss}(a, s) \land \delta' = \text{do}(a, s) \\
\text{Trans}(\{ @ \text{then} \delta_1 \text{ else } \delta_2 \text{ endif} \}, s, \delta', s') \equiv \phi[s] \land \text{Trans}(1\delta_1, s, \delta', s') \lor \neg \phi[s] \land \text{Trans}(1\delta_2, s, \delta', s')
\]

3 A semantics for executable Golog programs

As Levesque [11] argued, when planning with sensing, the outcome of the planning process should be a plan which the executing agent knows at the outset will lead to a final situation in which the goal is satisfied. Even in cases where we assume no uncertainty in the outcome of actions, and no exogenous actions, this remains challenging because of incomplete information about the initial state. When planning with programs, as we are proposing here, the problem only gets worse. In particular, Golog’s existing semantics does not consider sensing actions and furthermore does not consider whether the agent has the ability to execute a given program. As a first step towards planning with programs that sense, we define a semantics for Golog that ensures that any Golog program with a terminating situation will also be executable by an agent. This semantics provides the foundation for results in subsequent sections.

Intuitively, we need to ensure that at each step of program execution, an agent has all the knowledge necessary to execute that step. In particular, we need to ensure that the program is epistemically feasible. Once we define the conditions under which a program is epistemically feasible, we can either use them as constraints on the planner, or we can ensure that our planner only builds plans using programs that are known to be epistemically feasible at the outset.

To our knowledge, no such semantics exists. Nevertheless, there is related work. In [8], the semantics of programs with sensing is defined in an online manner, i.e., it is determined during the execution of the program. An execution is formally defined as a mathematical object, and the semantics of the program depends on such an object. The semantics is thus defined in the metalanguage, and therefore it is not possible to refer to the situations that would result from the execution of a program within the language. Several papers have addressed the problem of knowing how to execute a plan [4] or more specifically, a Golog program. In [10], a predicate CanExec is defined to establish when a program can be executed by an agent. In [20], epistemically feasible programs are defined using the online semantics of [8]. Finally, a simple definition is given in [13], which defines a self-sufficient property, such that \( \text{ssf}(\delta, s) \) is true iff an agent knows how to execute program \( \delta \) in situation \( s \). Its definition is given below.

\[
\text{ssf}(\text{nil}) \equiv \text{True}, \\
\text{ssf}(a, s) \equiv \text{KWhether}(\text{Poss}(a, s)), \\
\text{ssf}(\pi x. \delta, s) \equiv (\exists x) \text{ssf}(\delta, x), \\
\text{ssf}(1\delta_1 \delta_2, s) \equiv \text{ssf}(1\delta_1, s) \land \text{ssf}(1\delta_2, s), \\
\text{ssf}(@?\delta, s) \equiv \text{KWhether}(\delta, s), \\
\text{ssf}(1\delta_1 1\delta_2, s) \equiv \text{ssf}(1\delta_1, s) \lor (1\delta_2) \land \text{Trans}^*(1\delta_1, s, \text{nil}, s') \subseteq \text{ssf}(1\delta_2, s'), \\
\text{ssf}(\{ @ \text{then} \delta_1 \text{ else } \delta_2 \text{ endif} \}, s) \equiv \text{KWhether}(\delta, s) \lor (\text{ssf}[s] \land \\
\text{Trans}^*(1\delta, s, \text{nil}, s') \subseteq \text{ssf}(\text{while } \delta \text{ do } \text{endW}, s)).
\]

To define a semantics for executable programs with sensing, we modify the existing Golog transition semantics so that it refers to the knowledge of the agent, defining two new predicates \( \text{Trans}_K \) and \( \text{Final}_K \). We conjecture that our proposed semantics is equivalent to that of [8] in an online setting. (We plan to prove this in future work.) The definitions of \( \text{Trans}_K \) and \( \text{Final}_K \) follow.

\[
\text{Final}_K(\delta, s) \equiv \text{Final}(\delta, s) \\
\text{Trans}_K(\text{nil}, s, \delta', s') \equiv \text{False} \\
\text{Trans}_K(1\delta_1 1\delta_2, s, \delta', s') \equiv \text{Knows}(1\delta_1, s) \land \delta' = \text{nil} \land s' = s \\
\text{Trans}_K(\text{nil}, s, \delta', s') \equiv \text{Knows}(\delta, s) \land \delta' = \text{do}(a, s) \\
\text{Trans}_K(\text{while } \delta \text{ do } \text{endW}, s, \delta', s') \equiv \text{Knows}(\delta, s) \land s = s' \land \\
\delta' = \text{nil} \lor \text{Knows}(\delta, s) \land \text{Trans}_K(\delta, s, \delta', s').
\]
Lemma 3.1 Let \( \mathcal{D} = \Sigma \cup \mathcal{D}_{\text{act}} \cup \mathcal{D}_{\text{ap}} \cup \mathcal{D}_{\text{aw}} \cup \mathcal{K}_{\text{aw}} \cup \mathcal{D}_{\text{olog}} \cup \mathcal{D}_{\text{af}} \), where \( \mathcal{D}_{\text{af}} \) is the set of actions defining the ssf fluent. Then if \( \mathcal{K}_{\text{aw}} \) contains the reflexivity axiom for \( K \),

\[
\mathcal{D} \models (\forall s'. \mathcal{ssf}(\delta,s)) \Rightarrow \{ (\forall s'). \text{Do}(\delta,s,s') \equiv \text{Do}_K(\delta,s,s') \}
\]

The preceding lemma is fundamental for the rest of the work. In the following sections we show how theory compilation relies strongly on the use of regression of the \( \text{Do}_K \) predicate. Given our equivalence we can now regress Do instead of \( \text{Do}_K \) which produces significantly simpler formulae.

An important point is that the equivalence of the semantics is achieved for self-sufficient programs. Proving that a program is self-sufficient may be as hard as doing the regression of \( \text{Do}_K \). Fortunately, there are syntactic accounts of self-sufficiency [13; 20], such as tree programs in which each if-then-else that conditions on \( \delta \) is preceded by a senseq.

4 Planning with programs that sense

Motivated by the problem of Web service composition, and by the desire to use macro-actions in conventional planning settings, our main concern in this paper is with planning with programs that sense. As such, we extend the notion of planning with primitive actions to planning with programs that sense as the fundamental building blocks of a plan. One of our interests is to enable the use of pre-existing programs as macro-actions in a classical planning setting. As pointed out by [11], a plan in the presence of sensing is a program that may contain conditional and loop constructs. In our framework we define a plan in the presence of sensing as a Golog program.

Definition 1 (A plan) Given a theory of action \( \mathcal{D} \), and a goal \( G \) we say that Golog program \( \delta \) is a plan for \( G \) in situation \( s \) relative to theory \( \mathcal{D} \) iff \( \mathcal{D} \models (\forall s'). \text{Do}_K(\delta,s,s') \supseteq G(s') \).

In classical planning, a planning algorithm constructs plan \( \delta \) by choosing actions from a set \( A \) of primitive actions. Rather, in planning with programs that sense, the planner has an additional set \( C \) of programs, which may contain sensing actions, that it can use to construct plans.

Example Consider an agent working on an assembly line constructing several types of widgets. The agent is able to achieve high-level goals including building complex objects using the widgets of the assembly line. In order to achieve her goals the agent must do planning. The agent can perform a variety of primitive actions and also some built-in high-level programs. For example, the following program picks up blocks from the assembly chain, possibly repairs them, and then delivers them to a production zone.

\[
\delta = \text{pick}(b); \text{checkDamaged}(b);
\]

if \( \text{damaged}(b) \) then \( \text{repair}(b); \text{register}(b) \) else nil endif; \( \text{deliver}(b) \)

The action \( \text{pick}(b) \) picks a block \( b \) from the assembly line, action \( \text{checkDamaged}(b) \) is a sense action that senses whether or not \( b \) is damaged, action \( \text{deliver}(b) \) delivers \( b \) to a production zone, and action \( \text{register}(b) \) logs \( b \) in the “damaged” database.

In the interest of space, we do not show all the axioms in the theory; rather, we show some axioms that compose \( \mathcal{D}_{\text{aw}} \) and \( \mathcal{D}_{\text{af}} \).

\[
\begin{align*}
\text{Poss}(\text{pick}(b),s) & \supset \text{inChain}(b), \\
\text{Poss}(\text{repair}(b),s) & \supset \text{damaged}(b), \\
\text{a} = \text{repair}(b) & \supset \text{damaged}(b,\text{do}(a,s)), \\
a = \text{register}(b) & \supset \text{logged}(b,\text{do}(a,s)).
\end{align*}
\]

The successor state axioms for the fluents \( \text{logged} \), \( \text{damaged} \) (generated from \( \mathcal{D}_{\text{af}} \)), and for \( K \) (generated from \( \mathcal{K} \)) are as follows.

\[
\begin{align*}
\text{logged}(b,\text{do}(a,s)) & \equiv a = \text{register}(b) \vee \text{logged}(b,s), \\
\text{damaged}(b,\text{do}(a,s)) & \equiv \text{paintFresh}(b,s) \land a = \text{pick}(b) \vee \\
& \text{damaged}(b,s) \land a \neq \text{repair}(b), \\
K(s',\text{do}(a,s)) & \equiv (\exists s''). s'' = \text{do}(a,s') \land K(s'',s) \land \\
& \{ a = \text{checkDamaged}(b) \supset (\text{damaged}(s'') \equiv \text{damaged}(s)) \}.
\end{align*}
\]

Suppose we want an operator-based planner (e.g., STRIPS, Graphplan, SATplan, etc.) to use the complex action \( \delta \). Instead of a program, we would need to have an operator-based action representation (i.e. we need to represent \( \delta \) as a primitive action). This representation would not only describe the physical effects of the action (e.g., after we perform \( \delta(B) \), block \( B \) is in the production zone and not damaged), but also at a knowledge level (if we know that \( B \) is not damaged, after we perform \( B \) we know whether or not \( B \)'s paint is fresh!).

The rest of this section presents a method that, under certain conditions, transforms a theory of action \( \mathcal{D} \) and a set of programs with sensing \( C \) into a new theory, \( \text{Comp}[\mathcal{D},C] \), that describes the same domain as \( \mathcal{D} \) but that is such that programs in \( \text{Comp}[\mathcal{D},C] \) appear modeled by new primitive actions. In so doing, we are able to use traditional operator-based planners to plan with macro-actions, and to perform WSC with so-called composite services.

4.1 Theory compilation

A program with sensing may produce both effects in the world and in the knowledge of the agent. Therefore, if we want to replace a program by one primitive action, this action should have both knowledge and physical effects. In the standard situation calculus, though, it is normally assumed that actions either affect the world or the knowledge of the agent but not both. Therefore, we will compile each program into one sensing action and one physical action.

We now describe how we can generate a new theory of action that contains a new sensing action \( \text{Obss}_G \) and a new physical action \( \text{Physs}_G \) for each program \( \delta \). Then we prove that those actions, when executed one immediately after the other, capture all physical and knowledge-level effects of the original program \( \delta \).

We start with a theory of action \( \mathcal{D} = \Sigma \cup \mathcal{D}_{\text{act}} \cup \mathcal{D}_{\text{ap}} \cup \mathcal{D}_{\text{aw}} \cup \mathcal{K}_{\text{aw}} \cup \mathcal{D}_{\text{olog}} \cup \mathcal{D}_{\text{af}} \), about a set \( A \) of primitive actions, and we generate a new theory \( \text{Comp}[\mathcal{D},C] \) that contains a new set for SSA, precondition and unique name axioms.

We assume that the set of successor state axioms, \( \mathcal{D}_{\text{aw}} \), has been compiled from sets \( \mathcal{D}_{\text{af}} \) and \( \mathcal{K} \), and that the set of pre-condition axioms, \( \mathcal{D}_{\text{ap}} \), has been compiled from a set of necessary precondition conditions, \( \mathcal{D}_{\text{rec}} \). Furthermore, assume
we have a set of Golog tree programs \( C \) which may contain sensing actions such that for every \( \delta \in C \) it holds that \( D \models (\forall s). ssf(\delta, s) \). Finally, assume that the fluent symbol \( Enabled \) is not part of the language of \( D \). We generate the new theory in the following way.

1. Make \( D'_{eff} := D_{eff} \), \( D'_{nec} := D_{nec} \), and \( K'_a := K_a \), and \( D_{ana} := D_{ana} \).
2. We need that actions \( Obs_{\delta} \) and \( Phys_{\delta} \) are used either in sequence by the planner or not at all. We use the predicate \( Enabled \) to enforce this. The following axioms are added to \( D'_{eff} \) for each \( \delta \in C \):

   \[
   a = Obs_{\delta}(\overline{a}) \cup Enabled(Phys_{\delta}(\overline{a}), do(a, s)),
   \]

   \[
   a = Phys_{\delta}(\overline{a}) \cup \neg Enabled(Phys_{\delta}(\overline{a}), do(a, s)),
   \]

   i.e., \( Phys_{\delta} \) becomes \( Enabled \) immediately after \( Obs_{\delta} \) is executed.
3. For each \( \delta \in C \), we add the following necessary precondition axioms to \( D'_{nec} \):

   \[
   Poss(Obs_{\delta}(\overline{a}), s) \supseteq \mathcal{R}^+[\exists \overline{a} \, Do(\delta, s, s')],
   \]

   \[
   Poss(Phys_{\delta}(\overline{a}), s) \supseteq \neg Enabled(Phys_{\delta}(\overline{a}), do(a, s)),
   \]

   therefore, \( Obs_{\delta}(\overline{a}) \) can be executed iff program \( \delta \) could be executed in \( s \), and \( Phys_{\delta}(\overline{a}) \) iff it is \( Enabled \). The operator \( \mathcal{R}^+[\varphi] \) is a formula uniform in \( s \) and equivalent to \( \varphi \).
4. For each \( \alpha \in A \), and every \( \delta \in C \), we add the following necessary precondition axioms to \( D'_{ana} \):

   \[
   Poss(\alpha(\overline{a}), s) \supseteq \neg Enabled(Phys_{\delta}(\overline{a}), s).
   \]

   We add this because we do not want to allow an arbitrary primitive action after the execution of \( Obs_{\delta} \). Furthermore, for each \( \delta, \delta' \in C \) such that \( \delta \neq \delta' \),

   \[
   Poss(Obs_{\delta}(\overline{a}), s) \supseteq \neg Enabled(Phys_{\delta}(\overline{a}), s),
   \]

   \[
   Poss(Phys_{\delta}(\overline{a}), s) \supseteq \neg Enabled(Phys_{\delta}(\overline{a}), s).
   \]
5. For each fluent \( F(\overline{a}, s) \) in the language of \( D \) that is not \( \mathcal{K} \) fluent, and each complex action \( \delta \in C \) we add the following effect axioms to \( D'_{eff} \):

   \[
   a = Phys_{\delta}(\overline{a}) \wedge \mathcal{R}^+[\exists \overline{a} \, Do(\delta, s, s') \wedge F(\overline{a}, do(a, s))],
   \]

   \[
   a = Phys_{\delta}(\overline{a}) \wedge \mathcal{R}^+[\exists \overline{a} \, Do(\delta, s, s') \wedge \neg F(\overline{a}, do(a, s))],
   \]

   i.e. \( F \) is true (resp. false) after executing \( Phys_{\delta} \) in \( s \) if after executing \( \delta \) in \( s \) it is true (resp. false).
6. For each functional fluent \( f(\overline{a}, s) \) in the language of \( D \), and each complex action \( \delta \in C \) we add the following effect axiom to \( D'_{eff} \):

   \[
   a = Phys_{\delta}(\overline{a}) \wedge \mathcal{R}^+[\exists \overline{a} \, Do(\delta, s, s') \wedge z = f(\overline{a}, s')] \supset z = f(\overline{a}, do(a, s))
   \]

   \footnotetext{4}{A formula is uniform in \( s \) iff all terms of sort situation it mentions are \( s \).}
7. For each \( \delta \in C \), we add the following sufficient condition axiom to \( \mathcal{K}'_{a} \):

   \[
   a = Obs_{\delta}(\overline{a}) \wedge s'' = do(a, s') \wedge \mathcal{R}^+[\exists s_1, s_2 \, (Do(\delta, s, s_1) \wedge Do(\delta, s', s_2) \wedge \mathcal{K}(s_2, do(a, s)))]
   \]

   \[
   \alpha(x) \neq Obs_{\delta}(\overline{a}), \alpha(x) \neq Phys_{\delta}(\overline{a}), Obs_{\delta}(\overline{a}) \neq Phys_{\delta}(\overline{a})
   \]
8. For each \( \delta, \delta' \in C \) such that \( \delta \neq \delta' \) and \( \alpha \in A \), add the following to \( D''_{ana} \):

   \[
   \alpha(x) \neq Obs_{\delta}(\overline{a}), \alpha(x) \neq Phys_{\delta}(\overline{a}), Obs_{\delta}(\overline{a}) \neq Phys_{\delta}(\overline{a})
   \]
9. Compile a new set of SSAs \( D''_{ss} \) from \( D'_{eff} \), and a new set of precondition axioms \( D''_{ap} \) from \( D'_{nec} \). The new theory, is defined as follows.

   \[
   Comp[D, C] = \Sigma \cup D''_{ss} \cup D''_{ap} \cup D''_{ana} \cup D_{ana} \cup K_{ana} \cup D_{golog} \cup D_{ssf}.
   \]

**Theorem 4.1** If \( D \) is consistent and \( C \) contains only deterministic tree programs then \( \text{Comp} [D, C] \) is consistent.

Indeed, if \( C \) contains one non-deterministic action, we cannot guarantee that \( \text{Comp} [D, C] \) is consistent. Furthermore, we can prove that \( Phys_{\delta} \) emulates \( \delta \).

**Lemma 4.1** Let \( D \) be a theory of action, and let \( C \) be a set of deterministic Golog tree programs. Then, for all fluents \( F \) in the language of \( D \) that are not \( \mathcal{K} \) and for every \( \delta \in C \) such that \( D \models ssf(\delta, s) \), theory \( \text{Comp} [D, C] \) entails

\[
(\forall s, s', x). DoK(\delta, s, s') \supset (F(x, s') \equiv F(x, do(Phys_{\delta}, s))), \quad (\forall s, s', x, z). DoK(\delta, s, s') \supset (z = f(x, s') \equiv z = f(x, do(Phys_{\delta}, s)))
\]

Now we establish a complete correspondence at the physical level between our original programs and the compiled primitive actions after performing \( [Obs_{\delta}, Phys_{\delta}] \).

**Theorem 4.2** Under the same assumptions as Lemma 4.1, let \( \phi(\overline{a}) \) be an arbitrary situation-suppressed formula that does not mention the \( K \) fluent. Then,

\[
\text{Comp} [D, C] \models (\forall s, s', x). DoK(\delta, s, s') \supset (\phi(x)[s'] \equiv \phi(x)[do([Obs_{\delta}, Phys_{\delta}], s)])
\]

Also, there is a complete correspondence at a knowledge level between our original complex actions and the compiled primitive actions after performing \( [Obs_{\delta}, Phys_{\delta}] \).

**Theorem 4.3** Let \( D \) be a theory of action and \( C \) be a set of deterministic Golog tree programs, and \( \phi(\overline{a}) \) be a situation-suppressed formula. If \( \delta \in C \) and \( D \models (\forall s). ssf(\delta, s) \), then if \( D \) contains the reflexivity axiom for \( K \),

\[
\text{Comp} [D, C] \models (\forall x, s, s_1). DoK(\delta, s, s_1) \supset (\text{Knows}(\phi(x), s_1) \equiv \text{Knows}(\phi(x)[do([Obs_{\delta}, Phys_{\delta}], s)])).
\]

Now that we have established the correspondence between \( D \) and \( \text{Comp} [D, C] \) we return to planning. In order to achieve a goal \( G \) in a situation \( s \), we now obtain a plan using theory \( \text{Comp} [D, C] \). In order to be useful, this plan should have a counterpart in \( D \), since the executor cannot execute any of the “new” actions in \( \text{Comp} [D, C] \). The following result establishes a way to obtain such a counterpart.
Theorem 4.4 Let \( D \) be a theory of action, \( C \) be a set of deterministic Golog tree programs, and \( G \) be a formula of the situation calculus. Then, if \( \Delta \) is a plan for \( G \) in theory \( \text{Comp}[D,C] \) and situation \( s \), then there exists a plan \( \Delta' \) for \( G \) in theory \( D \) and situation \( s \). Moreover, \( \Delta' \) can be constructed from \( \Delta \).

Proof sketch: We construct \( \Delta' \) by replacing every occurrence of \( [\text{Obs}_s;\text{Phys}_g] \) in \( \Delta \) by \( \delta \). Then we prove that \( \Delta' \) also achieves the goal, from theorems 4.2 and 4.3.

It is worth noting that the preceding proof would not have worked if plans (Definition 1) had been defined as Golog programs with a concurrent construct (such as that of Congolog). Such a construct, say \( \delta_1 || \delta_2 \), would specify that complex actions \( \delta_1 \) and \( \delta_2 \) can be executed concurrently, i.e., any interleaved execution of \( \delta_1 \) and \( \delta_2 \) would reach the goal. Handling this case is important since some actual planners (even in the classical setting) are able to generate plans that are non-linear, i.e., that contain partially ordered sequences of actions (which in practice means concurrent execution).

Imagine that the planner has returned plan \( \{ \text{Obs}_s;\text{Phys}_g\} || A \) for goal \( G \). Given the preconditions in the theory, this means that executing either \( A;\text{Obs}_s;\text{Phys}_g \) or \( \text{Obs}_s;\text{Phys}_g;A \) would achieve the goal in \( \text{Comp}[D,C] \). Unfortunately, this does not necessarily mean that \( \delta || A \) will achieve the goal in \( D \), since \( \delta \) is a complex action. Allowing the execution of \( A \) during the execution of \( \delta \) may invalid some precondition of an action in \( \delta \) or change the truth value of a fluent in a way that is not predicted by the theory compilation. A simple—though not totally satisfactory—fix for this is that whenever the planner returns a plan of the form \( \{ \text{Obs}_s;\text{Phys}_g\} || \delta \) in theory \( \text{Comp}[D,C] \), then a plan in \( D \) is either \( \delta;A \) or \( \Delta;\delta \). We think that less restrictive solutions would imply tweaking the preconditions of the actions in \( \delta \). This issue is part of our current and future research.

Example (cont.) We now show the result of applying the theory compilation to the action theory of our example. For the fluent \( \text{damaged} \), axioms of the form:

\[
a = \text{Phys}_g(b) \land R^a[\langle \exists s' \rangle (\text{do}(\delta, s, s') \land (\neg \text{damaged}(\delta, s')) \supset (\neg \text{damaged}(\delta, \text{do}(a, s)))],
\]

simplify into

\[
a = \text{Phys}_g(b) \land \text{inChain}(b, s) \supset \neg \text{damaged}(\text{do}(a, s)).
\]

Following the same procedure, the SSA generated for \( \text{logged} \) is the following:

\[
\text{logged}(\text{do}(a, s)) \equiv a = \text{register}(b) \lor a = \text{Phys}_g(b) \land \text{paintFresh}(b, s) \lor \text{damaged}(b, s) \land \text{logged}(b, s)
\]

and the following is the SSA for \( K \),

\[
K'(s', \text{do}(a, s)) \equiv (\exists s'').s' = \text{do}(a, s') \land K(s'', s) \land (a = \text{checkDamaged}(b) \supset \text{damaged}(s'') \equiv \text{damaged}(s)) \land (a = \text{Obs}_g(b) \supset \{(\text{damaged}(s') \lor \text{paintFresh}(b, s')) \equiv (\text{damaged}(s) \lor \text{paintFresh}(b, s))\}).
\]

A last thing worth pointing out is that our theory compilation can only be used for complex actions that can be proved self-sufficient for all situations. We could have done this differently. As said before, we could use the conditions that need to hold true for a program to be self-sufficient as a precondition for the newly generated primitive actions. Indeed, formula \( sf(\delta, s) \) encodes all that is required to hold in \( s \) to be able to know how to execute \( \delta \), and therefore we could have added something like \( \text{Poss}(\text{Obs}_s(\delta), s) \supset R'_1[\langle \exists s' \rangle \text{do}(\delta, s, s') \land sf(\delta, s)] \) in step 3 of theory compilation. This modification keeps the validity of our theorems but the resulting expression in the precondition may usually contain complex formulae referring to the knowledge of the agent we view as a problematic in practical applications.

5 From theory to practice

We have shown that under certain circumstances, planning with programs can theoretically be reduced to planning with primitive actions. In this section we identify properties necessary for operator-based planners to exploit these results, with particular attention to some of the more popular existing planners. There are several planning systems that have been proposed in the literature that are able to consider the knowledge of an agent and (in some cases) sensing actions. These include Sensory Graphplan (SGP) [23], the MDP-based planner GPT [3], the model-checking-based planner MBP [2], the logic-programming-based planner \( \pi(\mathcal{P}) \) [22], and the knowledge-level planner PKS [16].

All of these planners but PKS keep an implicit or explicit representation of all the states in which the agent could be in during the execution of the plan (sometimes called belief states), they are propositional, and cannot represent functions. In our view, the limited expressiveness of these planners is extremely restrictive, especially because they are unable to represent functions, which is of a great importance in many practical applications including WSC. To our knowledge, PKS [16] is the only planner in the literature that does not represent belief states explicitly. Moreover, it can represent domains using first-order logic and functions. Nevertheless, it does not allow the representation of knowledge about arbitrary formulae. In particular it cannot represent disjunctive knowledge.

All of these planners are able to represent conditional effects of physical actions, therefore, the representation of action \( \text{Phys}_g \) is straightforward. Unfortunately, the representation of the effects of \( \text{Obs}_g \) is not trivial in some cases. Examining the general structure of the SSA for \( K \) after theory compilation we observe that to represent the effects of \( \text{Obs}_g \) we need two characteristics from a planner.

- The planner must be able to represent conditional sensing actions. Among the planners investigated, SGP is the only one that cannot be adapted to this requirement. The reason is that sensing actions in SGP cannot have preconditions or conditional effects. Others (\( \pi(\mathcal{P}) \), MBP) can be adapted to simulate conditional sensing actions by splitting \( \text{Obs}_g \) into several actions with different preconditions.

MBP does not consider sensing actions explicitly, however they can be ‘simulated’ by representing within the state the last action executed.
The planner must be able to represent that \( \text{Obs}_s \) reports the truth value of, in general, arbitrary formulae. Some planners (SGP, MPB) can represent arbitrary (propositional) observation formulae but others, e.g., (GPT, \( \pi(\mathcal{F}) \), FKS) cannot. This is a somewhat serious limitation for planners that cannot represent such knowledge effects. In our example, this would prevent those planners from realizing that if it is known that the block \( b \) is not damaged, then after executing \([\text{Obs}_s(b) ; \text{Phys}_s(b)]\), it knows whether \( \text{paintFresh}(b) \).

To overcome the limitations of such planners, we have designed an algorithm (which we omit for lack of space) that takes the final SSA for \( K \) and is able to generate some useful knowledge effect axioms. In our example, the algorithm would generate the effect axiom \( \text{Knows}(\neg \text{damaged}(b), s) \Rightarrow K\text{Whether}(\text{paintFresh}(b), \text{do}(\text{Obs}_s, s)) \). Unfortunately, the algorithm is incomplete in the sense that the rules generated cannot capture all knowledge effects stemming from sensing arbitrary formulae.

6 Summary and discussion

In this paper we examined the problem of planning by composing programs, rather than or in addition to primitive actions. The programs that form the building blocks of such plans can, themselves, contain both sensing and world-altering actions. We studied this problem in the language of the situation calculus, appealing to Golog to represent our programs. To this end, we proposed an offline execution semantics for Golog programs with sensing, proving its equivalence to previous online execution semantics, under certain conditions. We then proposed a compilation method that transforms our action theory with programs into a new theory where programs are replaced by primitive actions. This enabled us, in theory, to use traditional operator-based planning techniques to plan with programs that sense for a restricted but compelling class of problems. We concluded by discussing the applicability of these results to existing operator-based planners that allow sensing. Unfortunately, space precludes us from presenting the full proof. Details are available in a longer version of this paper [1]. We also excluded an algorithm that we have developed for planning with these compiled theories. This work makes an important contribution to the general problem of Web service composition by enabling the composition of so-called composite services using traditional operator-based planners that include sensing. The work also provides a mechanism for including macro-actions in operator-based planners that include sensing. We continue to explore these topics in future work.

References


Restrained Revision

Richard Booth
Department of Computing
Macquarie University
Sydney, NSW 2109 Australia
rbooth@ics.mq.edu.au

Samir Chopra
Department of Computer Science
Brooklyn College of CUNY
Brooklyn, NY 11210
schopra@sci.brooklyn.cuny.edu

Thomas Meyer
National ICT Australia
School of CSE, UNSW
Sydney, NSW 2052 Australia
tmeyer@cse.unsw.edu.au

Abstract

As partial justification of their framework for iterated belief revision Darwiche and Pearl convincingly argued against Boutilier’s natural revision and provided a prototypical revision operator that fits into their scheme. We show that the Darwiche-Pearl arguments lead naturally to the acceptance of a smaller class of operators which we refer to as admissible. These are characterised in terms of syntactic as well as semantic postulates. Admissible revision ensures that the penultimate input is not ignored completely, thereby eliminating natural revision, but includes the Darwiche-Pearl operator. Nayak’s lexicographic revision operator, and a newly introduced operator called restrained revision. We give a syntactic and a semantic characterisation of restrained revision, and demonstrate that it satisfies desirable properties. In particular, we show that it is the most conservative of admissible revision operators, while lexicographic revision is the least conservative. This makes possible an interesting comparison with the Darwiche-Pearl framework in which lexicographic revision is also the least conservative, but natural revision is the most conservative. Hence, restrained revision is plausibly viewed as an appropriate replacement for natural revision. Finally we show that restrained revision can also be viewed as a composite operator, consisting of natural revision preceded by an application of a “backwards revision” operator previously studied by Papini.

1 Introduction

Many formal treatments of iterated belief revision rely on recipes for manipulating plausibility orderings, usually total preorders, of possible worlds. Typically, these require that (a) the epistemic input α is entailed by the knowledge base associated with the revised plausibility ordering, and (b) that the remaining worlds are arranged in some plausible ordering corresponding to a rational change of an agent’s beliefs. This ensures that (a) we can obtain a new knowledge base from the lowest rank in the plausibility ordering of possible worlds and (b) that a new ranking of possible worlds is available as a revision target for the next epistemic input.

Most iterated revision schemes are sensitive to the history of belief changes\(^1\), based on a version of the ‘most recent is best’ argument, where the newest information is of higher priority than anything else in the knowledge base. Arguably the most extreme case of this is Nayak’s lexicographic revision [12; 13]. However, there are operators where, once admitted to the knowledge base, it rapidly becomes as much of a candidate for removal as anything else in the set when another, newer, piece of information comes along. Boutilier’s natural revision [4; 5] being a case in point. (A dual to this is what Rott\(^1\) terms radical revision where the new information is accepted with maximal, irremediable entrenchment – see also [17]). Another issue to consider is the problem termed temporal incoherence by Rott [15]:

the comparative recency of information should translate systematically into comparative importance, strength or entrenchedness

In an influential paper Darwiche and Pearl [6] proposed a framework for iterated revision. Their proposal is characterised in terms of sets of syntactic and semantic postulates, but can also be viewed from the perspective of conditional beliefs. To justify their proposal Darwiche and Pearl mount a comprehensive argument. The argument includes a critique of natural revision, which is shown to admit too few changes. In addition, they provide a concrete revision operator which is shown to satisfy their postulates. In many ways this can be seen as the prototypical Darwiche-Pearl operator. It is instructive to observe that the two best-known operators satisfying the Darwiche-Pearl postulates, natural revision and lexicographic revision, form the opposite extremes of the Darwiche-Pearl framework: Natural revision is the most conservative Darwiche-Pearl operator, while lexicographic revision is the least conservative of the Darwiche-Pearl operators.

In this paper we show that the Darwiche-Pearl arguments lead naturally to the acceptance of a smaller class of operators which we refer to as admissible. We provide characterisations of admissible revision, in terms of syntactic as well as semantic postulates. Admissible revision ensures that the penultimate input is not ignored completely. A consequence of this is that natural revision is eliminated. On the other hand,

---

\(^1\)An external revision scheme like [2; 7] is not.
admissible revision includes the prototypical Darwiche-Pearl operator as well as lexicographic revision, the latter result also showing that lexicographic revision is the least conservative of the admissible operators. The removal of natural revision from the scene leaves a gap which is filled by the introduction of a new operator we refer to as restrained revision. It is the most conservative of admissible revision operators, and can thus be seen as an appropriate replacement of natural revision. We give a syntactic and a semantic characterisation of restrained revision, and demonstrate that it satisfies desirable properties. In particular, and unlike lexicographic revision, it ensures that older information is not discarded unnecessarily, and it shows that the problem of temporal incoherence can be dealt with.

Although natural revision does not feature in the class of admissible revision operators, we show that it still has a role to play in iterated revision, provided it is first tempered appropriately. We show that restrained revision can also be viewed as a composite operator, consisting of natural revision preceded by an application of a “backwards revision” operator previously studied by Papini [14].

The paper is organised as follows. After outlining some notation, we review the Darwiche-Pearl framework in Section 2. This is followed by a discussion of admissible revision in Section 3. In Section 4 we introduce restrained revision, and in Section 5 we show how it can be defined as a composite operator. Section 6 concludes and briefly discusses some future work.

1.1 Notation

We assume a finitely generated propositional language $L$ which includes the constants $\top$ and $\bot$, is closed under the usual propositional connectives, and is equipped with a classical model-theoretic semantics. $V$ is the set of valuations of $L$ and $[\alpha]$ (or $[B]$) is the set of models of $\alpha \in L$ (or $B \subseteq L$). Classical entailment is denoted by $\models$ and logical equivalence by $\equiv$. Greek letters $\alpha, \beta, \ldots$ stand for arbitrary formulas.

2 Darwiche-Pearl Revision

Darwiche and Pearl [6] reformulated the AGM postulates [1] to be compatible with their suggested approach to iterated revision. This necessitated a move from knowledge bases to epistemic states. An epistemic state contains, in addition to a knowledge base, all the information needed for coherent reasoning including, in particular, the strategy for belief revision which the agent wishes to employ at a given time. This includes a plausibility ordering on all valuations, a total preorder, with elements lower down in the ordering deemed more plausible.

Definition 1 From every epistemic state $E$, a total preorder on valuations $\preceq_E$, and a consistent knowledge base $B(E)$ can be extracted.\footnote{The requirement that $B(E)$ be consistent enables us to obtain a unique knowledge base from the total preorder $\preceq_E$. Preservation of the results in this paper when this requirement is relaxed is possible, but technically messy.} $\min(\alpha, \preceq_E)$ denotes the minimal models of $\alpha$ under $\preceq_E$. The knowledge base associated with the epistemic state is obtained by considering the minimal models in $\preceq_E$ i.e., $B(E) = \min(\top, \preceq_E)$.

In the reformulated postulates $\ast$ is a belief change operator on epistemic states, not knowledge bases.

\[(E\ast1) \quad B(\ast \alpha) = Cn(B(\ast \alpha)) \]
\[(E\ast2) \quad \alpha \in B(\ast \alpha) \]
\[(E\ast3) \quad B(\ast \alpha) \subseteq B(\alpha) + \alpha \]
\[(E\ast4) \quad \text{If } \alpha \not\in B(\alpha) \text{ then } B(\alpha) + \alpha \subseteq B(\ast \alpha) \]
\[(E\ast5) \quad \text{If } \alpha \equiv \beta \text{ then } \ast \alpha = \ast \beta \]
\[(E\ast6) \quad \bot \in B(\ast \alpha) \iff \neg \alpha \]
\[(E\ast7) \quad B(\ast (\alpha \land \beta)) \subseteq B(\ast \alpha) + \beta \]
\[(E\ast8) \quad \text{If } \neg \beta \not\in B(\ast \alpha) \text{ then } B(\ast \alpha) + \beta \subseteq B(\ast (\alpha \land \beta)) \]

The observant reader will note that our assumption of a consistent $B(E)$ is incompatible with a successful revision by $\bot$. This requires that we jettison $(E\ast6)$ and insist on consistent epistemic inputs only.\footnote{The part of $(E\ast6)$ which requires a consistent $B(E \ast \alpha)$ is rendered superfluous by $(E\ast1)$ and the assumption that knowledge bases extracted from all epistemic states have to be consistent.} We shall refer to the reformulated AGM postulates, with $(E\ast6)$ removed, as RAGM. The main reason for the reformulation occurs in $(E\ast5)$, which states that revising by logically equivalent formulas results in the same epistemic state. The original AGM postulate requires only that the knowledge bases extracted from the resulting epistemic state be the same after revision by logically equivalent formulas.

RAGM guarantees a unique extracted knowledge base (modulo logical equivalence) when revision by $\alpha$ is performed. It sets $[B(E \ast \alpha)]$ equal to $\min(\alpha, \preceq_E)$ and thereby fixes the most plausible valuons in $\preceq_{E \ast \alpha}$. What is not fixed is how to order the remaining valuons.

We now list the Darwiche-Pearl postulates for iterated revision [6].

\[(C1) \quad \text{If } \alpha \models \beta \text{ then } B(\ast \beta \ast \alpha) = B(\ast \alpha) \]
\[(C2) \quad \text{If } \alpha \not\models \beta \text{ then } B(\ast \beta \ast \alpha) = B(\ast \alpha) \]
\[(C3) \quad \text{If } \beta \in B(\ast \alpha) \text{ then } \beta \in B(\ast \beta \ast \alpha) \]
\[(C4) \quad \text{If } \beta \not\in B(\ast \alpha) \text{ then } \neg \beta \not\in B(\ast \beta \ast \alpha) \]

The following are the corresponding semantic versions (with $v, w \in V$):

\[(CR1) \quad \text{If } v \in [\alpha], w \in [\alpha] \text{ then } v \preceq_E w \text{ iff } v \preceq_{E \ast \alpha} w \]
\[(CR2) \quad \text{If } v \in [\neg \alpha], w \in [\neg \alpha] \text{ then } v \preceq_E w \text{ iff } v \preceq_{E \ast \alpha} w \]
\[(CR3) \quad \text{If } v \in [\alpha], w \in [\neg \alpha] \text{ then } v \preceq_E w \text{ only if } v \preceq_{E \ast \alpha} w \]
\[(CR4) \quad \text{If } v \in [\alpha], w \in [\neg \alpha] \text{ then } v \preceq_E w \text{ only if } v \preceq_{E \ast \alpha} w \]

Darwiche and Pearl showed that, given RAGM, a precise correspondence obtains between $(C1)$ and $(CR1)$ above ($i = 1, 2, 3, 4$). The postulate $(C1)$ states that when two pieces of information—one more specific than the other—arrive, the first is made redundant by the second. $(C2)$ says that when two contradictory epistemic inputs arrive, the second one prevails; the second evidence alone yields the same knowledge base. $(C3)$ says that a piece of evidence $\beta$ should be retained
after accommodating more recent evidence $\alpha$ that entails $\beta$ given the current knowledge base. (C4) simply says that no
epistemic input can act as its own defeater. We shall refer to
the belief revision operators satisfying RAGM and (C1) to
(C4) as DP-operators.

One of the guiding principles of belief revision is the prin-
ciple of minimal change: changes to a belief state ought to be
kept to a minimum. What is not always clear is what ought to
be minimised. In AGM theory the prevailing wisdom is that
minimal change refers to the sets of sentences corresponding
to knowledge bases. But there are other interpretations. With
the move from knowledge bases to epistemic states, minimal
to the lowest possible changes
change can be defined in terms of the fewest possible changes
to the associated plausibility ordering $\preceq_E$. In what follows we
will frequently have the opportunity to refer to the latter in-
terpretation of minimal change.

3 Admissible Revision

We now consider two of the best-known DP-operators, and
propose a strengthening of the Darwiche-Pearl framework.
This strengthening is suggested by some of the arguments
advanced by Darwiche and Pearl themselves. It eliminates
one of the operators they criticise, and is satisfied by the sole
operator they provide as an instance of their framework.

The oldest known DP-operator is Boutilier’s natural revis-
ion [4; 5]. Its main feature is the application of the principle
of minimal change to epistemic states. It is characterised by
RAGM plus the following postulate:

(CB) If $\neg\beta \in B(\alpha)$ then $B(\alpha \wedge \beta) = B(\beta)$

(CB) requires that, whenever $B(\alpha)$ is inconsistent with $\beta$,
revising $\alpha$ with $\beta$ will completely ignore the revision by
$\alpha$. Its semantic counterpart is as follows:

(CBR) For $v, w \notin [B(\alpha)]$, $v \preceq_{\alpha \wedge \beta} w$ iff $v \preceq_E w$

From (CBR) it is clear that natural revision is an application
of minimal change to epistemic states. It requires that, bar-
ing the changes mandated by RAGM, the relative ordering
of valuations remains unchanged. So natural revision is the
most conservative of all DP-operators. Such a strict adher-
eence to minimal change is inadvisable and needs to be tem-
pered appropriately, an issue that will be addressed in Section
we see clearly that the animal is red, so we believe it is a red
bird. To remove further doubts we call in a bird expert who

弈出现来说，它是一只红色的鸟，所以它是一只红色的鸟。当它走近时，
这可以被看作是通过将$B(E) = Cn(\neg\beta) = Cn(\text{bird})$和$\alpha \equiv \text{red}
in (CB)$，使我们完全忽略观察$\alpha$如果它没有被采纳。

Given this example it is perhaps surprising that Darwiche and
Pearl never considered the following postulate:

(P) If $\neg\beta \notin B(\alpha)$ then $\beta \in B(\alpha \wedge \beta)$

Recently (P) was also proposed in [10] where it is named In-
dependence. It states that whenever $\beta$ is consistent with
a revision by $\alpha$, it should be retained if a $\beta$-revision is inserted
just before the $\alpha$-revision. Applying this to Example 1 we
see that, since red is consistent with $B(E \leftrightarrow \text{bird})$, we have
$\alpha \equiv \text{red} \rightarrow B(E \leftrightarrow \text{bird})$; that is, we have to believe
the animal, which we now know not to be a bird, is red. The
semantic counterpart of (P) looks like this:

(PR) For $v \in [\alpha]$ and $w \notin [\alpha]$, $v \preceq_{E \wedge \alpha} w$

(PR) requires an $\alpha$-world $v$ that is at least as plausible as a
$\alpha$-world $w$ to be strictly more plausible than $w$ after an $\alpha$
revision.

Proposition 1 If $\ast$ satisfies RAGM, then it satisfies (P) if it
also satisfies (PR).

Rule (PR) enforces certain changes in the ordering $\preceq_E$ after
receipt of $\alpha$. In fact as soon as there exist an $\alpha$-world and a
$\alpha$-world on the same plausibility level somewhere in $\preceq_E$,
(PR) implies $\preceq_{\alpha \wedge \beta} \neq \preceq_{\alpha}$. Furthermore these changes must
also occur even when $\alpha$ is already believed in $E$ to begin
with, i.e., $\alpha \in B(E)$. (Although of course if $\alpha \in B(E)$
then $B(E \wedge \alpha) = B(E)$, i.e., the belief set associated to $E$
will remain unchanged – this follows from RAGM.) The rules
(P)/(PR) ensure input $\alpha$ is believed with a certain minimal
strength of belief – enough to help it survive the next revi-

Note that (P) has the antecedent of (C4) and the conse-
quently of (C3). Thus (P) is stronger than (C3) and (C4) combined.
This is seen from the semantic counterparts of these postu-
lates as well. It also follows that the only concrete example of
an iterated revision operator provided by Darwiche and Pearl
(the operator they refer to as $\ast$ and which employs a form of
Spohnian conditioning [18]) satisfies (PR), and therefore (P)
as well. Furthermore, by adopting (P) we explicitly exclude
natural revision as a permissible operator. So accepting (P) is
a move towards the viewpoint that information obtained be-
fore the latest input ought not to be discarded unecessarily.

Based on this analysis we propose a strengthening of the
Darwiche-Pearl framework in which (C3) and (C4) are re-
placed by (P).

Definition 2 A revision operator is admissible iff it satisfies
RAGM, (C1), (C2), and (P).

Inasmuch as the Darwiche-Pearl framework can be visualised
as one in which $\alpha$-worlds slide “downwards” relative to $\neg\alpha$
worlds, admissible revision ensures that this “downwards”
slide is a strict one.

Another view of (P) is that it is a significant weakening of
the following property, first introduced in [13]:

Recalcitrance If $\neg\beta \in Cn(\alpha)$ then $\beta \in B(\alpha \wedge \beta)$

Semantically, (Recalcitrance) corresponds to the following
property, as was pointed out by Booth in [3] and implicitly
contained in [13]:

(R) For $v \in [\alpha], w \notin [\alpha], v \preceq_{E \wedge \alpha} w$
(Recalcitrance) is a property of Nayak’s lexicographic revision operator [12; 13], the second of the well-known DP-operators we consider. In fact, lexicographic revision is characterised by RAGM together with (C1), (C2) and (Recalcitrance), a result that is easily proved from the semantic counterparts of these properties and Nayak et al.’s semantic characterisation of lexicographic revision in [13]. An analysis of the semantic characterisation of lexicographic revision shows that it is the least conservative of the DP-operators, effecting the most changes in the relative ordering of valuations permitted by RAGM and the Darwiche-Pearl postulates. Since it is also an admissible revision operator, it follows that it is also the least conservative admissible operator.

The problem with (Recalcitrance) is that the decision of whether to accept β after a subsequent revision by α is completely determined by the logical relationship between β and α – the epistemic state $\mathcal{E}$ is robbed of all influence. The replacement of (Recalcitrance) by the weaker (P) already gives $\mathcal{E}$ more influence in the outcome. In the next section we constrain matters further by giving $\mathcal{E}$ as much influence as allowed by the postulates for admissible revision. Such a move ensures greater sensitivity to the agent’s epistemic record in making further changes.

Note that lexicographic revision assumes that more recent information takes complete precedence over information obtained previously. Thus, when applied to Example 1, it requires us to believe that the animal, previously assumed to be a bird, is indeed red, because red is a recent input which does not conflict with the most recently obtained input. While this is a reasonable approach in many circumstances, a dogmatic adherence to it can be problematic, as the following example shows.

Example 2 We observe a creature which is clearly red, but we are too far away to determine whether it is a bird or a land animal. So we adopt the knowledge base $B(\mathcal{E}) \equiv \text{red}$. Next to us is a person who declares that, since the creature is red, it must be a bird. We have no reason to doubt him, and so we adopt the belief $\text{red} \rightarrow \text{bird}$. Now the creature moves closer and it becomes clear that it is not a bird. The question is, should we continue believing that it is red? Under the circumstances described above we want our initial observation to take precedence over the opinion of the self-proclaimed expert and believe that the animal is red. But lexicographic revision does not allow us to do so.

While (P) allows for the possibility of retaining the belief that the animal is red, it does not enforce this belief. Below we provide a property which does so.\(^4\) To help us express this property, we introduce an extra piece of terminology and notation:

**Definition 3** $\alpha$ and $\beta$ counteract with respect to an epistemic state $\mathcal{E}$, written $\alpha \neg \neg \mathcal{E} \beta$, iff $\neg \beta \in B(\mathcal{E} \ast \alpha)$ and $\neg \alpha \in B(\mathcal{E} \ast \beta)$.

The use of the term counteract to describe this relation is taken from [13]. $\alpha \neg \neg \mathcal{E} \beta$ means that, from the viewpoint of $\mathcal{E}$, $\alpha$ and $\beta$ tend to “exclude” each other. Note that $\neg \neg \mathcal{E}$ is (a) symmetric, and (b) depends only on the total preorder $\preceq_{\mathcal{E}}$ obtained from $\mathcal{E}$. Furthermore, if $\alpha$ and $\beta$ are logically inconsistent then $\alpha \neg \neg \mathcal{E} \beta$, but the converse need not hold. Thus $\neg \neg \mathcal{E}$ can be seen as a weak form of inconsistency. Now consider the following property:

**(D)** If $\alpha \neg \neg \mathcal{E} \beta$ then $\neg \alpha \in B(\mathcal{E} \ast \alpha \ast \beta)$

(D) requires that, whenever $\alpha$ and $\beta$ counteract with respect to $\mathcal{E}$, $\alpha$ should be disallowed when an $\alpha$-revision is followed by a $\beta$-revision. That is, when the $\beta$-revision of $\mathcal{E} \ast \alpha$ takes place, the information encoded in $\mathcal{E} \ast \alpha$ precedes over the information contained in $\mathcal{E} \ast \beta$; Darwiche and Pearl considered this property (it is their rule (C6) in [6]), but argued against it, citing the following example.

Example 3 [6] We believe that exactly one of John and Mary committed a murder. Now we get persuasive evidence indicating that John is the murderer. This is followed by persuasive information indicating that Mary is the murderer. Let $\alpha$ represent that John committed the murder and $\beta$ that Mary committed the murder. Then (D) forces us to conclude that Mary, but not John, was involved in the murder. This, according to Darwiche and Pearl, is counterintuitive, since we should conclude that both were involved in committing the murder.

Darwiche and Pearl’s argument against (D) rests upon the assumption that more recent information ought to take precedence over information previously obtained. But as we have seen in Example 2, this is not always a valid assumption. In fact, the application of (D) to Example 2, with $\alpha = \text{red} \rightarrow \text{bird}$ and $\beta = \neg \text{bird}$, produces the intuitively correct result of a belief in the observed animal being red: $\text{red} \in B(\mathcal{E} \ast (\text{red} \rightarrow \text{bird}) \ast \neg \text{bird})$.

Another way to gain insight into the significance of (D) is to consider its semantic counterpart:

**(DR)** For $v \in [\neg \alpha]$, $w \in [\alpha]$, and $w \notin B(\mathcal{E} \ast \alpha)$, if $v \sim_{\mathcal{E}} w$ then $v \sim_{\mathcal{E} \ast \alpha} w$.

(DR) curtails the rise in plausibility of $\alpha$-worlds after an $\alpha$-revision. It ensures that, with the exception of the most plausible $\alpha$-worlds, the relative ordering between an $\alpha$-world and the $\neg \alpha$-worlds more plausible than it remains unchanged.

**Proposition 2** Whenever a revision operator $*$ satisfies RAGM, then $*$ satisfies (D) iff it satisfies (DR).

## 4 Restrained Revision

We now strengthen the requirements on admissible revision (those operators satisfying RAGM, (C1), (C2) and (P)) by insisting that (D) is satisfied as well. To do so, let us first consider the semantic definition of an interesting admissible revision operator. Recall that RAGM fixes the set of $(\Sigma_{\mathcal{E} \ast \alpha})$-minimal models, setting them equal to $\min(\alpha, \preceq_{\mathcal{E}})$, but places no restriction on how the remaining valuations should be ordered. The following property provides a unique relative ordering of the remaining valuations.

**(R)** $\forall v, w \notin B(\mathcal{E} \ast \alpha), v \preceq_{\mathcal{E} \ast \alpha} w$ iff

\[
\begin{cases} 
  v \preceq_{\mathcal{E}} w \text{ or}, \\
  v \preceq_{\mathcal{E}} w \text{ and } (v \in [\alpha] \text{ or } w \in [\neg \alpha])
\end{cases}
\]
(R) says that the relative ordering of the valuations that are not \((\leq_{E,A})\)-minimal remains unchanged, except for \(\alpha\)-worlds and \(\neg\alpha\)-worlds on the same plausibility level; those are split into two levels with the \(\alpha\)-worlds more plausible than the \(\neg\alpha\)-worlds. So RAGM combined with (R) fixes a unique operator.

**Definition 4** The (unique) revision operator satisfying RAGM and (R) is called restrained revision.

It turns out that restrained revision is the only admissible revision operator satisfying (D).

**Theorem 1** RAGM, (C1), (C2), (P) and (D) provide an exact characterisation of restrained revision.

The proof is easily obtained from the semantic counterparts of these properties.

Another interpretation of (R) is that it maintains the relative ordering of the valuations that are not \((\leq_{E,A})\)-minimal, except for the changes mandated by (PR). From this it can be seen that restrained revision is the most conservative of all admissible revision operators. So, in the context of admissible revision, restrained revision takes on the role played by natural revision in the Darwiche-Pearl framework.

Examples 2 and 3 share some interesting structural properties. In both, the initial knowledge base \(B(E)\) is pairwise consistent with each of the subsequent sentences in the revision sequence, while the sentences in each revision sequence are pairwise inconsistent. And in both examples the information contained in the initial knowledge base \(B(E)\) is retained after the revision sequence. These commonalities are instances of an important general result. Let \(\Gamma\) denote the non-empty sequence of inputs \(\gamma_1, \ldots, \gamma_n\), and let \(E * \Gamma\) denote the revision sequence \(E * \gamma_1 * \ldots * \gamma_n\). Furthermore we shall refer to an epistemic state \(E\) as \(\Gamma\)-compatible provided that \(\neg\gamma_i \notin B(E)\) for every \(i\) in \(\{1, \ldots, n\}\).

**Proposition 3** Restricted revision satisfies (O).

Although restrained revision preserves information which has not been directly contradicted, it is not dogmatically wedded to older information. If neither of two successive, but incompatible, epistemic states are in conflict with any of the inputs of a sequence \(\Gamma = \gamma_1, \ldots, \gamma_n\), it prefers the latter epistemic state when revising by \(\Gamma\).

**Proposition 4** Restricted revision satisfies the following property:

\((Q)\) If \(E\) and \(E * \alpha\) are both \(\Gamma\)-compatible but \(B(E) \cup B(E * \alpha) \models \bot\), then \(B(E * \alpha) \subseteq B(E * \alpha * \Gamma)\) and \(B(E) \not\subseteq B(E * \alpha * \Gamma)\).

Next we consider another preservation property, but this time, unlike the case for (O) and (Q), we look at circumstances where \(B(E)\) is incompatible with some of the inputs in a revision sequence.

\((S)\) If \(\neg\beta \in B(E * \alpha)\) and \(\neg\beta \in B(E * \neg\alpha)\) then \(B(E * \alpha * \neg\alpha * \beta) = B(E * \alpha * \beta)\),

Note that, given RAGM, the antecedent of (S) implies that \(\neg\beta \in B(E)\). Thus (S) states that if \(\neg\beta\) is believed initially, and that a subsequent commitment to either \(\alpha\) or its negation would not change this fact, then after the sequence of inputs in which \(\beta\) is preceded by \(\alpha\) and \(\neg\alpha\), the second input concerning \(\alpha\) is nullified, and the older input regarding \(\alpha\) is retained.

**Proposition 5** Restricted revision satisfies (S).

We now turn to two properties first mentioned (as far as we know) by Schlechta et al. in [16] (see also [11]):

\((\text{Disj1})\) \(B(E * \alpha * \beta) \cap B(E * \gamma * \beta) \subseteq B(E * (\alpha \lor \gamma) * \beta)\).

\((\text{Disj2})\) \(B(E * (\alpha \lor \gamma) * \beta) \subseteq B(E * \alpha * \beta) \cup B(E * \gamma * \beta)\).

\(\text{(Disj1)}\) says that if a sentence is believed after any one of two sequences of revisions that differ only at step \(i\) (step \(i\) being \(\alpha\) in one case and \(\gamma\) in the other), then the sentence should also be believed after that sequence which differs from both only in that step \(i\) is a revision by the disjunction \(\alpha \lor \gamma\). Similarly, \(\text{(Disj2)}\) says that every sentence believed after an \((\alpha \lor \gamma)\)-revision should be believed after at least one of \((\alpha \beta)\) and \((\gamma \beta)\). Both conditions are reasonable properties to expect of revision operators.

**Proposition 6** Restricted revision satisfies \((\text{Disj1})\) and \((\text{Disj2})\).

It can be shown that lexicographic revision satisfies these rules.

We now provide a more compact syntactic representation of restrained revision. First we show that (C1) and (P) can be combined into a single property, and so can (C2) and (D).

**Proposition 7** Given RAGM,

1. \((\text{C1})\) and (P) are together equivalent to the single rule

\[(\text{C1P})\] \(\neg\alpha \notin B(E * \beta)\) then \(B(E * \alpha * \beta) = B(E * (\alpha \lor \beta))\).

2. \((\text{C2})\) and (D) are together equivalent to the single rule

\[(\text{C2D})\] \(\alpha \iff_E \beta\) then \(B(E * \alpha * \beta) = B(E * \beta)\).

Both \((\text{C1P})\) and \((\text{C2D})\) provide conditions for the reduction of the two-step revision sequence \(E * \alpha \beta\) to a single-step revision. \((\text{C1P})\) reduces it to an \((\alpha \lor \beta)\)-revision when \(\alpha\) is consistent with a \(\beta\)-revision. \((\text{C2D})\) reduces it to a \(\beta\)-revision, ignoring \(\alpha\) completely, when \(\alpha\) and \(\beta\) counteract with respect to \(E\). Now, it follows from RAGM that the consequent of \((\text{C1P})\) also obtains when \(\beta\) is nullified, and \(\alpha\) is retained. Putting this together we get a most succinct characterisation of restrained revision.

**Proposition 8** Restricted revision is the unique operator which satisfies RAGM and:

\[B(E * \alpha * \beta) = \begin{cases} B(E * \beta) & \text{if } \alpha \iff_E \beta \text{ otherwise.} \end{cases}\]

If we were to replace \(\alpha \iff_E \beta\) in the first clause above by the stronger \(\alpha \iff \beta\), we would obtain instead the characterisation of lexicographic revision given in [13].
5 Restrained Revision as a Composite Operator

As we saw in Section 3, Boutilier’s natural revision operator – let us denote it in this section by $\oplus$ – is vulnerable to damaging counterexamples such as the red bird Example 1, and fails to satisfy the very reasonable postulate (P). Although a new input $\alpha$ is accepted in the very next epistemic state $E \oplus \alpha$, $\oplus$ does not in any way provide for the preservation of $\alpha$ after subsequent revisions. As Hans Rott [15, p. 128] describes it, “[t]he most recent input sentence is always embraced without reservation, the last but one input sentence, however, is treated with utter disrespect”. Thus, there seem to be convincing reasons to reject $\oplus$ as a viable operator for performing iterated revision. However, the literature on epistemic state change constantly reminds us that keeping changes minimal should be a major concern, and when judged from a purely minimal change viewpoint, it is clear that $\oplus$ can’t be beat! How can we find our way out of this apparent quandary? In this section we show that the use of $\oplus$ can be retained, provided its application is preceded by an intermediate operation in which, rather than revising $E$ by new input $\alpha$, essentially $\alpha$ is revised by $E$.

Given an epistemic state $E$ and sentence $\alpha$, let us denote by $E \diamond \alpha$ the result of this intermediate operation. $E \diamond \alpha$ is an epistemic state. The idea is that when forming $E \diamond \alpha$, the information in $E$ should be maintained. That is, the total preorder $\preceq_{E \diamond \alpha}$ should satisfy

$$v \preceq_{E} w \text{ implies } v \preceq_{E \diamond \alpha} w.$$  

(1)

But rather than leaving behind $\alpha$ entirely in favour of $E$, as much of the informational content of $\alpha$ should be preserved in $E \diamond \alpha$ as possible. This is formalised by saying that for any $v \in [\alpha], w \in [-\alpha]$, we should take $v \preceq_{E \diamond \alpha} w$ as long as this does not conflict with (1) above. It is this second requirement which will guarantee $\alpha$ enough of a “presence” in the revised epistemic state $E \diamond \alpha$ to help it survive subsequent revisions and allow (P) to be captured. Taken together, the above two requirements are enough to specify $\preceq_{E \diamond \alpha}$ uniquely:

$$v \preceq_{E \diamond \alpha} w \text{ iff } \begin{cases} v \preceq_{E} w \text{ if } v \in [\alpha] \text{ or } w \in [-\alpha], \\ v \preceq_{E} w \text{ otherwise}. \end{cases}$$  

(2)

Thus, $\preceq_{E \diamond \alpha}$ is just the lexicographic refinement of $\preceq_{E}$ by the “two-level” total preorder $\preceq_{\alpha}$ defined by $v \preceq_{\alpha} w$ iff $v \in [\alpha]$ or $w \in [-\alpha]$. This “backwards revision” operator is not new. It has been studied by Papini in [14]. We do not necessarily have $\alpha \in B(E \diamond \alpha)$ (this will hold only if $-\alpha \not\in B(E)$), and so $\diamond$ does not satisfy RAGM.

Given $\alpha$, we can define the composite revision operator $*_{\diamond}$ by setting

$$E *_{\diamond} \alpha = (E \diamond \alpha) \oplus \alpha$$  

(3)

This is reminiscent of the Levi Identity [9], used in AGM theory as a recipe for reducing the operation of revision on knowledge bases to a composite operation consisting of contraction plus expansion. In (3), $\oplus$ is playing the role of expansion. The operator $*_{\diamond}$ does satisfy RAGM. In fact, as can easily be seen by comparing (2) above with condition (R) at the start of Section 4, $*_{\diamond}$ coincides with restrained revision.

Proposition 9 Let $*_{R}$ denote the restrained revision operator. Then $*_{R} = *_{\diamond}$

Thus we have proved that restrained revision can be viewed as a combination of two existing operators.

6 Conclusion

We have shown that the Darwiche-Pearl arguments lead to the acceptance of the admissible revision operators as a class worthy of study. The restrained revision operator, in particular, exhibits quite desirable properties. Besides taking the place of natural revision as the operator adhering most closely to the principle of minimal change, its satisfaction of the properties (O) and (Q) shows that it does not unnecessarily remove previously obtained information.

For future work we would like to explore more thoroughly the class of admissible revision operators. In this paper we saw that restrained revision and lexicographic revision lie at opposite ends of the spectrum of admissible operators. They represent respectively the most conservative and the least conservative admissible operators. A natural question is whether there exists an axiomatisable class of admissible operators which represents the “middle ground”. One clue for finding such a class can be found in the counteracts relation $\approx_{E}$ which can be derived from an epistemic state $E$.

As we said, this relation depends only on the preorder $\preceq_{E}$ associated to $E$. In fact, given any total preorder $\preceq$ over $V$ we can define the relation $\approx_{E}$ by

$$\alpha \approx_{E} \beta \iff \min(\alpha, \preceq) \subseteq [-\beta] \text{ and } \min(\beta, \preceq) \subseteq [-\alpha].$$

Then clearly $\approx_{E} = \approx_{E}$ $\approx_{E}$. Furthermore if $\preceq$ is the full relation $V \times V$ then $\approx_{E}$ reduces to logical inconsistency. A counteracts relation stronger than $\approx_{E}$, but still weaker than logical inconsistency can be found by setting $\approx_{E} = \approx_{E}$, where $\approx'$ lies somewhere in between $\approx_{E}$ and $V \times V$. Hence one avenue worth exploring might be to assume that from each epistemic state $E$ we can extract not one but two preorders $\preceq_{E}$ and $\preceq_{E}$ such that $\preceq_{E} \subseteq \preceq_{E} \subseteq \preceq_{E}$. Then, instead of only requiring $\alpha \approx_{E} \beta$ to deduce $\neg \alpha \in B(E \diamond \alpha \beta)$, as is done with restrained revision (the postulate (D)), we could require the stronger condition $\alpha \approx_{E} \beta$ for this to hold. We are currently experimenting with strategies for using the second preorder to guide the manipulation of $\preceq_{E}$ to enable this property to be satisfied.

Acknowledgement

Much of the first author’s work was done while a researcher at Wollongong University. He wishes to thank Aditya Ghose both for making it possible to enjoy a great working environment there, and also for some interesting comments on this work. National ICT Australia is funded by the Australia Government’s Department of Communications, Information and Technology and the Arts and the Australian Research Council through Backing Australia’s Ability and the ICT Centre of...
Excellence program. It is supported by its members the Australian National University, University of NSW, ACT Government, NSW Government and affiliate partner University of Sydney.

References


Modeling the role of (ab)normality in the ascription of causality judgments by agents

Didier Dubois  Henri Prade
IRIT, 118, route de Narbonne, 31062 Toulouse Cedex 4, France
{dubois, prade}@irit.fr

Abstract
How agents ascribe causal links in face of a sequence of reported events, on the basis of their knowledge about how the world behaves, is a problem of interest, which should be distinguished from classical abduction in diagnosis tasks. The paper suggests an approach to this problem, assuming that agents’ knowledge is expressed in terms of non-monotonic consequence relations (which discriminate between what is normal and what is exceptional). This has several benefits. It enables to emphasize the role played by elements perceived as abnormal among the reported events when ascribing causal relationships, and distinguish between factors that facilitated the occurrence of events, from other events that are considered as being more instrumental. Besides, it also points out the bipolar nature of imprecisely known causal links where effects that are guaranteed to be possible are distinguished from effects that are just non-impossible.

1 Introduction
What does it mean, when do we say that “an event, or an action A causes an event B”? This question has been for a long time the object of speculations from philosophers [16], [18], [27], logicians, physicists, but also from lawyers [14], [20] (since responsibility and causality are related notions), and from cognitive psychologists. It is only rather recently that researchers in Artificial Intelligence (AI) were in turn interested in problems arising from the modeling of causal links, see, e.g., Shafer [29], Pearl [23], Halpern and Pearl [12]-[13], or Turner [32]. One can distinguish several AI problems where the idea of causality plays a central role. The problem discussed here is not the one of classical diagnosis. In diagnosis, one does not know what cause(s) has/have taken place, and it amounts to finding out possible cause(s) having plausibly led to some observed facts. This is a matter of abduction, taking advantage of knowledge about links between potential causes and effects [25]. This paper is not either dealing with the qualitative simulation of dynamical systems, nor with the problem of describing changes caused by the execution of actions, nor with what does not change when actions are performed.

Our concern here is a different question, namely the explanation of a sequence of reported events, in terms of pairs of events that could be considered as related by a causality relation. Indeed, we are interested here in distinguishing, in such a chain of known events, those which maintain relations that can be described as “causal”, in order to be able to provide explanations which are cognitively relevant for a human user, on the basis of beliefs entertained by the agent. Indeed, equipping inference or decision support systems with explanations capabilities is a general issue in AI, and the idea of causality is closely related to the one of explanation (and maybe more particularly to the idea of “negative explanation” [26], for answering questions of the “why not” type (“why did event A not occur?”). Causality is generally associated to the idea of conditional counterfactual. Namely, A causes B insofar as it is true that had A not taken place, B would not have occurred. This idea underlies the approach initiated by G. H. von Wright (see, e.g. [4] for an overall discussion) in modal logic. This formalization aims at capturing the idea that the action of an agent i is the cause that p is true if and only if either
- p were false before the action and if agent i had not made the action then p would not have become true, or
- if agent i maintains p true against the normal course of things.

Later, the idea of counterfactual was also at the basis of the approach proposed by Pearl [23] in the setting of conditional probabilistic networks. One should be able to answer such questions as “what are the direct cause(s) of event A?”, “to what degree B is a direct cause of A”, “can it be said that B is an indirect effect of A?”, or “to what extent an agent can be regarded as responsible for event A?”. The assessment of responsibility on the basis of the identification of causal relationships raises still further problems that are not discussed here. Let us note however that the idea of “counterfactual”, alone, may be perhaps not enough to fully account for the idea of causality, as perceived by humans. Indeed, in the case of events that have occurred in an abnormal situation, it seems that the explanation must privilege the “abnormal” conditions among the circumstances which took place, as pointed out by Hart and Honoré [14], and experimentally checked by cognitive psychologists [15]. Moreover, there are important differences between causality relation and logical entailment that are worth investigating, especially in connection, with the modeling of uncertainty and imprecision in causal relations.

The paper is organized in two main parts. Section 2 briefly reviews some existing proposals in AI, while section 3 advocates...
an approach to causality assessment in the setting of non-monotonic consequence relations. This paper is partly based on material contained in [7] and is a revised and expanded version of an unpublished paper [8].

2 Some Recent Viewpoints on Causality

Researchers in Artificial Intelligence often refer to the concept of causality in an implicit or explicit manner, at least when they speak about "causal relations" or "causal processes". This is generally the case in diagnosis problems, where starting from observations, possibly vague and uncertain, one seeks to infer the (most) plausible causes of a given situation. In these approaches, the identification of the causes is based on knowledge connecting "causes" and "effects". These relations are often modeled by conditional probabilities of the form $\text{Prob}(\text{effects} | \text{causes})$, the effects often not being known with precision and certainty. The Bayesian framework (Pearl [21], [22]) in particular makes it possible to highlight the phenomenon of explanation revision ("explaining away"). This occurs when an effect is equally explained by two independent potential causes, and then a subsequent observation that favors one of these causes leads to a reduction in the plausibility of the other cause. However, Bayesian nets, which provide a representation of a joint probability distribution under the form of a directed graph, do not necessarily reflect causal links between their nodes (different graphical representations are obtained depending on the ordering in which variables are taken) [6].

Following another path, Shafer [28], [29] developed a "causal logic" starting from the idea of trees of events (where each node corresponds to a possible choice, to a situation), that one can then equip with measures of uncertainty. This approach by tree of events makes it possible to represent the temporal sequence between events. More generally, the notion of "event space" makes it possible to handle situations that correspond to nodes in trees of more or less detailed events. The situations are connected not only by relations of precedence, but also by five basic possible relations of specificity and entailment that take place between two (instantaneous) events $S$ and $T$. Namely, an event $S$ can: i) be a specialization of $T$ (if $S$ happens, $T$ also happens at the same time); ii) require $T$ for its achievement (if $S$ happens, $T$ has already happened); iii) announce (be certainly followed by) $T$ (if $S$ happens, $T$ must happen afterwards); iv) be possibly followed by $T$ (if $S$ happens, it is possible that $T$ happens afterwards); v) exclude $T$ (if $S$ happens, $T$ will not happen). Shafer [30], without proposing a universal definition of causality, discusses the specific meaning of the classical concept of cause, from the point of view of Law: an action that is necessary and sufficient for an effect. He stresses that in the relation "$A$ causes $B$", $A$ is an action (which can be performed by an animate agent, or even by an inanimate agent, as for instance a storm), while $B$ is an instantaneous event, which contrasts with the five relations above that relate instantaneous events.

More recently, within a different approach, Halpern and Pearl [12] distinguish the idea of a real cause ("cause in fact") from that of a potential cause. They use as a starting point ideas from I. J. Good on the problem of determining responsibilities (which must be based on "real causes"). This issue is illustrated by the typical example of two fires $A$ and $B$ progressing towards an house. If fire $A$ burns the house before fire $B$, then fire $A$ is considered as being "the real cause" for the damages, even it is known that the house would have certainly burnt as well due to fire $B$. For modeling such problems, Halpern and Pearl used a framework where there is an a priori distinction between endogenous variables (the possible values of which are governed by structural equations, corresponding, for example to physical Laws), and exogenous variables (determined by external factors). The latter type of variables cannot be entitled as causes. The definition of causality in this setting remains closely related to the idea of conditional "counterfactual ". It allows the counterfactual test to be conducted over a particular contingency (which needs not correspond to the actual situation in question), and it formalizes the notion of an active causal process. More precisely, for these authors, the fact $A$ that a subset of endogenous variables has taken some definite values is the real cause of an event $B$, if $A$ and $B$ are true in the real world, if this subset is minimal, and if another value assignment to this subset of variables would make $B$ false, the values of the other endogenous variables that do not directly participate to the occurrence of $B$ being fixed in some manner, and if $A$ alone is enough for bringing about $B$ in this context. This approach offers a reasonable model of the idea of causality and makes it possible to treat examples that pose problems in other approaches of causality. In addition, another important problem was also studied by Pearl [24]. It relates to the distinction between the direct effects and the indirect effects of an event. This distinction is indeed of great practical importance in fields such as legal reasoning. Building upon this notion of actual cause, Chockler and Halpern [2] introduce definitions of responsibility and blame. The extent to which a cause (or agent) is responsible for an effect is graded rather than dichotomous, and depends on the presence of other potential causes (or agents). However, some authors (noticeably Zadeh [34]) argue that causality is indefinable, pointing out that in case of a confluence of events, it is not clear to what "degree", if any, each potential cause did or does contribute to the resulting situation, it is hard to distinguish between statistical correlation and uncertain causality, or to identify causes in a chain of events (raising the question "is causality transitive?").

Example (from Zadeh [34]): “I am called by a friend. He needs my help and asks me to rush to his home. I jump into my car and drive as fast as I can. At an intersection, I am hit by another car. I am killed. Who caused my death? My friend; I; or the driver of the car that hit me.” Note that in such a scenario, it seems possible to expand the list of candidate causes very easily, in an almost endless manner, as here, e.g., “my emotionality that limits my capacities to avoid accidents”, “ in my hurry, I had not fastened my security belt”, or even “the fact that the phone was working and I was there for receiving the call”.

Considering the sequence of events, “my friend calls I rush to his home I have an accident I am killed”, what is
striking here is “one might have another output as well”. Indeed, “I have an accident” is only a possible consequence from “I rush with my car”, while in the sequence “I throw a stone with force into the window, the pane is broken”, our perception of causality does not seem to be debatable, since the result is a sure consequence that follows with almost total certainty from my gesture. This raises the question of the representation of imprecise and uncertain causal relations.

3 Causality, Nonmonotonicity and Bipolarity

The notion of causality is naturally encountered when modeling dynamical systems. In A.I., this was the case in qualitative physics [3] (where causality plays a role in qualitative simulation when propagating constraints in influence graphs), and in logics of action. Relations between nonmonotonic inference and causality were already emphasized by authors dealing with reasoning about actions and the frame problem; see especially McCain and Turner [19], Turner [32], Giunchiglia et al. [11].

This was motivated by the insufficiency of material implication for representing a causal link between a cause and an effect. “Causal rules” are then understood as “there is a cause for effect B to be true if it is true that A has just been executed”, where “there is a cause for” is modeled by a modal operator. In the following, taking some inspiration rather from works in belief revision, a nonmonotonic consequence relation is also used for approaching causality. However, we are not interested in finding what causes the change of beliefs of an agent, but rather in determining from his knowledge what the agent considers to be the cause of some observed change in his environment.

This problem should not be confused either with the abduction of causal explanations from observations (which is another problem related to belief revision and nonmonotonic reasoning, see Bouliguer and Becher [1A], Pagnucco et al. [20A]). Indeed in the following, we are not trying to guess what plausibly happened from observations, since what happened is supposed here to be reported.

3.1 Reported facts and knowledge about change

We first define the notion of epistemic state in a dynamic environment, before introducing two related definitions pertaining to the perception of causality, and discussing them. Assume an agent believes a certain number of facts about the current state of the world are true at time t. Let us first observe that there are three possible epistemic attitudes of an agent about a fact B. Namely,

- B is known to be true (more generally, B is more or less certainly true);
- B is unknown;
- B is known to be false (more generally, B is more or less certainly false).

How is an epistemic state revised when an action (performed by some agent) or an event takes place? Let A be a reported fact like an action or an event. Let t and t+i be two instants just before and after the fact A is reported. Let B be a fact pertaining to time t. There are five basic situations concerning the evolution of the epistemic status of B [5]:

- i) B unknown at time t; knowing that A has occurred, B remains unknown in t+1 (A does not inform about B)
- ii) B unknown at time t; knowing that A has occurred, B is known to be true in t+1 (A justifies B)
- iii) B is known to be true at time t; knowing that A has occurred, B remains known to be true in t+1 (B independent from A)
- iv) B is known to be true at time t; knowing that A has occurred, B becomes unknown in t+1 (A cancels B)
- v) B is known to be true at time t; knowing that A has occurred, B is now known as false in t+1 (A refutes B, or equivalently, causes ¬B).

Note that in [5], the change concerned the evolution of agent’s beliefs about a static world, upon learning additional information. Here, we consider how the agent’s beliefs evolve when something happened in the world and is reported to the agent.

Following von Wright (see [4]), one can distinguish two scenarios where one would say that A causes B:

- 1) B was known to be false, and after A takes place B becomes true, although in the normal course of things, where A does not happen, B would have persisted to be false (stable state). This corresponds to the fifth case above;
- 2) B was known to be true, and after A takes place B is still true, although in the normal course of things, where A does not happen, B would have become false (unstable state). This is not covered by any case above, since it is implicitly assumed that B remain unchanged if A (and anything else) does not occur.

The above remarks suggest taking advantage of nonmonotonic reasoning for describing some forms of perceived causality.

Assume that in context C one has normally ¬B. This is supposed to be represented by means of a nonmonotonic consequence relation C ⊨ ¬B, in the sense of Kraus, Lehmann and Magidor [17], which reads “if we are in context C, then generally B is false”. Such a relation ⊨ is reflexive and characterized by the above properties forming the so-called “system P” (⊨ denotes the classical logical entailment):

- Left equivalence: E ⊨ G and E = F imply F ⊨ G; Right weakening: E ⊨ F and F = G imply E ⊨ G;
- AND: E ⊨ F and E ⊨ G imply E ⊨ F ∧ G; OR: E ⊨ F and E ⊨ G imply E ⊨ F ∨ G;
- Cautious monotony: E ⊨ F and E ⊨ G imply E ⊨ F ∨ G;
- Cut: E ⊨ F and E ⊨ G imply E ⊨ G.

The reason for choosing this inference framework is its extreme cautiousness and the fact that it corresponds to a qualitative version of probabilistic and possibilistic reasoning frameworks. In the discussion, the following assumptions are made:

i) the agent observes (or knows) a series of reported facts, including B, A, B∗ , namely B was false at time t when A takes place and B becomes true after;
ii) persistence of the truth status of B whatever in the normal course of things, namely B ⊨ B∗ , B ⊨ B∗ ;
iii) temporal indices are omitted when they are not necessary for avoiding confusion;
iv) A and F are reported actions, B, C are state of facts, even if notations do not discriminate them. Assume now that there exist F and A such that $F \not\square C \not\square B$ (where $\square$ denotes the negation of $\rightarrow$) on the one hand, and $A \not\square F \not\square C \not\rightarrow B$ on the other hand, then we can say that in context C, A together with $F \text{ cause } B$, while $F \text{ facilitates } B$ its appearing. More formally,

**Definition 1** (Facilitation): Given a nonmonotonic consequence relation $\rightarrow$, if $C \not\rightarrow \square B$ and $F \not\square C \not\square \square B$, then $F$ is said to *facilitate* the appearing of $B$ in context $C$.

Note that the above nonmonotonic system P does not allow to exploit pieces of knowledge of the facilitation type since it does not provide for the expression of the negation of an inference. Using this information requires the use of rational closure, where cautious monotony is strengthened into [17]:

Rational monotony: $E \not\square \square F$ and $E \not\rightarrow \sim G$ imply $E \not\square F \not\rightarrow \sim G$.

**Definition 2** (Perceived causation): Given a nonmonotonic consequence relation $\rightarrow$, if $C \not\rightarrow \square B$ and $A \not\square C \not\rightarrow B$, then A is said to *cause B* in context $C$.

Of course, in the above definition A can stand for any compound reported fact such as $A' \square A''$.

Here, $C \not\rightarrow \square B$, $F \not\square C \not\square \square B$, $A \not\square C \not\rightarrow B$ have to be understood as pieces of default knowledge available to the agent for interpreting the chain of reported facts $\square B, (\text{in context } C), A_r, B_{r+1}$, together with the persistence law $\square B, \not\square C \not\rightarrow \square B_{r+1}$ (which can be deduced from $C \not\rightarrow \square B$, and $\square B, \not\square \square B_{r+1}$). In such a case, $A_r$ may indeed appear to the agent as being a cause for the change from $B_r$ to $B_{r+1}$, since $C \not\rightarrow \square B_r$ and $A_r \not\square C \not\rightarrow B_{r+1}$, entail $A_r \not\square \square B, \not\square C \not\rightarrow B_{r+1}$. With a different chain of reported fact, such as $\square B_r, (\text{in context } C), A_r, \not\square B_{r+1}$, the agent would be led to conclude that action $A_r$ has failed to produce its normal effect for some unknown reason.

**Example** When driving (context C) it is known that one has generally no accident ($\neg B$), i.e. $C \not\rightarrow \square B$, this is no longer true when driving being drunk ($F$), which is not safe w. r. t. accident (namely $F \not\square C \not\square \square B$, and driving being drunk in a speedy way ($G$) leaves you to have an accident ($B$) usually, i.e. $A \not\square C \not\rightarrow B$, with $A = F \not\square G$.

Note that the second situation imagined by von Wright could be represented by $B_r \not\square C \not\rightarrow B_{r+1}$, and $B_r \not\square A \not\square C \not\rightarrow B_{r+1}$. Letting $C' = B_r \not\square C$, this can be rewritten $C' \not\rightarrow \square B_{r+1}$, and $A \not\square C' \not\rightarrow B_{r+1}$, which is formally definition 2.

Clearly, the above definitions take advantage of the nonmonotonic behavior of $\not\rightarrow$, which allows different, although related, contexts to lead to opposite results, e.g., depending on the situation, the same action may sometime lead to different consequences, preventing an event from happening, or on the contrary making it happen.

### 3.2 Abnormality and causation

Moreover, one may consider in the above example that being drunk is a priori abnormal. This is an extra consideration that may lead a user to consider not only that being drunk when driving facilitates the occurrence of accident, and that driving in a speedy way when one is drunk can be the cause of an accident, but since being drunk is abnormal, this should be regarded as one of the main causes of an accident occurring in this case.

Indeed, since $C \not\rightarrow \square F$ and $C \not\rightarrow F$ are always inconsistent [17], there remain three cases for a facilitating factor F w. r. t. context C to be examined, the last one being in fact impossible as you will see:

- $C \not\rightarrow \square F$ and $C \not\rightarrow F$ i.e., $F \text{ is neither normal nor abnormal}$;
- $C \not\rightarrow \square F, i.e., F \text{ is an abnormal circumstance}$, which makes it more interesting from an explanation point of view;
- $C \not\rightarrow F$ i.e., $F \text{ is normal}$, but in such a case this is inconsistent with $C \not\rightarrow \square B$ and $F \not\square C \not\square \square B$, due to the cautious monotony [17] of $\not\rightarrow$, which forces $C \not\rightarrow F, C \not\rightarrow B \not\square C \not\rightarrow \square B$.

This expresses the following intuitively satisfying property: Something normal in a context cannot facilitate a change in the normal course of things for this context (a fortiori, something normal cannot be a cause for an observed unexpected change).

### 3.3 Explanation vs. causation

Note that one may think of a slightly different view of causation, weaker than Definition 2., called ‘Justification’ Namely

**Definition 3** Given a nonmonotonic consequence relation $\rightarrow$, if $C \not\rightarrow \square B$ and $A \not\square C \not\rightarrow B$, then A is said to *justify B* in context $C$.

This is situation ii) encountered in 3.1.1, hence the use of the name “justification” for distinguishing it from definition 2. However, in face of facts C, $\square B_r, A_r, B_{r+1}$, an agent knowing C $\not\square B$ and $A \not\square C \not\rightarrow B$ may perhaps doubt that the change from $\square B_r, B_{r+1}$ is really due to $A_r$, although the latter is indeed the very reason for his believing $B_{r+1}$. Indeed situation $\square B_r$ at time $t$ appears to be contingent, hence unknown to the agent, since it is not the normal course of things in context C: the agent is actually aware of both $C \not\rightarrow \square B$ and $C \not\rightarrow \square B$ (rather than $C \not\rightarrow \square B$, otherwise definition 2 would apply)$^1$. On the contrary, the case when $C \not\rightarrow \square B$ and $A \not\square C \not\rightarrow B$ captures an idea of causality according to which the agent may legitimately believe that A caused B to become true (1st situation a la Von Wright).

In a nutshell, the case whereby $C \not\rightarrow \square B, C \not\rightarrow \square B$ and $A \not\square C \not\rightarrow B$ cannot be interpreted as the observation of a causal phenomenon by an agent: all we can say is that reporting A caused the agent to start believing B, not that the agent believes that A caused B.

$^1$ Indeed, $C \not\rightarrow \square B_t$ would be necessary to be able to deduce $\square B_t \not\square C \not\rightarrow \square B_{t+1}$ from $\square B_t \not\rightarrow \square B_{t+1}$. 

25
What we call justification is akin to the notion of explanation following Spohn [31]: namely, “A is a reason for B” if raising the epistemic rank for A raises the epistemic rank for B. Gärdenfors [10] captured this view to some extent, assuming that A is a reason for B, if B is not retained in the contraction of A. Williams et al. [33] could account for the Spohnian view in a more refined way using kappa-rankings and transmutations, distinguishing between weak and strong explanations. Since our framework has possibilistic semantics it could properly account for this line of thought, even if our distinction between perceived causation and epistemic justification is not the topic of the above works.

Let us come back to Def. 2 of causation. Observe that such a view of causality, also agrees with the idea that causality is not coherent with classical entailment. Indeed if A causes B and B logically entails B’ (i.e. B |= B’), one cannot say that A causes B’, since while A ⊨ C ⊨ B’ holds by right weakening of A ⊨ C ⊨ B, C ⊨ ¬B’ does not hold generally, since ¬B’ entails ¬B).

Moreover, definition 2 does not make causality transitive. Indeed, in a given context, A causes B and B causes D do not entail A causes D, as it can be checked:

\[ C \models \neg B \text{ and } A \models C \models B \text{ and } C \models \neg D \text{ and } B \models C \models \neg D \text{ do not entail } C \models \neg D \text{ and } A \models C \models \neg D, \text{ by lack of transitivity of } \models. \]

Besides, according to definition 2, if A’ |= A, A causes B does not entail A’ causes B, since C |= D and A |= C |= B does not entail A’ |= C |= B (when A’ |= A). This is now illustrated.

Example “Throwing a stone in a window (A) crashes it” does not entail that “Throwing a small stone in a window (A’) crashes it”. Note that one cannot either conclude that “Throwing a big stone in a window (A’) crashes it”, due to the extreme cautiousness of |-. Using the rational entailment [17] instead of the preferential one would accommodate this situation, if some knowledge about the harmlessness of small stones is added.

Lastly, due to the left equivalence property of |-, one has that “A causes C” entails “B causes C” if A ≠ B. This might be troublesome in some cases. Suppose that is part of our knowledge that each time action A takes place, action B also takes place (and only in this case). Then, it may be debatable to regard the effects of A as being also those of B. However, observe that when time is taken into account, it would be easy to rule out the cases where At = Bt+1. The remaining situation At = Bt is more mysterious for such acts that cannot be causally related, if Bt is not just another name for At. In other words, left equivalence is not a problem if the equivalence is understood as a mere consequence of the axioms of propositional calculus. It is more problematic if applied to equivalences induced by additional knowledge about the world, so that A and B may refer to distinct objects having the same behavior.

### 3.4 The bipolarity of imprecise causal links

The lack of agreement of causality with logical entailment can be seen on the following example, which also shows problems posed by the description of imprecise causality relations. Assume that “Disease X causes a fever necessarily in interval \( I = [38°, 39°] \)”, with the intended meaning that the values of temperature apart from \( I \) are incompatible with disease X and where all the values in \( I \) are compatible with the possibility of disease X. Let us consider an interval \( I' = [38°, 40°] \) containing \( I \). Though “fever in \( I' \) implies “fever in \( I \)”, one may be reluctant to say that “disease X causes fever in \( I' \)”. On the contrary, “disease X explains fever” only in intervals which are subsets of \( I \), e.g. [38°, 38.5°]. This suggests that the relation of causality should work in some sense in the opposite way w. r. t. the relation of logical entailment, as observed by Besnard and Cardier [1]. This is also related to the paradoxical question of the modelling of disjunctive effects, when the disjunction reflects uncertainty about the effects. This leads to state

**Definition 4** A explains \( B' \) if \( B' \models B \) and A causes \( B \) (in the sense of Definition 2).

However, according to Def. 2, if \( B' \models B \), A causes \( B \) does not entail A causes \( B' \), since one has not always \( A \models C \models B' \). Indeed, Definition 2 only reflects one half of the above view of causality, namely the second condition in Definition 2 just expresses that in context C, any instance of A leads (normally) to a world where B is true. But it does not ensure that any model of B can be reached as an actually observable effect. The second half of what a causal relation should express according to what is suggested by the above example, would correspond to a statement asserting that any model of B is for sure a potential effect.

This two-sided view of the causal relation “A causes B”, agrees with a bipolar modeling [9] of its meaning, namely

i) The counter-models of B are not possible effects of A;

ii) Any model of B is an effect guaranteed to be possible.

This two-sided view of causality can be grasped in the possibility theory setting, in agreement with the nonmonotonic entailment |-, by means of the three constraints \( N(\neg B|C) > 0 \), \( N(B|A \models C) > 0 \), and \( N(\models B|A \models C) > 0 \), where \( N \) and \( \models \) are a necessity and a guaranteed possibility measure [9] respectively. While a necessity measure \( N(A) \) focuses on the most possible world where \( A \) is false, \( \models(A) \) focuses on the least possible worlds where \( A \) is true. In fact, \( N(\neg B|C) > 0 \) and \( N(B|A \models C) > 0 \) respectively provide a semantics for \( C \models \neg B \) and \( A \models C \models B \); see, e.g., [6]. A guaranteed possibility measure \( \models \) is such that \( N(\models E) \leq N(F|E) \) if \( F \models G \), i.e. \( \models \) works backwards w. r. t. classical entailment. This is a topic for further research.

### 4 Conclusion

The modeling of causality in A. I. should take into account problems raised by the incompleteness of available information and its level of granularity. The proposed view of perceived causality is built from the ideas of (ab)normality, non-monotonicity and belief revision. In no way, it should be regarded as an alternative proposal for approaching causality in A.I., but rather as complementary to other recently proposed approaches as focusing on particular issues neglected by them. Its cognitive plausibility still has to be checked. The approach has also to be developed in many directions, including its relation to updating and to logics of action, to the modelling of agents’ intentionality and to argumentation aspects of causal explanations.
Acknowledgments
The authors wish to thank Leila Amgoud for useful discussions and for presenting a preliminary version of this paper at the summerschool on "Causality, Uncertainty, and Ignorance", in Konstanz, on Aug. 15-21, 2004. The paper also benefited from remarks by Philippe Besnard and Florence Dupin de Saint-Cyr.

References
Cumulative Effects of Concurrent Actions on Numeric-Valued Fluents

Esra Erdem
Institute of Information Systems,
Vienna University of Technology, Vienna, Austria
esra@kr.tuwien.ac.at

Alfredo Gabaldon
National ICT Australia
Sydney, Australia
alfredo.gabaldon@nicta.com.au

Abstract

We propose a situation calculus formalization of action domains that include numeric-valued fluents (so-called additive or measure fluents) and concurrency. Our approach allows formalizing concurrent actions whose effects increment/decrement the value of additive fluents. For describing indirect effects, we employ mathematical equations in a manner that is inspired by recent work on causality and structural equations.

1 Introduction

In this paper we study the problem of formalizing action domains that include numeric-valued fluents and concurrency, in the situation calculus [McCarthy, 1963]. These fluents, known as additive fluents [Lee & Lifschitz, 2003] or measure fluents [Russel & Norvig, 1995], are used for representing measurable quantities such as weight or speed. An obvious practical application of reasoning about additive fluents is planning with resources, which usually are measurable quantities whose value is incremented/decremented by the execution of actions.

The ability to build plans in concurrent domains with numeric-valued fluents is crucial in real world applications. However, there has not been much work on formal accounts of this problem. Although there are several planning systems designed to work in concurrent domains with resources,1 most of them simplify the problem by requiring that concurrent actions be serializable. That is, actions are allowed to execute concurrently as long as their effect is equivalent to the effect of executing the same actions consecutively. This assumption eliminates practically all the semantic issues of the problem. On the other hand, this requirement precludes planners from solving many interesting problems. Consider for instance a simple problem where there are two resources $R_1$, $R_2$ and actions $A$, $B$ such that $A$ consumes one unit of $R_1$ and produces one unit of $R_2$, and $B$ consumes one unit of $R_2$ and produces one of $R_1$. Suppose also that there is the constraint $R_1 > 0$ at all times, and that they are initially $R_1 = R_2 = 1$. The simple plan consisting of the concurrent execution of $A$ and $B$ is not serializable, hence out of the scope of most planning systems.

In addition to a more general account of concurrency with additive fluents, we are also interested in allowing certain forms of indirect effects of actions (ramifications) on additive fluents. For instance, when a robot adds some water into a small container, and the water overflows into a larger container, the increment in the amount of water in the large container can be viewed as an indirect effect of the robot’s action. Given that the total amount of water in both containers is preserved, one may want to capture indirect effects by means of a mathematical equation. However, one is immediately confronted with a problem similar to the problem that led to the introduction of explicit notions of causality in action theories (see [Lin, 1995; McCain & Turner, 1995; Thielcher, 1997] among others): mathematical equations are symmetric and thus cannot express the causal relationship among the fluents in the equation. In this paper we present a formalization of indirect effects of concurrent actions on additive fluents. Our approach is in some respects based on the work of [Iwasaki & Simon, 1986; Halpern & Pearl, 2001] on causal reasoning with structural equations.

Our formalization for reasoning about the effect of concurrent actions on additive fluents builds on the work of [Reiter, 2001] and [Lee & Lifschitz, 2003]. We generalize Reiter’s basic action theories in the concurrent situation calculus [Reiter, 2001] with an account of additive fluents that is inspired on [Lee & Lifschitz, 2003], which, on the other hand, restricts additive fluents to range over finite sets of integers and does not consider the kind of indirect effects of actions on these fluents that we do.

2 Action Theories with Additive Fluents

We axiomatize action domains as basic action theories of the concurrent situation calculus [Reiter, 2001]. In addition to situations, objects, and primitive actions, the concurrent situation calculus includes a sort for concurrent actions, which are treated as sets of primitive actions. For a detailed description of the language and of basic action theories we refer the reader to [Reiter, 2001].

Additive fluents in the situation calculus are functional fluents that take numerical values, usually within a range. We

---

1[Koehler, 1998; Rintanen & Jungholt, 1999; Kvarnström, Doherty, & Haslum, 2000; Bacchus & Ady, 2001; Do & Kambhampati, 2003] are recent examples.
will assume that for each additive fluent \( f \), a range constraint \([L_f, U_f]\) is given, meaning that in every situation \( s \), \( L_f \leq f(s) \leq U_f \). These range constraints will usually be treated as qualification constraints, i.e., as additional action preconditions. Later when we consider indirect effects, we will see how these constraints also play a role there.

2.1 Direct effect axioms

For describing direct effects of actions on additive fluents, we introduce a function \( contr_f(x, a, s) \) for each additive fluent \( f \). Intuitively, \( contr_f(x, a, s) \) is the amount that the action \( a \) contributes to the value of \( f \) when executed in situation \( s \).

According to [Reiter, 1991], successor state axioms for functional fluents are sometimes derived from effect axioms of the form \( \gamma(x, v, a, s) \models f(x, do(a, s)) = v \). Similarly, we describe the effects of primitive actions on additive fluents by axioms of the form:

\[
\kappa_f(x, v, a, s) \supset contr_f(x, a, s) = v
\]

where \( \kappa_f(x, v, a, s) \) is a first-order formula whose only free variables are \( x, v, a, s \), does not mention function \( contr_f \) for any \( g \), and \( s \) is its only term of sort situation. For instance, when a robot \( r \) dumps a container \( B \) with \( n \) liters of water, this action causes its contents to decrease by \( n \):

\[
(\exists r)[a = \text{damp}(r) \land n = -\text{B}(s)] \supset contr_B(a, s) = n.
\]

From such effect axioms, we intend to derive successor state axioms for additive fluents by the same kind of transformation in [Reiter, 1991], which is based on an explanation closure assumption.

2.2 Successor state axioms

The effect axioms (1) describe the effects of atomic actions on additive fluents. We can obtain successor state axioms for these fluents in the concurrent situation calculus as follows.

As a first step, similar to how effect axioms for regular fluents are handled in [Reiter, 2001], we assume that if a primitive action has an effect on an additive fluent, then there is one effect axiom of the form (1) describing this effect, and that otherwise the effect of the action is to contribute zero to the additive fluent. This assumption allows us to derive a definition of the following form for each function \( contr_f \):

\[
contr_f(x, a, s) = v \equiv \kappa_f(x, v, a, s) \lor v = 0 \land \neg(\exists v')[\kappa_f(x, v', a, s)]
\]

Frequently the formula \( \kappa_f(x, v, a, s) \) is a disjunction of the form \( a = \alpha_1 \land \kappa_{\alpha_1, f}(x, v_1, a, s) \lor \ldots \lor a = \alpha_k \land \kappa_{\alpha_k, f}(x, v_k, a, s) \). When this is the case, we write axiom (2) as follows:

\[
contr_f(x, a, s) = \begin{cases} 
v_1 & \text{if } a = \alpha_1 \land \kappa_{\alpha_1, f}(x, v_1, a, s) \\
v_k & \text{if } a = \alpha_k \land \kappa_{\alpha_k, f}(x, v_k, a, s) \\
0 & \text{otherwise}
\end{cases}
\]

Example 1 Consider the missionaries and cannibals problem with two boats. The number of missionaries \( Mi \) or cannibals \( Ca \) at Bank1 or Bank2 of the river is described by the additive fluent \( num(g, l, s) \). The action of crossing the river is described by \( cross(b, l, num, nc) \) ("\( num \) number of missionaries and \( nc \) number of cannibals are crossing the river by boat \( b \) to reach the location \( l' \)).

The only action in the domain, \( cross \), has a direct effect on the additive fluent \( num \):

\[
\begin{align*}
contr_{num}(g, l, a, s) = & \\
\quad \begin{cases} 
n_1 & \text{if } (\exists b, n_2) a = cross(b, l, n_1, n_2) \land g = Mi \\
n_2 & \text{if } (\exists b, n_1) a = cross(b, l, n_1, n_2) \land g = Ca \\
-\kappa_1 & \text{if } (\exists b, n_2, l') a = cross(b, l', n_1, n_2) \land \quad g = Mi \land l \neq l' \\
-\kappa_2 & \text{if } (\exists b, n_1, l') a = cross(b, l', n_1, n_2) \land \quad g = Ca \land l \neq l' \\
0 & \text{otherwise}
\end{cases}
\end{align*}
\]

Once the axioms defining \( contr_f \) are in place, the successor state axioms for additive fluents are straightforward to write. What remains is to add up the contributions of all the primitive actions in a concurrent action. Such a sum defines the following function:

\[
cContr_f(x, c, s) = \sum_{a \in c} contr_f(x, a, s).
\]

The successor state axiom for each additive fluent \( f \) is

\[
f(x, do(c, s)) = f(x, s) + cContr_f(x, c, s).
\]

Example 2 Consider again the missionaries and cannibals problem of Example 1. The location of a boat \( b \) is described by the non-additive functional fluent \( \text{loc}(b, s) \). For this fluent, the successor state axiom is of the usual form:

\[
\text{loc}(b, do(c, s)) = l \equiv (\exists n_1, n_2)(\text{cross}(b, l, n_1, n_2) \in c) \lor \quad \neg(\exists n_1, n_2, l')(\text{cross}(b, l', n_1, n_2) \in c) \land \text{loc}(b, s) = l.
\]

For the additive fluent \( num \), the successor state axiom is of the form (3):

\[
\text{num}(g, l, do(c, s)) = \text{num}(g, l, s) + cContr_{num}(g, l, c, s).
\]

2.3 Action preconditions

In a basic action theory as described above, a concurrent action is possible only if each of its primitive actions is possible. However, a set of primitive actions each of which is individually possible may be impossible when executed concurrently. To handle such cases, we describe the conditions under which the primitive actions in \( c \) conflict with each other, denoted by \( conflict(c) \), and require their negation as additional preconditions of \( c \). For example, in the blocks world, a concurrent action containing the two primitive actions \( \text{stack}(x, z) \) and \( \text{stack}(y, z) (x \neq y) \) has a conflict, denoted by:

\[
\text{conflict}(c) \overset{\text{def}}{=} (\exists x, y, z)[\text{stack}(x, z) \land \text{stack}(y, z) \in c \land x \neq y],
\]

so we include \( \neg\text{conflict}(c) \) as a precondition for \( c \).

Another requirement for a concurrent action to be possible is that it must result in a situation that satisfies the range constraints on additive fluents. We use \( RC(s) \) to denote the conjunction of the range constraints on each additive fluent \( f \):

\[
\bigwedge_{f} \quad L_f \leq f(s) \leq U_f.
\]
conjoined with additional qualification constraints if given (see Example 3).

For additive fluents, most conflicts are covered by treating the range constraints as a precondition. For example, suppose that there is a fluent \( f \), with the range constraint \([0, 10]\) and the initial value \( f(S_0) = 5 \), and actions \( A \) which doubles the current value of \( f \) when executed and \( B \) which contributes 5 to \( f \). Due to the range constraint, although each action is possible in \( S_0 \), the concurrent action \( \{A, B\} \) is not. On the other hand, actions that set additive fluents to absolute values are an exception. A concurrent action that includes an action that sets the value of a fluent, e.g., “dump bucket;” and an action that contributes to the same fluent, e.g., “pour into bucket,” has a conflict that needs to be encoded explicitly by conflict(\(c\)).

To exclude both sorts of conflicting cases among actions, we include in an action theory a precondition axiom of the form

\[
\text{Poss}(c, s) \equiv (\exists a)(a \in c) \land (\forall a \in c)\text{Poss}(a, s) \land \\
\neg\text{conflict}(c, s) \land R^1[\text{RC}(do(c, s))].
\] (4)

Here, \( R^1[W] \) is a formula equivalent to the result of applying one step of Reiter’s regression procedure [Reiter, 1991]. Intuitively, by applying one regression step we obtain a formula that is relative to \( s \) and is true iff \( W \) is true in \( do(c, s) \). If the regressed formula holds in \( s \), it is guaranteed that, after executing \( c \), the constraints \( RC \) will hold. Regressing \( W \) is necessary in order to obtain an axiom of the form \( \text{Poss}(c, s) \equiv \Pi(c, s) \) where \( \Pi(c, s) \) is a formula whose truth value depends on situation \( s \) and on no other situation.

A single primitive action \( A \) can be viewed as a singleton concurrent action \( \{A\} \). Thus, reasoning about executable sequences is done in terms of concurrent actions only [Reiter, 2001]:^2

\[
\text{executable}(s) \equiv (\forall c, s')(do(c, s') \subseteq s \supset \text{Poss}(c, s')).
\]

**Example 3** Continuing with the axiomatization of the missionaries and cannibals problem, given the capacity of each boat by a situation independent function capacity\((b)\), we have the following precondition axiom for cross:\(2\)

\[
\text{Poss}(\text{cross}(b, l, n_1, n_2), s) \equiv \text{loc}(b, s) \neq l \land n_1 + n_2 \neq 0 \land n_1 + n_2 \leq \text{capacity}(b).
\]

One possible conflict we must consider is two cross actions to different locations but with the same boat:

\[
\text{conflict}(c, s) \equiv (\exists l_1, l_2)(l_1 \neq l_2) \land (\exists b, n_1, n_2)\text{cross}(b, l, n_1, n_2) \in c \land \\
\text{cross}(b, l_1, n_1, n_2) \in c.
\]

In this example, the constraints \( RC(s) \) are more interesting than just upper and lower bounds on the additive fluents, since there are additional constraints on the numbers of missionaries relative to cannibals: missionaries must not be outnumbered by cannibals. We include the following constraint:

\[
\text{RC}(s) \equiv (\exists g)(\text{num}(C_0, l, s) > \text{num}(M_0, l, s) \land \\
\text{num}(M_0, l, s) > 0) \land \\
(\forall g, l)(0 \leq \text{num}(g, l, s) \leq \text{MaxNumber}).
\]

The constant MaxNumber is the upper bound on fluent num for both cannibals and missionaries.

### 3 Ramification Constraints on Additive Fluents

A domain that does not contain any actions with ramifications can be described as an action theory in the concurrent situation calculus as discussed in the previous sections. How do we describe in the concurrent situation calculus a domain that contains an action with indirect effects on some fluents? In this section we provide an answer to this question for a particular representation of ramifications.

**Example 4** Suppose that we have a small container and a large container for storing water. The small container is suspended over the large container so that, when the small container is full of water, the water poured into the small container overflows into the large container. Suppose also that there are three taps: one directly above the small container, by which some water can be added to the containers from an external source, one on the small container, by which some water can be released from the small container into the large container, and a third tap on the large container to release water to the exterior. We want to formalize this domain in the concurrent situation calculus.

The amount of water in the small and the large containers is represented by the additive fluents: small\((s)\) and large\((s)\). Another additive fluent, total\((s)\), represents the total amount of water in the containers.

We introduce the action add\((n)\) to describe the action of adding \( n \) liters of water to the containers by opening the tap over them, and the actions releaseS\((n)\) and releaseL\((n)\) by axioms of form (1). The action add\((n)\) contributes directly to total:

\[
(\exists n)[a = \text{add}(n) \land v = n] \supset \text{contr}_{\text{total}}(a, s) = v
\]

and to small:

\[
(\exists n)[a = \text{add}(n) \land v = n] \supset \text{contr}_{\text{small}}(a, s) = v.
\]

The action releaseS\((n)\) contributes directly to small:

\[
(\exists n)[a = \text{releaseS}(n) \land v = -n] \supset \text{contr}_{\text{small}}(a, s) = v.
\]

and to large:

\[
(\exists n)[a = \text{releaseL}(n) \land v = -n] \supset \text{contr}_{\text{large}}(a, s) = v. \tag{5}
\]

The action releaseL\((n)\) contributes to large:

\[
(\exists n)[a = \text{releaseL}(n) \land v = -n] \supset \text{contr}_{\text{total}}(a, s) = v.
\]

From these direct contribution axioms, we obtain definitional axioms of the form (2):

\[
\text{contr}_{\text{small}}(a, s) = \begin{cases} 
  n & \text{if } a = \text{add}(n) \\
  -n & \text{if } a = \text{releaseS}(n) \\
  0 & \text{otherwise}
\end{cases}
\]

\[^2\text{Intuitively, an expression } s \subseteq s' \text{ means that } s \text{ is a subsequence of } s'.\]
\[
\text{contr}_{\text{large}}(a, s) = \begin{cases} 
    n & \text{if } a = \text{release}S(n) \\
    -n & \text{if } a = \text{release}L(n) \\
    0 & \text{otherwise}
\end{cases}
\]
\[
\text{contr}_{\text{total}}(a, s) = \begin{cases} 
    n & \text{if } a = \text{add}(n) \\
    -n & \text{if } a = \text{release}L(n) \\
    0 & \text{otherwise}
\end{cases}
\]

3.1 Range constraints and ramification

In earlier sections, the range restrictions were treated as qualification constraints: if executing an action will falsify them, the action is consider impossible. In this example, however, the upper bound on the value of small plays a different role. Actions that seemingly would increase the value of small over \(U_{\text{small}}\) should not be considered impossible, but actually to increase its value up to \(U_{\text{small}}\).

This fact will be captured explicitly in the definition of the concurrent contribution of actions to small as follows:

\[
c\text{Contr}_{\text{small}}(\vec{x}, c, s) = \begin{cases} 
    U_{\text{small}} - \text{small}(s) & \text{if } \text{sum}_{\text{small}} > U_{\text{small}} - \text{small}(s) \\
    \text{sum}_{\text{small}} & \text{otherwise}
\end{cases}
\]

where \(\text{sum}_{\text{small}}\) stands for \(\sum_{a \in c} \text{contr}_{\text{small}}(\vec{x}, a, s)\).

In general, functions \(c\text{Contr}_{f}\) are defined as follows:

\[
c\text{Contr}_{f}(\vec{x}, c, s) = \begin{cases} 
    U_{f} - f(\vec{x}, s) & \text{if } \text{sum}_{f} > U_{f} - f(\vec{x}, s) \\
    L_{f} - f(\vec{x}, s) & \text{if } \text{sum}_{f} < L_{f} - f(\vec{x}, s) \\
    \text{sum}_{f} & \text{otherwise}
\end{cases}
\]

where \(\text{sum}_{f}\) stands for \(\sum_{a \in c} \text{contr}_{f}(\vec{x}, a, s)\), and the first two lines in the right-hand side being present only if the range restriction \(U_{f}\), resp. \(L_{f}\), are a source of ramifications. Note that if the range restrictions play no role in ramifications and the two lines are thus missing, the definition of \(c\text{Contr}_{f}\) is just as shown earlier.

3.2 Contribution equations

The next question in formalizing the ramifications is how to describe the causal influence among the fluents. In our water container example, the relation among the fluents could be described by the equation:

\[
total(s) = \text{small}(s) + \text{large}(s)
\]

which must hold in all situations \(s\). However, this equation does not capture the arrangement of the containers that makes water flow from the small container into the large one. The reason is clear: such algebraic equations are symmetric and are not meant to describe how changes in one fluent causally influence other fluents in the equation.

Causal reasoning with equations has been considered before in Al. [Iwasaki & Simon, 1986] (subsequently IS) considers the problem of making explicit the causal relation among variables in an equation describing a mechanism—a component of a device or system. IS assumes each mechanism is described by a single structural equation describing how variables influence other variables. [Halpern & Pearl, 2001] (subsequently HP) also uses structural equations, in this case with the purpose of representing causal relations among random variables for modeling counterfactuals.

Our approach to handling indirect effects on additive fluents has been influenced by IS and HP. In order to represent indirect effects on fluents, we will use equations in a similar fashion as structural equations are used in the aforementioned work to describe causal influence among variables. We use structural equations under mainly four assumptions.

1. Similarly to IS and HP, we assume that each equation represents a single mechanism. That is, an equation describes the indirect contribution of actions to one fluent in terms of the contribution to the value of the other fluents in the equation.

2. Both IS and HP require each variable to be classified as either exogenous or endogenous. This is reasonable for the settings they consider where there is no agent intervening with the mechanism represented by the equation. All external intervention is fixed a-priori, which allows classifying variables this way. In our case, external intervention\(^3\) depends on what particular action is executed. Hence a fluent may be exogenous (directly affected) with respect to one primitive action and endogenous with respect to another primitive action, with both actions occurring concurrently. Thus, in our approach we do not assume that fluents can be separated into exogenous and endogenous classes.

3. We do not intend to derive a causal ordering among fluents as IS does for variables. We assume, as done in HP and recent work on causality [Lin, 1995; McCain & Turner, 1995; Thielscher, 1997], that the causal relation among fluents is explicit in the axioms describing the indirect contributions of actions.

4. We assume, as IS and HP do, that the causal influence among fluents is acyclic. Lifting this assumption remains a topic for future work.

With these assumptions, suppose that, in axiomatizing our domain, for describing each causal mechanism, we provide an equation similar to the structural equations in HP: for a fluent \(f\), the equation would have the form \(f = \mathcal{E}\) where \(\mathcal{E}\) is an expression in terms of the fluents on which \(f\) causally depends. In the case of additive fluents, such an expression is in fact a linear combination of functions. Just as the structural equations in HP, an equation such as \(f = \mathcal{E}\) is asymmetric in the sense that the equation determines the value of \(f\) but not the value of any of the fluents in the right-hand side. We use such equations, however, not to compute the value of fluents, but to compute the contribution to the value of the fluents that results from executing an action. From an equation \(f = \mathcal{E}\), we obtain an almost identical equation but instead of written in terms of fluent functions, written in terms of functions \(c\text{Contr}_{f}\) and an abbreviation \(i\text{Contr}_{f}(\vec{x}, c, s)\) for each fluent \(f\) that intuitively denotes the amount that action \(c\) indirectly contributes to \(f\) in situation \(s\).

If an equation describing indirect effects on a fluent \(f\) is not given, then

\[
i\text{Contr}_{f}(\vec{x}, c, s) \overset{\text{def}}{=} 0.
\]
Otherwise, suppose that equation $f = E(f_1, \ldots, f_n)$ is given, where $E(f_1, \ldots, f_n)$ is a linear combination of fluents $f_1(\vec{x}_1, s), \ldots, f_n(\vec{x}_n, s)$. Then we define $iContr_f(\vec{x}, c, s)$ as follows:

$$
iContr_f(\vec{x}, c, s) \overset{\text{def}}{=} E(cContr_f(\vec{x}_1, c, s), \ldots, cContr_{f_n}(\vec{x}_n, c, s)) - cContr_f(\vec{x}, c, s)
$$

**Example 5** Consider Example 4. Suppose that the mechanism of containers is described by the contribution equation

$$large(s) = total(s) - small(s) \quad (6)$$

and the range restrictions on the fluents are as follows:

$$L_{total} = L_{small} = L_{large} = 0,
U_{total} = 6, U_{small} = 2, U_{large} = 4.
$$

Any concurrent action whose total effect on the fluents results in a situation where these range restrictions are violated is impossible, in accordance with our axiom (4) for $Poss(c, s)$ described earlier, except for restriction $U_{small}$. If an action’s contribution to $small$ will result in a larger value than its upper bound $U_{small}$ allows, the action is not rendered impossible, but instead has an indirect effect. The indirect effect of increasing $small$ too much is expressed by the following equation which can be obtained from the contribution equation $(6)$:

$$iContr_{large}(c, s) \overset{\text{def}}{=} cContr_{total}(c, s) - cContr_{small}(c, s) - cContr_{large}(c, s).
$$

Given the initial values

$$total(S_0) = 2, small(S_0) = 1, large(S_0) = 1,
$$

and the concurrent action

$$c = \{add(6), releaseL(1), releaseL(2)\},
$$

we obtain:

$$cContr_{total}(c, S_0) = 4, \quad iContr_{total}(c, S_0) \overset{\text{def}}{=} 0,
$$

$$cContr_{small}(c, S_0) = 1, \quad iContr_{small}(c, S_0) \overset{\text{def}}{=} 0,
$$

$$cContr_{large}(c, S_0) = -1, \quad iContr_{large}(c, S_0) \overset{\text{def}}{=} 4.
$$

### 3.3 Successor state axiom with indirect effects

After defining direct and indirect contributions of actions on an additive fluent $f$, there only remains to define the successor state axiom for such a fluent. We define such an axiom as follows:

$$f(\vec{x}, do(c, s)) = f(\vec{x}, s) + iContr_f(\vec{x}, c, s)
$$

where

$$iContr_f(\vec{x}, c, s) \overset{\text{def}}{=} cContr_f(\vec{x}, c, s) + iContr_f(\vec{x}, c, s).
$$

This axiom replaces $(3)$ in domain axiomatizations.

**Example 6** For our container example with the values from Example 5, we obtain $total(c, S_0) = 2 + 4 = 6, small(c, S_0) = 1 + 1 = 2, and large(c, S_0) = 1 + 3 = 4.$

Once the contribution equation has been compiled into the theory, we can prove that the original, symmetric equation is satisfied in every situation provided it is satisfied initially. This can be done by a very simple application of the induction axiom:

$$(\forall P)[P(S_0) \land (\forall c, s)[P(s) \supset P(do(c, s))]] \supset (\forall s)P(s)
$$

with

$$P(s) \overset{\text{def}}{=} total(s) = small(s) + large(s).
$$

**Proposition** Let $D$ stand for the water container theory presented through out this section and $eq(s)$ stand for $total(s) = small(s) + large(s)$. Then,

$$D \models eq(S_0) \supset (\forall s)eq(s).
$$

### 3.4 An example with two mechanisms

Consider a variation of the container example where a container of medium size is inserted between the small container and the large container, so that, when the small container is full, the water poured into the small container overflows into the medium container, and, when the medium container is full, the water overflows into the large container. Here we consider two mechanisms: one consisting of the small container and the medium container, and the other consisting of this inner mechanism and the large container.

The amount of water in the medium container is represented by the additive fluent $medium(s)$. Another additive fluent, $inner(s)$, represents the total amount of water in the small container and the medium container.

We describe the direct contributions of the actions $add(n)$, $releaseS(n)$, and $releaseL(n)$ by the direct effect axioms of Example 4, except $(5)$, and the axioms

$$(\exists n)[a = add(n) \land v = n] \supset contr_{inner}(a, s) = v,
$$

$$(\exists n)[a = releaseS(n) \land v = n] \supset contr_{medium}(a, s) = v.
$$

Given the contribution equations

$medium(s) = inner(s) - small(s),
large(s) = total(s) - inner(s),$

one for each mechanism, the indirect effect of adding too much water to the small container is expressed by the following equations:

$$iContr_{medium}(c, s) \overset{\text{def}}{=} cContr_{inner}(c, s) - cContr_{small}(c, s)
$$

$$iContr_{large}(c, s) \overset{\text{def}}{=} cContr_{total}(c, s) - cContr_{inner}(c, s)
$$

In addition to the range restrictions on $small$ and $large$ as in Example 5, consider the following range restrictions

$$L_{medium} = L_{inner} = L_{total} = 0,
U_{medium} = 3, U_{inner} = 5, U_{total} = 9.$$

These must be consistent with the equation $(6)$. 


Suppose that initially all containers have 1 unit of water:
\[
\begin{align*}
small(S_0) &= medium(S_0) = large(S_0) = 1, \\
inner(S_0) &= 2, \\
total(S_0) &= 3,
\end{align*}
\]
and that the concurrent action
\[
c = \{\text{add}(8), \text{release}(1), \text{release}(2)\}
\]
is executed at the initial situation. Then we can compute the contributions of this action to each additive fluent:
\[
\begin{align*}
&cContr_{\text{small}}(c, S_0) = 1, \\
&cContr_{\text{medium}}(c, S_0) = 1, \\
&cContr_{\text{large}}(c, S_0) = -2, \\
&cContr_{\text{inner}}(c, S_0) = 3, \\
&cContr_{\text{total}}(c, S_0) = 6,
\end{align*}
\]
and the amount of water in each container:
\[
\begin{align*}
small(c, S_0) &= 1 + 1 + 2 = 4, & inner(c, S_0) &= 2 + 3 = 5, \\
medium(c, S_0) &= 1 + 2 = 3, & total(c, S_0) &= 3 + 6 = 8, \\
large(c, S_0) &= 1 + 3 = 4.
\end{align*}
\]

4 Conclusion

In this paper we introduced a formalization of additive fluents in concurrent domains that is based on Reiter’s basic action theories in the concurrent situation calculus. This formalization allows reasoning about the effect of actions that increment/decrement integer or even real valued fluents. Moreover, we presented an approach to reasoning about indirect effects through the use of equations that are conceptually similar to the structural equations of [Iwasaki & Simon, 1986; Halpern & Pearl, 2001]. To the best of our knowledge, this is the first attempt at formalizing ramifications of concurrent actions on numeric-valued fluents.

Our approach to ramifications on additive fluents based on equations that express the direction of causal influence explicitly, is in line with recent work on causality in theories of action [Lin, 1995; McCain & Turner, 1995; Thielscher, 1997]. In our equations, causal direction is made explicit by the use of function \(iContr\) on one side and the use of functions \(cContr\) on the other side of the equations.

After compiling such ramification constraints in the form of equations into the action theory, in the spirit of [Lin & Reiter, 1994; Lin, 1995; McIlraith, 2000], the constraints become logical consequences of the resulting theory (as shown by Proposition for our example).

Planning with concurrency and resources is currently a subject of intense research and we believe that a formal, logic-based account of the problem is an important contribution. The proposal we have put forward in this paper allows formalizing a much more general class of domains than current planning systems are designed to solve, and thus it is useful for specifying what is a correct solution to a planning problem in such generalized domains.

5 Acknowledgments

We are grateful to Fangzhen Lin for many stimulating discussions on the subject of this paper, including a variation of the container example. We are also grateful to Yves Lespérance for helpful comments on an earlier version of the paper.

References


Actions as Special Cases
(Preliminary Report)

Selim T. Erdoğan and Vladimir Lifschitz
Department of Computer Sciences
The University of Texas at Austin
1 University Station C0500
Austin, TX 78712-0233 USA
{selim,vl}@cs.utexas.edu

Abstract
This paper is motivated by the idea of interaction between two directions of research in knowledge representation: the design of action description languages and the development of libraries of reusable, general-purpose knowledge components. Writing an action description that characterizes actions in terms of their effects, as common today, can be compared to writing a program that does not use standard subroutines. We conjecture that a library of standard descriptions for a number of “basic” actions can facilitate writing, understanding and modifying action descriptions. To illustrate this idea, we show how the action PushBox in the Monkey and Bananas domain can be described as a special case of the “library action” Move.

1 Introduction
Research on describing actions started with the invention of STRIPS [Fikes and Nilsson, 1971] and ADL [Pednault, 1994] and led in recent years to the design of very expressive action languages, such as C+ [Giunchiglia et al., 2004]. The heart of every action language is a syntactic mechanism for describing effects of actions on fluents. When we define, for instance, the Monkey and Bananas domain in STRIPS, we can specify how pushing the box affects the location of the box by including appropriate atoms in the description of the operator PushBox(l): we put At(Box,l) for every location l on its delete list, and At(Box,l) on its add list. In C+ the same idea can be expressed by the causal law

\[ \text{PushBox}(l) \text{ causes } \text{Loc}(\text{Box}) = l \]  

(1)

(quoted from [Giunchiglia et al., 2004], Figure 2, reproduced in Section 4 below).

Descriptions like these are common in knowledge representation, but they are strikingly different from the descriptions of actions that humans give to each other informally. The dictionary says, for instance, that pushing is moving by steady pressure. This phrase explains the meaning of the word push not by listing the effects of this action, but by presenting it as a special case of another action, move, that is supposed to be already familiar to the reader. Some actions may need to be described directly in terms of the changes that they cause; to move, for instance, means to cause to change position, according to the dictionary. But in most cases the easiest way to describe an action is to relate it to more basic actions.

Here is one more example of describing one action as a special case of another. A surgeon may indicate the action he wants to be performed by saying, “Scalpel.” John McCarthy [1993] explains that in the context of an operation this one word may mean “Please give me the number 3 scalpel.” The action to be performed is described as a special case of the basic action give.

Writing a traditional action description that characterizes actions directly in terms of their effects can be compared to writing a program that does not use standard subroutines. A traditional action language is similar to a programming language that comes without a library of standard functions.

We would like to apply the idea of a library of reusable, general-purpose knowledge components [Barker et al., 2001] to the design of action languages. We conjecture that a library of standard descriptions for “basic” actions, such as move and give, can facilitate writing, understanding and modifying action descriptions. We conjecture that such a library can be written in a language based, syntactically and semantically, on C+.

In Section 2 we argue, on the other hand, that nondefinite action descriptions (see [Giunchiglia et al., 2004, Sec. 5]) are essential for this project. Since the Causal Calculator\(^1\) is an implementation of the definite fragment of C+, it will not be possible to use that system, at least directly, to process references to library action descriptions.

In Sections 3 and 4 we discuss a specific example illustrating the idea of using a library to specify action domains. We begin with a C+ description of the action Move that can be included, in principle, in a library of general-purpose action descriptions. Then we review the formalization of the Monkey and Bananas domain from [Giunchiglia et al., 2004] and show how to replace some of the C+ propositions in that formalization with a group of \(C+\) propositions that characterizes PushBox as a special case of Move. Finally, we state a theorem expressing the adequacy of this reformulation.

\(^1\)http://www.cs.utexas.edu/users/tag/ccalc/.
2 Expressing Synonymity in C+

If we have a library description of the effect of an action a on a fluent f, and we would like to describe an action domain that involves an action a', similar to a, affecting a fluent f', similar to f, then how can the library help us? Instead of describing the effect of a' on f' directly, we may be able to use a postulate relating a' to a and a postulate relating f' to f. In the simplest case, these postulates may say that a' is completely synonymous with a, and f' is completely synonymous with f.

Expressing the synonymy of actions and fluents in C+ involves a subtlety, and before discussing it we will review some of the ideas behind the design of that language.

The semantics of C+ is based on the distinction between the assertion “p is true” and the assertion “there is a cause for p,” advocated in [Geffner, 1990; Lin, 1995; McCain and Turner, 1997]. Propositions (causal laws) of C+ can be viewed as implications of the form “if F is true then there is a cause for G.” For instance, proposition (1) is understood to mean: if the action PushBox(l) is executed then there is a cause for the fluent Loc(Box) to have the value l in the next state.”

Besides propositions describing effects of actions on fluents, C+ has “static laws,” which describe causal relationships between fluents. They do not contain symbols for actions. Static laws have the form

\[
\text{caused } F \text{ if } G
\]

where F and G are fluent formulas (conditions on the values of fluents). The use of static laws allows us to describe indirect effects of actions. For instance, the formalization of the Monkey and Bananas domain reproduced in Section 4 below includes the static law

\[
\text{caused Loc(Bananas)} = l \text{ if HasBananas} \land \text{Loc(Monkey)} = l.
\]

(3)

It says that there is a cause for the bananas to be at l if the monkey has the bananas and is at l. In the presence of this static law, any action affecting the location of the monkey (such as walking or pushing the box) will affect the location of the bananas in the same way, provided that the monkey is holding the bananas.

Static laws can be used also for expressing assumptions about the default values of fluents. By including a causal law of the form

\[
\text{caused } F \text{ if } F
\]

(“if F is true then there is a cause for this”) in an action description we can express that, by default, F is assumed to be true.

Let us go back now to the problem of expressing the synonymy of a fluent f’ with a fluent f in C+. A natural idea would be to use the static laws

\[
\text{caused } f' = v \text{ if } f = v \text{ if } f' = v,\]

(5)

They say: if the value of f is v then there is a cause for f’ to have the same value; if the value of f’ is v then there is a cause for f to have the same value. But these causal laws have an unwanted implication: they make f = v and f’ = v “true by default,” just as (4) expresses a default assumption about F. For this reason, (5) is not an acceptable representation of the synonymy of f’ with f.

We will express that f’ is synonymous with f by postulating

\[
\text{caused } f = v \iff f' = v \text{ if } \top.
\]

for every v in the common domain of f and f’. These causal laws say: there is a cause for f and f’ to have the same value. They are not definite, because their heads f’ = v \iff f = v are not atoms. They can be combined into one static law:

\[
\text{caused } \bigwedge_v f = v \iff f' = v \text{ if } \top.
\]

Standard notational conventions allow us to abbreviate it by

\[
\text{caused } f = f'.\]

(6)

Recall that according to the semantics of C+, every action description represents a transition system—a directed graph whose vertices are states, and whose edges are labeled by events (see Section 3 for an example). Let D be an action description whose signature does not contain f’. Denote by D* the action description obtained from D by adding f’ as a fluent constant with the same domain as f, and adding (6) to its set of causal laws.3 The value of f’ in any state of D* is the same as the value of f in that state. Furthermore, D* is a “conservative extension” of D, in the sense that restricting the states of D* to the fluents different from f’ establishes an isomorphism between the transition system represented by D* and the transition system represented by D. Definite laws (5) do not have such a conservative extension property.

Similarly, the synonymy of an action constant a with an action constant a’ can be expressed by the action dynamic law

\[
\text{caused } a = a'.\]

It has properties similar to the two properties of (6) stated above.

The use of nondefinite causal laws like these will play an essential role when we turn to the problem of relating PushBox to Move in Section 4.

3 Moving Things

Our “general-purpose” formalization of the action Move is a family of C+ action descriptions depending on two parameters. For any nonempty finite sets of symbols P and L, the action description MOVE(P, L) below represents the properties of moving physical objects (elements of P) to locations (elements of L).

The signature and the causal laws of MOVE(P, L) are as follows:4

\[\text{Notation: } p, p_1 \text{ range over } P; l \text{ ranges over } L.\]

---

2This is similar to the difficulty with formalizing Denecker’s “two gears” domain [McCain, 1997, Section 7.5.5] in the definite fragment of causal logic.

3We assume that f’ is simple if f is simple, and statically determined if f is statically determined.

4A complete definition of the syntax and semantics of C+ can be found in [Giunchiglia et al., 2004, Section 4].
Simple fluent constants:
- Location(p)
- Move(p)
- Mover(p)
- Destination(p)

Domains:
- Location(p): L
- Move(p): Boolean
- Mover(p): P ∪ {None}
- Destination(p): L ∪ {None}

Causal laws:

always Mover(p) = None ⇒ ¬Move(p)
always Destination(p) = None ⇒ ¬Move(p)

Move(p) causes Location(p) = l if Destination(p) = l (7)
Move(p) causes Location(p1) = l
if Mover(p) = p1 ∧ Destination(p) = l (8)
nonexecutable Move(p)
if Location(p) = Destination(p) (9)
nonexecutable Move(p) if Mover(p) = p1
∧ Location(p1) ≠ Location(p) (10)

exogenous Move(p) (11)
exogenous Mover(p)
exogenous Destination(p)

inertial Location(p) (12)

The constants Mover(p) and Destination(p) are used here as attributes of the action Move(p) in the sense of [Giunchiglia et al., 2004, Section 5.6]. When the action Move(p) is executed, the value of Mover(p) is the agent executing that action, and the value of Destination(p) is the location to which p is being moved; otherwise the value of each attribute is None (“undefined”). Executing Move(p) causes the location of p and the location of Mover(p) to be equal to Destination(p). The action is not executable if Destination(p) is the current location of p, and also if p and Mover(p) are in different places.

Consider, for example, the transition system represented by the action description

\[ MOVE(\{\text{Monkey}, \text{Box}, \text{Bananas}\}, \{L_1, L_2, L_3\} \), (13)\]

(This choice of “actual parameters,” substituted for the “formal parameters” \( P, L \), corresponds to the use of MOVE in the next section.) This graph has 27 vertices, corresponding to the states—assignments of locations \( L_1, L_2, L_3 \) to fluents Location(Monkey), Location(Box) and Location(Bananas). Every edge of this graph is labeled by an event—an assignment of values to the action constants. In one of these events, for instance, the monkey is moving the box from \( L_2 \) to \( L_3 \), where the bananas are. The corresponding edge of the graph is shown in Figure 1.

4 Pushing the Box as a Special Case of Moving

Here are the signature and the causal laws of the action description MB, proposed in [Giunchiglia et al., 2004, Figure 2] as a description of the Monkey and Bananas domain in C+:

Notation: \( x \) ranges over \( \{\text{Monkey}, \text{Bananas}, \text{Box}\} \); \( l \) ranges over \( \{L_1, L_2, L_3\} \).

Simple fluent constants:
- Loc(x): \{L_1, L_2, L_3\}
- HasBananas, OnBox

Domains:
- Loc(x): Boolean
- HasBananas, OnBox: Boolean

Action constants:
- Walk(l), PushBox(l)
- ClimbOn, ClimbOff, GraspBananas

Causal laws:

causd Loc(Bananas) = l if HasBananas ∧ Loc(Monkey) = l (14)
causd Loc(Monkey) = l if OnBox ∧ Loc(Box) = l (15)

Walk(l) causes Loc(Monkey) = l
nonexecutable Walk(l) if Loc(Monkey) = l
nonexecutable Walk(l) if OnBox

PushBox(l) causes Loc(Box) = l (16)
PushBox(l) causes Loc(Monkey) = l (17)
nonexecutable PushBox(l) if Loc(Monkey) = l
nonexecutable PushBox(l) if OnBox

nonexecutable PushBox(l) if Loc(Monkey) ≠ Loc(Box) (18)

ClimbOn causes OnBox
nonexecutable ClimbOn if OnBox
nonexecutable ClimbOn if Loc(Monkey) ≠ Loc(Box)

ClimbOff causes ¬OnBox
nonexecutable ClimbOff if ¬OnBox

GraspBananas causes HasBananas
nonexecutable GraspBananas if HasBananas
nonexecutable GraspBananas if ¬OnBox
nonexecutable GraspBananas
if Loc(Monkey) ≠ Loc(Bananas)

nonexecutable Walk(l) ∧ PushBox(l)
nonexecutable Walk(l) ∧ ClimbOn
nonexecutable PushBox(l) ∧ ClimbOn
nonexecutable ClimbOff ∧ GraspBananas

exogenous Walk(l)
Its causal laws are

\[
\begin{align*}
\text{Move}(\text{Monkey}) &= I \\
\text{Mover}(\text{Monkey}) &= \text{None} \\
\text{Destination}(\text{Monkey}) &= \text{None} \\
\text{Move}(\text{Box}) &= t \\
\text{Mover}(\text{Box}) &= \text{Monkey} \\
\text{Destination}(\text{Box}) &= L_3
\end{align*}
\]

\[
\begin{align*}
\text{Move}(\text{Bananas}) &= I \\
\text{Mover}(\text{Bananas}) &= \text{None} \\
\text{Destination}(\text{Bananas}) &= \text{None} \\
\text{Location}(\text{Box}) &= L_3 \\
\text{Location}(\text{Bananas}) &= L_3
\end{align*}
\]

**Figure 1:** An edge of the graph represented by action description (13)

\[
\text{exogenous } \text{PushBox}(l) \quad (18)
\]

\[
\text{exogenous } \text{ClimbOn}
\]

\[
\text{exogenous } \text{ClimbOff}
\]

\[
\text{exogenous } \text{GraspBananas}
\]

\[
\text{inertial } \text{Loc}(x) \quad (19)
\]

\[
\text{inertial } \text{HasBananas}
\]

\[
\text{inertial } \text{OnBox}
\]

Action \text{PushBox} is a special case of \text{Move}, in which the object that is being moved is the box, the mover is the monkey, and the destination may be any one of the locations \(L_1, L_2, L_3\). On the right margin we assigned numbers to the causal laws of \(\text{MB}^*\) that have counterparts in \(\text{MOVE}(P, L)\). Our goal is to find a collection of causal laws relating \(\text{MB}\) to \(\text{MOVE}(P, L)\) that will make (14)–(19) redundant. Causal laws (14) and (15), describing the effects of \text{PushBox}, will become “special cases” of (7) and (8), which describe the effects of \text{Move}. Causal laws (16) and (17), describing some of the preconditions of \text{PushBox}, will become redundant in the presence of the general preconditions (9) and (10) of \text{Move}. (The third precondition of the action \text{PushBox}—the fact that it cannot be executed if the monkey is on the box—is domain-specific and has no counterpart in the “library description” from Section 3.) Finally, (18) and (19) will become redundant in the presence of (11) and (12).

Our reformulation \(\text{MB}^*\) of \(\text{MB}\) is defined as follows. Its signature is the union of the signature of \(\text{MB}\) with the signature of the instance (13) of the “library description” of \text{Move}. Its causal laws are

- the causal laws of \(\text{MB}\), except (14)–(19),
- the causal laws of (13), and
- the following causal laws, connecting (13) with \(\text{MB}\):

\[
\begin{align*}
\text{caused } \text{Location}(p) &= \text{Loc}(p) \\
\text{caused } \text{Move}(\text{Box}) &= \bigvee_l \text{PushBox}(l)
\end{align*}
\]

\[
\text{caused } \neg \text{Move}(p) \quad (p \neq \text{Box}) \quad (22)
\]

\[
\text{caused } \text{Mover}(\text{Box}) = \text{Monkey} = \text{Move}(\text{Box}) \quad (23)
\]

\[
\text{caused } \text{Destination}(\text{Box}) = l \equiv \text{PushBox}(l) \quad (24)
\]

where \(p\) ranges over \{\text{Monkey}, \text{Box}, \text{Bananas}\}, and \(l\) over \{\text{L}_1, \text{L}_2, \text{L}_3\}.

Proposition (20) says that \text{Location} is synonymous with \text{Loc}. Propositions (21) and (22) tell us that moving the box amounts to pushing it to some location, and that no object other than the box is ever moved. According to (23), the mover is the monkey whenever the box is being moved. According to (24), the destination is \(l\) whenever the box is pushed to \(l\).

Action description \(\text{MB}^*\) is not exactly equivalent to \(\text{MB}\), because its signature is different. A state of \(\text{MB}\) assigns values to the fluent constants

\[
\text{Loc}(p), \text{HasBananas}, \text{OnBox};
\]

a state of \(\text{MB}^*\) assigns values to all these constants and also to \text{Location}(p). An event of \(\text{MB}\) assigns values to the action constants

\[
\text{Walk}(l), \text{PushBox}(l), \text{ClimbOn}, \text{ClimbOff}, \text{GraspBananas};
\]

an event of \(\text{MB}^*\) assigns values to all these constants and also to

\[
\text{Move}(p), \text{Mover}(p), \text{Destination}(p).
\]

The theorem below shows, however, that the transition systems represented by \(\text{MB}\) and \(\text{MB}^*\) are isomorphic to each other. In this sense, our reformulation of \(\text{MB}\) based on the “toy library” from Section 3 is adequate.

About action descriptions \(D\) and \(D'\) we say that \(D\) is a residue of \(D'\) if the signature of \(D\) is a part of the signature of \(D'\), and restricting the states and events of \(D'\) to the signature of \(D\) establishes an isomorphism between the transition system for \(D'\) and the transition system for \(D\).

**Theorem** \(\text{MB}\) is a residue of \(\text{MB}^*\).

The proof of the theorem uses a general lemma that explains how transition systems are affected by the addition of
causal laws of a form similar to (20)–(24). By Dom(c) we denote the domain of a constant c.

Lemma Let D be an action description of a signature σ, and let c be a constant that does not belong to σ. For every \( v \in \text{Dom}(c) \), let \( F_v \) be a formula of the signature σ such that

- the formulas
  
  \[
  \bigvee_{v \in \text{Dom}(c)} F_v
  \]

  and
  
  \[
  \bigwedge_{v, w \in \text{Dom}(c), c \neq w} \neg (F_v \land F_w)
  \]

  are tautologies;

- if c is an action constant then \( F_v \) does not contain fluent constants;

- if c is a statically determined fluent constant then \( F_v \) does not contain action constants;

- if c is a simple fluent constant then \( F_v \) contains neither action constants nor statically determined fluent constants.

Let \( D' \) be the action description of the signature \( \sigma \cup \{c\} \) obtained from D by adding the rules

\[ \text{caused} \; c = v \equiv F_v \]

for all \( v \) from \( \text{Dom}(c) \). Then D is a residue of \( D' \).

Notice that the formulas \( F_v \) uniquely define the value of constant c in terms of atoms from \( \sigma \). This is why, at any vertex (or any edge) of the transition system represented by D, the values of fluents (or actions) will define a value for c, and this will be the value of c at the vertex (or edge) of the transition system represented by \( D' \).

The conservative extension property mentioned in Section 2 is a special case of the lemma above, where only atoms are allowed as formulas \( F_v \).

5 Conclusion

We have seen that a group of causal laws in the \( C_+ \) description of the Monkey and Bananas domain can be replaced by a reference to a general-purpose description of the action Move. This fact suggests that representing properties of actions can be facilitated by using a library of standard action descriptions.

The design of such a library and a methodology for its use are topics for future work, with many interesting issues to be addressed. A central one is how to use different special cases of the same library action together. For instance, of the actions in MB, three others besides PushBox can be expressed as special cases of Move. The actions Walk, ClimbOn and ClimbOff may be viewed as the monkey moving itself. Distinguishing between different “instances” of the same library action is essential.

This paper illustrated the use of a library action by starting with an existing action description and modifying it to make use of the library. Of course, as the library develops, action descriptions will be written to use the library directly, rather than as an afterthought. We expect this methodology to simplify the process of writing action descriptions.

The syntactic form of the causal laws used in this work makes the resulting action descriptions nondefinite. Therefore they currently cannot be used with the Causal Calculator, which is an implementation of the definite fragment of \( C_+ \). Investigation on ways to extend the implementation to handle nondefinite theories of the kind used here is an ongoing topic of research.

Acknowledgements

We are grateful to Paolo Ferraris, Joohyung Lee and Wan-wan Ren for useful discussions. This work is partially supported by the National Science Foundation under Grant IIS-0412907.

References


Negotiating Logic Programs

Norman Foo and Thomas Meyer
National ICT Australia and School of Computer Science and Engineering
University of New South Wales
NSW 2052, Australia

Yan Zhang and Dongmo Zhang
School of Computing and Information Technology
University of Western Sydney
Penrith South DC
NSW 1797, Australia.

Abstract

Negotiation between two agents is modelled as a one-time encounter between two extended logic programs. Each offers an answer set. Their mutual deal can be regarded as a trade on their answer sets. An agent can achieve this by weakening its program to surrender some literals that conflict with the offer of the other agent. We examine how ways of weakening affects the answer sets, and how they may be used to effect a deal. They are shown to satisfy the classification of outcomes that follow from known postulates for rational negotiation.

1 Introduction

Negotiation has been studied in many contexts (e.g. [Rosenstein and Zlotkin 94]). Ideally we would like to represent all kinds of intentions, attitudes, strategies, etc. of the agents who negotiate, but the complexity would be prohibitive for any kind of reasoning about them and their behavior. In game theory the usual models are greatly simplified in an attempt at rigorous analysis that can hopefully be extended to more realistic settings. Our work here can be regarded as a modest addition to recent research (e.g. [De Vos and Vermeir 02] [Foo et al. 04]) that introduced logic programs to reason about well-known game-theoretic scenarios. The novel features in this paper include: (1) there is no appeal to utilities; (2) agents are represented as general (extended) logic programs; (3) a negotiation is modelled as a trade between two agents with their respective program solutions as the commodity. A persuasive reason for adopting programs as models of agents is that the complexity of agent intentions, etc., can be gradually scaled up as the simpler models are understood. Two directions immediately suggest themselves: model the repeated encounter negotiations, and model agent epistemic states and intentions as rules. We first describe our agent model and briefly review the relevant logic programming background. Then methods for weakening a program, modelling partial concession by an agent in negotiation, are examined. A brief review of recent formal postulates on rational negotiation is next. Finally we evaluate our agent program concessions against these postulates.

2 Representation of Agents

While real-life negotiation between two agents is possibly iterative, involving repeated encounters in which the agents trade demands and concessions, for the moment we only model a single encounter in which both agents declare their demands simultaneously, and then each agent uses this mutual information to modify their original demands to achieve an agreement. The scenario in which an agreement is reached only after repeated encounters is akin to the notion of extensive games, while the alternative scenario of a one-time encounter is akin to that of a normal form game in the game theory context [Luce and Raiffa]. In that context it is known that there are translations between the two kinds of games formalisms, but normal form games are easier to analyse to understand the limits of what player strategies can achieve or guarantee. Likewise, we confine our attention to single encounter negotiation to gain intuition that will be useful in our future work on repeated encounters.

We model each agent as an extended logic program (ELP), a formalism introduced by Gelfond and Lifschitz [Gelfond and Lifschitz 90] to express both classical negation and negation-as-failure. Using ELPs permit us to make contradictory demands explicit and also admit incomplete knowledge by agents. The accepted semantics of ELPs are answer sets (op cit), which are in turn based on the stable model semantics of a less expressive formalism — general or normal logic programs (NLP) — also introduced by Gelfond and Lifschitz [Gelfond and Lifschitz 88].

The one encounter model of negotiation is effectively the presentation by each agent of an answer set. Based on the pair of presented answer sets and assumptions about the pre-disposition (-risky, cautious, obstinate, dominant, etc.) of the agents, they will arrive at the union of the modified answer sets as the agreement pair. Thus, the central issue in reaching this agreement is how each agent should modify its answer set. Since answer sets are determined by the agent program this modification is tantamount to changes in its program. Moreover, these changes should ideally be faithful to the intended modification of the answer set.

3 Review of Logic Programs

In this paper the logical language is propositional (ground atoms). A standard reference for the topics in this paragraph is the text by Lloyd [Lloyd 87]. The simplest logic
programs are the definite logic programs (DLPs) in which the rules are pure Horn clauses. A typical rule \( r \) is of the form \( A \leftarrow B_1, \ldots, B_n \) where \( A, B_1, \ldots, B_n \) are atoms. A definite program \( \Pi \) is a finite collection of such rules. The notation \( \text{head}(r) \) means the atom in the consequent of rule \( r \), and \( \text{body}(r) \) means the set of atoms in its antecedent. Thus for the typical rule \( r \) above we have \( \text{head}(r) = A \) and \( \text{body}(r) = \{B_1, \ldots, B_n\} \), and \( r \) can therefore also be denoted by \( \text{head}(r) \leftarrow \text{body}(r) \). The declarative semantics of \( \Pi \) is its smallest Herbrand model \( M_\Pi \), and its (equivalent) operational semantics is its least fixed point \( \text{lfp}(\Pi) \) or what amounts to the same thing the set of query (atoms) which succeeds; since this is unique we may call it the least model of \( \Pi \). By \( \text{Atoms}(\Pi) \) we mean the set of atoms that occur in \( \Pi \). A rule is redundant if it is never used to answer query \( B' \) for any atom \( B \in \text{lfp}(\Pi) \). A more formal way to define redundancy is via the standard \( T_\Pi \downarrow \) operator which records the rules used to produce \( \text{lfp}(\Pi) \), but the informal view suffices here. The important fact is that redundant rules can be deleted with no effect on the semantics of the program. One easy consequence is this:

**Lemma 1** For a DLP \( \Pi \) if \( A \notin \text{lfp}(\Pi) \) then any rule \( r \) with \( \text{head}(r) = A \) is redundant.

Default reasoning required the introduction of a kind of negation into logic programs, viz. inferring the negation of an atom if all attempts to demonstrate the truth of its positive form fails. This is the “negation-as-failure” notion of negation, denoted by \( \neg A \), and in the general logic programs (GLPs) or normal logic programs (NLPs) such terms are allowed in the bodies of rules. We will call them normal logic programs (NLPs) for short. With this generalization the bodies of rules split into those terms that are atoms and those atoms that are negated using \( \neg \); these sets are denoted respectively \( \text{pos}(r) \) and \( \text{neg}(r) \) for rule \( r \). Thus a normal rule \( r \) can be written as \( \text{head}(r) \leftarrow \text{pos}(r), \neg \text{neg}(r) \). An example NLP rule \( r \) is \( A \leftarrow B, C, \neg D, \neg E \), whence \( \text{head}(r) = \{A\} \), \( \text{pos}(r) = \{B, C\} \) and \( \text{neg}(r) = \{D, E\} \). As before, \( \text{body}(r) \) means \( \text{pos}(r), \neg \text{neg}(r) \) and likewise with \( \text{Atoms}(\Pi) \). The accepted semantics of NLPs is given by its Gelfond-Lifschitz transform [Gelfond and Lifschitz 88], which we now informally recall. Let \( S \) be a set of atoms and \( \Pi \) an NLP. Each rule \( r \) of \( \Pi \) is filtered by \( S \) as follows: if \( \text{neg}(r) \cap S \neq \emptyset \) then \( r \) is blocked. Intuitively this is because \( S \) is considered to be a “guess” at a model of \( \Pi \) and hence its atoms are assumed to be true. Then no rule \( r \) with \( A \in \text{neg}(r) \) and also \( A \in S \) can be applicable since the “default” \( \neg A \) is false relative to \( S \). The surviving unblocked rules may still have negated atoms in their bodies, but by assumption these are automatically satisfied by \( S \) and therefore can be dropped from their bodies. Hence we end up with a subset of the original rules of \( \Pi \) in which the negated terms have been deleted, viz., we are back to a DLP. Since this program derived from \( \Pi \) is \( S \)-relative, it is customary to denote it by \( \Pi^S \). Being a DLP \( \Pi^S \) has a least fixed point as its semantics. Gelfond and Lifshitz (op. cit.) define \( S \) to be a stable model of \( \Pi \) if \( \text{lfp}(\Pi^S) = S \).

If we further generalize the syntax of NLP to allow the atoms to be classically negated, then each rule can have literals in its head and body, and also negated literals in its body. Such programs were introduced also by Gelfond and Lifschitz [Gelfond and Lifschitz 90] who called them extended logic programs (ELPs). There are therefore two kinds of negation in ELPs — classical (\( \neg \)) and negation-as-failure (not). A typical ELP rule \( r \) is \( \neg A \leftarrow B, \neg C, \neg D, \neg E \). The models of ELPs can be regarded as stable models of an NLP in which the literals, say \( A \) and \( \neg A \) are represented (or encoded) respectively by atoms \( A \) and \( A' \). In fact this is essentially what Gelfond and Lifschitz did in their paper that introduced ELPs. The encoding of the preceding ELP rule \( r \) is the NLP rule \( r' = A' \leftarrow B, C', \neg D, \neg E' \), an NLP rule. If this is done, then the original ELP \( \Pi \) is transformed to (more precisely, encoded by) a NLP \( \Pi' \), and the stable model semantics for NLP is applicable. Each consistent stable model \( M \) (no pair \( A \) and \( A' \) in it), when translated back to the original stable model literals (\( A \) to itself, \( A' \) to \( \neg A \)), is an answer set of \( \Pi \).

### 4 DLP Reducts

Consider a definite logic program \( \Pi \). Suppose for some subset \( \Delta \subseteq M_\Pi \) of atoms we wish to modify \( \Pi \) to some definite program \( \Pi' \) with model \( M_{\Pi'} \) such that \( M_{\Pi} \setminus M_{\Pi'} = \Delta \), i.e., some atoms are lost in the new model.

The algorithm presented below removes one atom from the model of its input program; a generalization does this for sets of atoms. The single atom removal algorithm is a composition of two sub-algorithms, the first being the well-known unfolding of a program and the other a retraction of rules whose heads are atoms in \( \Delta \). Unfolding has been researched in detail since Tamaki and Sato [Tamaki and Sato 84] introduced it in the context of program optimization, and we refer to Aravindan and Dung [Aravindan and Dung 95] for the results needed for this sub-algorithm. To recall, given a definite program \( \Pi \), its unfolding with respect to an atom \( A \) is a new program \( \text{Unfold}(\Pi, A) \) obtained from \( \Pi \) by application of the following procedure:

1. For each pair of rules \( r, r' \) in \( \Pi \) with \( \text{head}(r) = A \) and \( A \in \text{body}(r') \), add the rule \( \text{head}(r') \leftarrow \text{body}(r), \text{body}(r') \setminus \{A\} \).
2. Delete rules \( r \) with \( A \in \text{body}(r) \).

The intuitive idea behind unfolding with respect to \( A \) is to eliminate \( A \) in the bodies of rules using partial execution. This works even if there are rules \( r \) in \( \Pi \) with \( \text{head}(r) = A \) and \( A \in \text{body}(r) \), i.e. there is a recursion in \( A \) (which are equivalent to the empty program [Lin 02] and are redundant) since they will be deleted in the second step. We can further simplify the new program by deleting other redundant rules in this manner: Remove all clauses \( r \) such that \( B \in \text{head}(r) \) and \( B \in \text{body}(r) \). We will assume that simplification is always done. Besides having no rule in \( \text{Unfold}(\Pi, A) \) with \( A \) in its body, unfolding has a nice property: (see Dung. (op. cit.).

**Observation 1** Unfolding preserves the model of \( \Pi \), i.e., the model of \( \text{Unfold}(\Pi, A) \) is the same.

The retraction part is this procedure.

Given a definite program \( \Pi \) for which every rule \( r \) satisfies \( A \notin \text{body}(r) \), the retract of atom \( A \) from
II is a new program \( \text{Retract}(II, A) \) obtained from II by deleting all rules \( r \) with \( \text{head}(r) = A \).

Observe that \( A \notin \text{Atoms} (\text{Retract}(II, A)) \). The retract of an atom \( A \) from a program is undefined if the condition \( A \notin \text{body}(r) \) is violated for some \( r \) in it.

Given a definite program \( II \) with model \( M \) the following transformation produces a new definite program \( \text{Reduce}(II, A) \) whose model is \( M \setminus \{ A \} \), i.e., eliminates the atom \( A \) from the model \( M \).

\[
\text{II}' = \text{Unfold}(II, A);
\text{Reduce}(II, A) = \text{Retract}(II', A)
\]

Retracting an atom is weaker than the operation of contraction in the well-known AGM theory of belief revision [Gardenfors 88]. In the AGM framework the contraction of an atom \( A \) from a theory \( T \) may also result in “side-effects” of contradictions of other atoms from \( T \). However, the retract here is more like its namesake in Prolog, and removes only rules with that atom in their heads.

**Lemma 2** The model of \( \text{Reduce}(II, A) \) is \( M_{II} \setminus \{ A \} \).

**Lemma 3** The models of \( \text{Reduce}(\text{Reduce}(II, A), B) \) and \( \text{Reduce}(\text{Reduce}(II, B), A) \) are the same.

The proof of this is by brute force, showing the equality (after simplification) of the two orders of reduction.

If we are only interested in the semantics of the transformed program, Lemma 3 justifies the use of \( \text{Reduce}(II, \Delta) \) to denote any of the equivalent programs that result from the algorithm when applied successively to \( II \) using the atoms from \( \Delta \) in any order. These lemmas imply the following proposition.

**Proposition 1** The model of \( \text{Reduce}(II, \Delta) \) is \( M_{II} \setminus \Delta \).

5 NLP and ELP Reducts

As mentioned in section 2 above agents will be represented as ELPs. Generally these do not have unique answer sets. So any reduction algorithm for them has to be a generalization of that for DLPs. The diagram below summarizes our approach. On the left hand side the objects are NLPs (ELPs) under the components of the reduction algorithm. On the right hand side the objects are DLPs. As indicated in the diagram the DLPs on the right are obtained from a corresponding NLP (ELP) via a Gelfond-Lifschitz transformation (GLT) relative to a stable model (answer set) \( S \). The diagram is generic for stable models (answer sets) \( S \), i.e. there is one for each such set. The idea is to refer questions about the stable models (answer sets) on the left to the unique models (least fixed points) on the right. By the remarks in section 3 it suffices to examine NLPs and stable models. The questions for ELPs can be decided via the ELP encoding of an ELP. The reduction algorithm for NLP also consists of two steps, of which the first is unfolding. This is virtually the same as that for DLPs where unfolding of \( A \) in the body of a rule is simply the syntactic replacement of \( A \) by the body of a rule with \( A \) in its head. That this body may contain positive and negated literals is not material. More precisely, if \( r \) is \( A \leftarrow \text{pos}(r), \text{neg}(r) \) and \( r' \) is \( B \leftarrow \text{pos}(r'), \text{neg}(r') \) then an added rule is \( B \leftarrow \text{pos}(r') \setminus \{ A \}, \text{pos}(r), \text{neg}(r), \text{neg}(r') \).

**Observation 2** Unfolding NLPs and DLPs is known [Aravindan and Dung 95] to preserve models, so the commutativity of the upper rectangle of figure 1 follows.

The commutativity of the lower rectangle is about the retract step. Certainly we still retract all rules \( r \) with \( \text{head}(r) = A \), as well as those with \( A \in \text{pos}(r) \). The new consideration is what we do with a rule \( r \) with \( A \in \text{neg}(r) \). To justify the two ways of treating this we make a digression into two interpretations of negation-as-failure.

5.1 Negation as Failure

There are two ways of treating the retraction of a negation-as-failure atom in a rule. We illustrate them using an example program (call it II) adapted from [Gelfond and Lifschitz 90].

\[
\{ \text{Eligible}(X) \leftarrow \text{HighGPA}(X); \text{Eligible}(X) \leftarrow \text{FairGPA}(X); \text{not Advantage}(X); \text{Advantaged}(X) \leftarrow \text{Wealthy}(X); \text{Wealthy}(a); \text{HighGPA}(b); \text{FairGPA}(c). \}
\]

The informal idea of these rules is to determine eligibility of college applicants for scholarships. \text{Advantaged}(X) signifies that \( X \) comes from an advantaged background, so the second rule captures the idea that disadvantaged applicants with only a fair GPA are still eligible.

Suppose we weaken II by suppressing the atom \text{Advantaged}(\_). Unfolding of II with respect to this atom has no effect on it. If we now delete the rule with \text{Advantage}(X) in the head the resulting program II’ is:

\[
\{ \text{Eligible}(X) \leftarrow \text{HighGPA}(X); \text{Eligible}(X) \leftarrow \text{FairGPA}(X); \text{not Advantage}(X); \text{Wealthy}(a); \text{HighGPA}(b); \text{FairGPA}(c). \}
\]

There are two ways to regard \text{not Advantage}(X) in the rule of II’. One way is to simply suppress it, giving the program II’:

\[
\{ \text{Eligible}(X) \leftarrow \text{HighGPA}(X); \text{Eligible}(X) \leftarrow \text{FairGPA}(X); \text{Wealthy}(a); \text{HighGPA}(b); \text{FairGPA}(c). \}
\]

II’ has one stable model \( S'' = \{ \text{Wealthy}(a), \text{HighGPA}(b), \text{FairGPA}(c), \text{Eligible}(b), \text{Eligible}(c) \} \), which has lost the atom \text{Advantaged}(a) from the original stable model.

Another way is to delete the rule in which \text{not Advantage}(X) occurs, giving II”:

\[
\{ \text{Eligible}(X) \leftarrow \text{HighGPA}(X); \text{Wealthy}(a); \text{HighGPA}(b); \text{FairGPA}(c). \}
\]

II” has one stable model \( S''' = \{ \text{Wealthy}(a), \text{HighGPA}(b), \text{FairGPA}(c), \text{Eligible}(b) \} \), which has lost both \text{Advantaged}(a) and \text{Eligible}(c) from the original model. Intuitively, in the absence of information about \text{Advantaged}(c), \( S''' \) still yields a desired outcome while
$S''$ does not\(^1\). The informal reason is that the occurrence of not $\text{Advantaged}(X)$ in the original rule is intuitively interpreted as a default test in the context of fair GPAs.

Contrast this with an example rule attributed to John McCarthy: $\text{Cross} \leftarrow \text{not Train}$, presumably occurring in a program that decides when a car should cross a level-crossing. The retracting of $\text{Train}$ by just deleting it in the rule would result in the fact $\text{Cross} \leftarrow$, which may be dangerous. In this case the intuitive treatment is to delete the rule altogether. The difference between this use of negation-as-failure and the preceding one is that here it is a support for the head of the rule.

Hence, both ways of treating retract should be considered when the informal semantics is not available.

### 5.2 Retracting negated literals

The intuitions of subsection 5.1 above suggest two ways of handling the not $A$ terms in the bodies of rules at the retract step, viz., (1) delete not $A$ in rules, and (2) delete the rules whose bodies contain not $A$. It is natural to call the former weak and the latter strong. Therefore, for NLPs we have two versions of the retraction algorithm:

**Weak retract:**

Given a definite program $\Pi$ for which every rule $r$ satisfies $A \not\in \text{pos}(r)$, the retract of atom $A$ from $\Pi$ is a new program $W\text{Retract}(\Pi, A)$ obtained from $\Pi$ by deleting all rules $r$ with head $(r) = A$, and by deleting not $A$ from all rules.

**Strong retract:**

Given a definite program $\Pi$ for which every rule $r$ satisfies $A \not\in \text{pos}(r)$, the retract of atom $A$ from $\Pi$ is a new program $S\text{Retract}(\Pi, A)$ obtained from $\Pi$ by deleting all rules $r$ with head $(r) = A$, and by deleting all rules whose bodies contain not $A$.

The two overall versions of reduction for NLPs are then:

Program $W\text{Reduce}(\Pi, A)$ is obtained from $\Pi$ by:

\[
\Pi' = \text{Unfold}(\Pi, A);
\]

\[
W\text{Reduce}(\Pi, A) = W\text{Retract}(\Pi', A).
\]

Program $S\text{Reduce}(\Pi, A)$ is obtained from $\Pi$ by:

\[
\Pi' = \text{Unfold}(\Pi, A);
\]

\[
S\text{Reduce}(\Pi, A) = S\text{Retract}(\Pi', A).
\]

Analogous to the case of DLPs, both these reductions remove $A$ from all stable models. To show this we refer to the commutative diagram in figure 1. The following observation is a direct consequence of the definition of stable models.

**Observation 3** At any horizontal line in figure 1, $S$ is a stable model of the program on the left if and only if it is the least fixed point (or least model) of the program on the right.

This observation reduces questions about a stable model $S$ of an NLP $\Pi$ to questions about the least fixed points of the DLP $\Pi^S = \text{GLT}(\Pi, S)$ in the figure, and thence they can be answered using results from section 4. The next observation follows from the fact that there are no not terms in DLPs.

\(^1\)However, there is a scenario in which $S'''$ is reasonable, viz., when it is intended to repeal affirmative action. This is analogous to the Train example.

**Observation 4** If $\Pi$ is a DLP then $W\text{Retract}(\Pi, A) = S\text{Retract}(\Pi, A) = \text{Retract}(\Pi)$.\(^2\)

Since NLP can have more than one stable model, the analog of lemma 2 is something along the lines of: Suppose $S$ is a stable model of $\Pi$; if $S \setminus \{A\}$ is non-empty it is a stable model of $W\text{Reduce}(\Pi, A)$, where $W\text{Reduce}$ is either $W$ (weak) or $S$ (strong). However, as suggested by the following example, it is slightly more complicated. Consider the program $\Gamma$: $\{A \leftarrow \text{not E}; C \leftarrow A, \text{not D}; \text{E} \leftarrow \text{not A}\}$ The stable models of $\Gamma$ are $\{E\}$ and $\{A, C\}$. Deleting $A$ from them yields $\{E\}$ and $\{C\}$. On the other hand, if we do weak reduction the only stable model is $\{E\}$, while strong reduction yields the stable model $\{C\}$.

**Proposition 2** Suppose $S$ is a stable model of $\Pi$ and $S \setminus \{A\}$ is non-empty.

1. If $A \not\in S$ then $S$ is also a stable model of $W\text{Reduce}(\Pi, A)$, but not always of $S\text{Reduce}(\Pi, A)$.

2. If $A \in S$ then $S \setminus \{A\}$ is a stable model of $S\text{Reduce}(\Pi, A)$, but not always of $W\text{Reduce}(\Pi, A)$.

Let $\text{Mod}(\Pi)$ be the set of stable models of $\Pi$. The generalization of lemma 3 is:

**Lemma 4**

\[
\text{Mod}(W\text{Reduce}(W\text{Reduce}(\Pi, A), B)) = \text{Mod}(W\text{Reduce}(W\text{Reduce}(\Pi, B), A))
\]

\[
\text{Mod}(S\text{Reduce}(S\text{Reduce}(\Pi, A), B)) = \text{Mod}(S\text{Reduce}(S\text{Reduce}(\Pi, B), A)).
\]

Therefore for a set $\Delta$ we have unambiguous meanings for $W\text{Reduce}(\Pi, \Delta)$ and $S\text{Reduce}(\Pi, \Delta)$. The following is then a consequence of proposition 2.

**Proposition 3** Suppose $S$ is a stable model of $\Pi$ and $S \setminus \Delta$ is non-empty.

1. If $\Delta \cap S = \emptyset$ then $S$ is also a stable model of $W\text{Reduce}(\Pi, \Delta)$, but not always of $S\text{Reduce}(\Pi, \Delta)$.

2. If $\Delta \subset S$ then $S \setminus \Delta$ is a stable model of $S\text{Reduce}(\Pi, \Delta)$, but not always of $W\text{Reduce}(\Pi, \Delta)$.

**Corollary 1** $A \not\in S$ for each $S \in \text{Mod}(W\text{Reduce}(\Pi, A))$. $A \not\in S$ for each $S \in \text{Mod}(S\text{Reduce}(\Pi, A))$.

**Definition 1** Fix a language. An NLP (ELP) program $\Pi'$ is a weakening of an NLP (ELP) program $\Pi$ if there is a one-to-one pairing between their models, such that for each $M' \in \text{Mod}(\Pi')$ there is an $M \in \text{Mod}(\Pi)$ such that $M' \subseteq M$.

In weakening a program, literals are lost from its stable models. In the negotiation context, an agent (program) may choose to surrender a particular subset $\Delta$ of its literals to avoid a conflict with the other party. The weakenings that result in losing no more than is necessary are maximal. Formally a weakening $\Pi'$ of $\Pi$ is $\Delta$-maximal if for each model pair $(M', M)$ it is the case that $M \setminus M' \subseteq \Delta$.

**Corollary 2** $S\text{Reduce}(\Pi, \Delta)$ is $\Delta$-maximal when $\Delta \subset S$ for each $S \in \text{Mod}(\Pi)$.
6 Negotiation between Programs

Imagine an encounter between two agents 1 and 2 represented respectively as ELPs (NLPs) \( \Pi_1 \) and \( \Pi_2 \). It is natural to regard any negotiated agreement or deal between them to have a connection with the answer sets (stable models) of the two programs. One way to do this is to consider the “merged” program \( \Pi_1 \cup \Pi_2 \) and its answer sets. We might then proceed to perform the reduction algorithm on this program, nominating subsets of literals to retract to achieve modified answer sets according to different rationality criteria. We consider this approach in a companion paper, but did not evaluate its behavior relative to rationality. Here we examine an alternative and more concrete approach by regarding the encounter as these successive events: Agents 1 and 2 each choose one of their answer sets, reveal the literals in the sets as their demands, and then each modifies its demands to arrive at an agreement. The modification is achieved by reductions on the agent program. Reductions effect the concession by agents of parts of their demands. This is evaluated for rationality using the postulates in the (next) subsection 6.1.

We are very much aware of the simplicity of our formalization of agent negotiation as a one time encounter. One does not have to reflect much to see that real-life negotiation involves multiple encounters. What passes for bargaining in fact repeated encounters in which the agents weaken some demands, accept some of their opponent’s demands, but come back with new demands, etc. until mutual satisfaction is reached. In the ELP context weakening can be captured as reduction, acceptance can be interpreted in two ways, and new demands may amount to switching answer sets. Acceptance of some demands of the other agent can be either passive or active. The passive mode just blocks the agent from ever demand literals contradicting the accepted set. The active mode is program expansion by adding the accepted literals as new facts into the agent’s program. This is like Prolog’s assert meta-procedure, and the dual to retract. A side effect of this active mode is the alteration of the program’s answer sets. Answer set switching can also be caused by another aspect of real-life negotiation that we have ignored, viz., preference orderings on rules and literals. These interesting features are being investigated.

6.1 Postulates for Rational Negotiation

We review recent work by Meyer, et al. [Meyer et al. 04] on rationality postulates that try to capture how rational agents should behave in a one encounter negotiation. These postulates are expressed abstractly in logic. We will evaluate our particular logic programming models of agents and algorithms for program change against these postulates so that we have some guarantees of rationality. The review below is essentially a summary of the relevant section of the cited paper (op. cit.). The language \( L \) of these postulates is finitely generated and propositional. Falsum is denoted by \( \bot \), logical entailment by \( \vdash \), logical equivalence by \( \equiv \) and logical closure by \( Cn \). A theory is a set of sentences closed under logical entailment. \( M(K) \) denotes the models of a set of sentences \( K \) and \( M(\alpha) \) that of a single sentence \( \alpha \).

A deal \( D \) is defined as an abstract object. Any deal is defined with respect to a demand pair \( K = (K_1, K_2) \), with \( K_i \) (\( i = 1, 2 \)), being a consistent theory of \( L \), representing the initial demands, or demand set, of agent \( i \). For simplicity agents are assumed to be logically omniscient. The theory developed models as being concerned with the outcome of the process of negotiation rather than the process by which this is reached. This is close to the one-time encounter model we choose for this paper and to normal form games. The outcome \( O(D) \) of a deal \( D \) is formalized as a set of sentences representing the demands which both agents have agreed upon. A deal \( D \) is outcome-permissible iff \( O(D) \) satisfies the following rationality postulates:

\[(O1) \ O(D) = Cn(\neg O(D))\]
\[(O2) \ O(D) \not\subseteq \bot\]
\[(O3) \text{If } K_1 \cup K_2 \not\subseteq \bot \text{ then } O(D) = Cn(K_1 \cup K_2)\]
\[(O4) \text{If } O(D) \cup (K_1 \cap K_2) \not\subseteq \bot \text{ then } (K_1 \cap K_2) \subseteq O(D)\]

\((O1)\) ensures that outcomes are theories and \((O2)\) that outcomes are consistent. \((O3)\) states that if the initial demand sets do not conflict, an agent will accept all demands of the other agent, i.e. agents cooperate when possible. \((O4)\) says that if the outcome and common demands are consistent, then all common demands have to be included. The intuitive justification is that if a potential outcome \( O \) is consistent with the demands that the agents have in common, it would be to the benefit of both rather to strengthen \( O \) to contain all commonly held demands.

6.2 A classification of deals

The constraints placed on outcomes by \((O1)-(O4)\) lead naturally to a taxonomy of deals in which we distinguish between four kinds of deals. A trivial deal is one for which the outcome is \( Cn(K_1 \cup K_2) \). This can only occur when the combination of the initial demand sets is consistent. In the next type of deal one of the agents, the master, gets to keep all its demands. An i-dominant deal \( (i = 1, 2) \) is one in which agent \( i \) plays the role of the master. Observe that the different supersets of the demands of agent \( i \) correspond to the different i-dominant deals. The third type of deal is the class of cooperative deals, where the outcome is consistent with the initial demand set of each agent and includes their common demands. Finally, we have the class of neutral deals in which the outcome of negotiation is inconsistent with the common demands. Meyer, et al. (op. cit.) defined these types of deals formally and proved that for any negotiation scheme that satisfies the above rationality postulates every deal is one of these types. For our purpose here however, the preceding informal description suffices to classify negotiation between two agents represented as NLPs (and hence as ELPs).

6.3 Algorithm for a Deal

The notation \( \neg L \) means (1) \( \neg L \) when \( L \) is an atom, and (2) \( A \) when \( L \) is the negation \( \neg A \) of atom \( A \). Extending the notation, by \( \neg \Delta \) where \( \Delta \) is a set of literals is meant the set \( \{\neg L \mid L \in \Delta \} \). Suppose agents 1 and 2 declare respective answer sets \( M_1 \) and \( M_2 \) as their demands. The contradiction set \( \text{Contrad}_1 \) of 1 is the set \( \{ L \in M_1 \mid \neg L \in M_2 \} \), and symmetrically for \( \text{Contrad}_2 \); thus \( \text{Contrad}_1 = [\neg \text{Contrad}_2 \text{Contrad}_1] \).

\[\text{Details and identities suppressed to maintain anonymity.}\]
and symmetrically with indices swapped. The initial agreement set $Agree$ of both 1 and 2 is $M_1 \cap M_2$. An agent 1 may wish to keep a set $Ψ_1 \subseteq M_1$, giving up $M_1 \setminus Ψ_1$. The subset $Ψ_1 \cap Contrad_1$ has the literals that conflict with some in $M_2$, so agent 2 will have to give up at least $\neg(Ψ_1 \cap Contrad_1)$ for consistency. If the new answer sets are to be consistent with each other, and both keep the agreement set, then the choice of $Ψ_1$ and $Ψ_2$ are constrained as follows: (1) $Agree \subseteq Ψ_1 \subseteq M_1 \cup (M_2 \setminus Contrad_2)$ and $Agree \subseteq Ψ_2 \subseteq M_2 \cup (M_1 \setminus Contrad_1)$ (2) $Ψ_1 \cap Contrad_1 \neq \emptyset$ implies $\neg(Ψ_1 \cap Contrad_1) \cap Ψ_2 = \emptyset$ and $\neg(Ψ_2 \cap Contrad_2) \cap Ψ_1 = \emptyset$. Call any pair $(Ψ_1, Ψ_2)$ that satisfies the constraints admissible.

**Deal Algorithm:**
For any admissible pair $(Ψ_1, Ψ_2)$:

1. $Π_1' = SReduce(Π_1, M_1 \setminus Ψ_1)$
2. $Π_2' = SReduce(Π_1, M_1 \setminus Ψ_2)$
3. $M_1' = Ψ_1; M_2' = Ψ_2$
4. Outcome = $M_1' \cup M_2'$

By corollary 2 the algorithm results in new programs $Π_1'$ and $Π_2'$ with new answer sets $M_1'$ and $M_2'$. Some further easily verified observations on the algorithm:

- If $Contrad_1 = \emptyset$ (and therefore also $Contrad_2 = \emptyset$) then $Ψ_1 = M_1$ and $Ψ_2 = M_2$ is an admissible pair. In that case the $SReduce$ function in the algorithm returns the original programs. The outcome is then simply the union of the two original answer sets of the agents, viz., the trivial deal.
- If $M_1 \subseteq Ψ_1$ then by (2) any $Ψ_2$ such that $Ψ_2 \cap Contrad_2 = \emptyset$ is an admissible pair. For such a pair the outcome is a 1-dominant deal. The symmetric case is a 2-dominant deal.
- If $Ψ_1 = M_1 \setminus Contrad_1$ and $Ψ_2 = M_2 \setminus Contrad_2$, the outcome is a cooperative deal.

These conform to the classification of outcome types in subsection 6.2 above. The apparently missing one is the neutral outcome. We conjecture that such outcomes will manifest themselves once preferences and fact assertions are introduced. However, an arguable view is the following. The outcome above is only ostensibly cooperative if we consider $M_1$ and $M_2$ as sets. If we admit logical closure $Con$, the picture changes. Here is an example. Suppose $M_1 = \{p, q, r\}$ and $M_2 = \{\neg p, \neg q, r\}$. Then set-theoretically we have $M_1 \cap M_2 = \{r\}$; but if we had considered instead $Con(M_1)$ and $Con(M_2)$ the formula also common to both is $p \iff q$. Hence if consequence closure $Con$ is admitted as part of the description of stable models, an outcome in which the agents 1 and 2 give up $\langle p, q \rangle$ and $\langle \neg p, \neg q \rangle$ respectively is not a cooperative deal since it fails to preserve the common (implicit) formula $p \iff q$; in fact this is a neutral outcome. Evidently, we have to be careful in such evaluations against rationality postulates whose theorems assume logical closures.

---

A case can be made that it is the programs that should be regarded as the outcomes, and their stable models are extensional side effects.

---

**References**


Compiling Qualitative Preferences into Decision-Theoretic Golog Programs

Christian Fritz and Sheila McIlraith
Department of Computer Science
University of Toronto
Toronto, Ontario. Canada.
{fritz,sheila}@cs.toronto.edu

Abstract
Personalization is becoming increasingly important in agent programming, particularly as it relates to the Web. We propose to develop underspecified, task-specific agent programs, and to automatically personalize them to the preferences of individual users. To this end, we propose a framework for agent programming that integrates rich, non-Markovian, qualitative user preferences with quantitative Markovian reward functions. We begin with DT-Golog, a first-order, decision-theoretic agent programming language in the situation calculus. We present an algorithm that compiles qualitative preferences into Golog programs and prove it sound and complete with respect to the space of solutions. To integrate these preferences into DT-Golog we introduce the notion of multi-program synchronization and restate the semantics of the language as a transition semantics. We demonstrate the utility of this framework with an application to personalized travel planning over the Web. To the best of our knowledge this is the first work to combine qualitative and quantitative preferences for temporal reasoning. Further, while the focus of this paper is on the integration of qualitative and quantitative preferences, a side effect of this work is realization of the simpler task of integrating qualitative preferences alone into agent programming.

1 Introduction
Personalization is becoming increasingly important to agent programming. Service-sector agent programs such as personal assistants or travel planners are often characterized by a relatively well-defined but under-specified set of tasks that can be realized in a variety of different ways. As with an office admin assistant or a travel agent, these high-level tasks are commissioned by numerous different customers/users. A good agent program, like a good office assistant or travel planner must be able to personalize the service they provide to meet the preferences and constraints of the individual.

Consider the oft-used example of travel planning: Fiona would like to book a trip from Toronto, Canada to Edinburgh, Scotland for work. She’d like to depart between July 25 and 28, returning no sooner than August 5, but no later than August 8. She would prefer not to connect through London Heathrow, as she had a bad experience being stuck at Heathrow when air traffic controllers went on strike last year. She’ll need a hotel in Edinburgh, preferably close to the castle but if the plane arrives late at night, she’d prefer a hotel close to the airport. Fiona needs to economize, so she’d like the cheapest flights and hotel accommodations possible. Nevertheless, she’s willing to pay $100 more to get a direct flight. Finally, she has to work July 29 – August 5, so she’s willing to spend up to $200 more to maximize sightseeing days before July 29 and/or after August 5.

This, presumably realistic setting, displays three types of constraints or preferences that are commonplace in many planning and agent programming application domains: hard constraints (when to go and where), qualitative preferences (airport and hotel preferences), and quantitative preferences (financial restrictions).

We approach the problem of personalizable agent programs by developing task-specific, but underspecified agent programs that have sufficient non-determinism to support personalization. Personalization is achieved by integrating these agent programs with the three types of constraints illustrated in our example above. The goal of this paper is to investigate the integration of qualitative and quantitative preferences into agent programming, and specifically into the agent programming language Golog [14].

Golog is a first-order agent programming language based on the situation calculus. Golog enables the specification of, potentially nondeterministic, agent programs in the context of a domain-specific action theory. As such, Golog programs impose hard constraints on the possible evolution of the domain. Decision-Theoretic Golog (DT-Golog) [5] extends Golog with the ability to solve MDP-like planning problems up to a given horizon and starting in a known initial situation. In so doing, DT-Golog can handle infinite state (situation) spaces while exploiting the underlying power of Golog to restrict the search space.

There is a large body of research on the use of quantitative preferences in automated reasoning. Indeed, decision-theoretic planning via Markov Decision processes (MDP) [13] provides an effective means of generating task plans that maximize a user’s expected utility. Unfortunately, preferences and constraints must be specified in terms of numeric,
Markovian reward functions. Such specifications can be difficult to elicit and don’t capture qualitative user preferences. Bacchus et al. [1] addressed the Markovian restriction, by enabling the use of non-Markovian rewards. They did so by augmenting state representation with a new set of temporal variables. Nevertheless, they did not allow for qualitative preferences.

Unfortunately, there has been little work on the incorporation of qualitative preferences into planning, save recent work by [3; 16; 7]. These approaches are able to represent qualitative non-Markovian user preferences, while [3; 16] also propose a means of planning with such preferences.

In [9] Domshlak et al. integrate quantitative soft constraints and qualitative preferences expressed using the CP-nets formalism [4]. They approach the problem by approximating the CP-net with soft constraints expressed in a semi-formalism. Nevertheless, their focus is on reasoning about preferences, i.e. deciding on an ordering of possible world states, and it is not obvious how their approach applies to planning or agent programming. In particular, the language they use for specifying preferences does not enable the expression of temporally extended preferences, which we believe are essential to the task at hand.

In this paper we address the problem of combining non-Markovian quantitative preferences, expressed in first-order temporal logic, with quantitative decision-theoretic reward functions and hard symbolic constraints. We do so by compiling non-Markovian quantitative preferences into a DT-Golog program, integrating the potentially competing preferences through a multi-program synchronization. The resultant DT-Golog program, maximizes the users expected utility within the expression of temporally extended preferences, which we believe are essential to the task at hand.

Our work is related to that of Gabaldon [12] who, following previous work by Bacchus and Kabanza [2] and Doherty and Kvarnström [8], compiles temporal logic formulae into preconditions of actions in the situation calculus. There, the temporal formulae are hard constraints that serve to reduce the search space. In contrast, we are unable to eliminate any of the search space, since qualitatively less preferred situations may yield the best final solution. Also related is the work of Sardina and Shapiro [15] who integrate qualitative prioritized goals into the IndiGolog programming language. Our approach differs from theirs in several ways: our qualitative preference language is richer than their specification of prioritized goals; we compile preferences into a Golog program which is more efficient from a computational perspective; and we enable the integration of both qualitative and quantitative constraints.

In Section 2 we review the situation calculus and Golog. In Section 3 we propose a first-order language for specifying non-Markovian qualitative user preferences. The semantics of the language is described in the situation calculus. Section 4 describes our approach to integrating preferences. It comprises three steps: compilation of non-Markovian qualitative preferences into a Golog program; multi-program synchronization of the resulting Golog program with an existing Golog program; and given this newly synchronized program, a means of defining preferences over different possible sub-programs. Included are a soundness and completeness result relating to our compilation, and a new transition semantics for DT-Golog. We have implemented our approach as an extension to Readylog [10], an existing on-line decision-theoretic Golog interpreter. We demonstrate its utility with an application to personalized travel planning over the Web, as discussed in Section 5. We summarize our contributions in Section 6.

2 Situation Calculus and Golog

The situation calculus is a logical language for specifying and reasoning about dynamical systems [14]. In the situation calculus, the state of the world is expressed in terms of functions and relations (fluents) relativized to a particular situation $s$, e.g., $F(\langle \vec{x}, s \rangle)$. In this paper, we distinguish between the set of fluent predicates, $\mathcal{F}$, and the set of non-fluent predicates, $\mathcal{R}$, representing properties that do not change over time. A situation $s$ is a history of the primitive actions, $a \in \mathcal{A}$, performed from a distinguished initial situation $s_0$. The function $do(a, s)$ maps a situation and an action into a new situation. The theory induces a tree of situations, rooted at $s_0$.

A basic action theory in the situation calculus, $\mathcal{D}$, comprises four domain-independent foundational axioms, and a set of domain-dependent axioms. The foundational axioms $\Sigma$ define the situations, their branching structure and the situation predecessor relation $\sqsubseteq: s \sqsubseteq s'$ states that situation $s$ precedes situation $s'$ in the situation tree. $\Sigma$ includes a second-order induction axiom. The domain-dependent axioms are strictly first-order. Details of the form of these axioms can be found in [14]. Following convention we will generally refer to fluents in situation-suppressed form, e.g., $at(\text{toronto})$ rather than $at(\text{toronto}, s)$.

Golog (e.g., [14]) is a high-level logic programming language for the specification and execution of complex actions in dynamical domains. It builds on top of the situation calculus by providing Algol-inspired extralogical constructs for assembling primitive situation calculus actions into complex actions $\delta$. Constructs include the following:

- $\alpha$ — primitive actions
- $\delta_1, \delta_2$ — sequences
- $\delta^? (\alpha)$ — tests
- $\delta^? (\pi x).\phi (x)$ — nondeterministic choice of arguments
- $\delta^* (\alpha)$ — nondeterministic iteration
- $\text{ndet}(L)$ — nondeterministic choice of (complex) action in list, $L$
- if $\phi$ then $\delta_1$ else $\delta_2$ endIf — conditionals
- proc $P(\overline{\mathcal{P}})$ $\delta$ endProc — procedure

These constructs can be used to write programs in the language of a domain theory, e.g.,

\begin{verbatim}
buyAirTicket(\overline{x});
if far then rentCar(\overline{y}) else bookTaxi(\overline{y}) endIf.
\end{verbatim}

There are two popular semantics for Golog programs: the original evaluation semantics [14] and a related single-step transition semantics that was proposed for on-line execution [6]. Following the evaluation semantics, complex actions are
macros that expand to situation calculus formulae. The abbreviation $do(\delta, S_0, do(\vec{a}, S_0))$ denotes that the Golog program $\delta$, starting execution in $S_0$ will legally terminate in situation $do(a_1, do(a_2, \ldots, do(a_n, S_0)))$ \(^1\). The following are some example macro expansions.

\begin{align*}
&\text{Do}(a, s, s') \overset{\text{def}}{=} \text{Poss}(a[s], s') \\
&\text{Do}(\vec{p}(\vec{s}), s, s') \overset{\text{def}}{=} \forall \vec{s} \in s \land s = s' \\
&\text{Do}(\neg\text{ndet}([\vec{s}], [\vec{r}], s, s')) \overset{\text{def}}{=} \text{Do}(\vec{s}(1), s, s') \lor \text{Do}(\neg\text{ndet}(\vec{s}), s, s') \overset{\text{def}}{=} \text{Do}(\vec{s}(1), s, s') \\
&\text{Do}(\neg\text{ndet}([\vec{s}], [\vec{r}], s, s')) \overset{\text{def}}{=} \text{Do}(\vec{s}(1), s, s')
\end{align*}

Given a domain theory, $D$ and Golog program $\delta$, program execution must find a sequence of actions $\vec{a}$ such that: $D \models \text{Do}(\delta, S_0, do(\vec{a}, S_0))$. Recall that $D$ induces a tree of situations rooted at $S_0$. Requiring that $D$ entails $\text{Do}(\delta, S_0, do(\vec{a}, S_0))$ serves to constrain the situations in the tree to only those situations consistent with the expansion of $\delta$.

These hard constraints can reduce the problem size by orders of magnitude. Consider the following estimate of our travel planning example. The full grounded search space involves $365^2$ date combinations and 1901 airports. Considering 10 available flights for every combination, there are more than $4.8 \cdot 10^{12}$ flights. Optimistically assuming that at each destination there are only 10 hotels with 5 rooms types each, the total number of possible action combinations increases to $6.2 \cdot 10^{21}$. Using a DT-Golog procedure such as the one that follows reduces the number of alternatives to approximately $3 \cdot 3 \cdot 10 \cdot 50 = 4500$ cases that are relevant to Fiona. Such reductions are of particular importance for agent programming on the Web, where the vastness of information creates enormous search spaces.

In this paper we exploit a decision-theoretic variant of Golog called DT-Golog [5], which extends Golog to deal with uncertainty in action outcomes and general reward functions. DT-Golog can be viewed alternatively as an extension to Golog, or as a means to give “advice” to a decision-theoretic planner that maximizes expected utility.

For example, our travel planning problem could be described by the following DT-Golog procedure:

```plaintext
proc( travel_planner,
    pickBest( depart_dt, [726..728],
        pickBest( return_dt, [805..807],
            searchFlight("YYZ", "EDI", depart_dt, return_dt),
            searchHotel("EDI", depart_dt, return_dt),
            pickBest( bestFlight, allFlights,
                [ reserveFlight(bestFlight),
                  if(\text{not}(error),\text{payFlight(bestFlight)))],
                ?(\text{not}(\text{outflight} = \text{none}),\text{not}(\text{inflight} = \text{none}))),
            pickBest( bestHotel, allHotels,
                [ reserveHotel(bestHotel),
                  if(\text{not}(error),\text{payHotel(bestHotel))],
                ?(\text{not}(\text{hotel} = \text{none}))))).)
```

Note the extensive use of the DT-Golog construct pickBest! Value, Range, Program which picks the best value for Program from the range of possibilities. E.g., our program picks the best departure and return dates from the specified ranges (726 denotes July 26, etc.), and so on. In this framework the utility theory is specified by action costs (e.g., the cost of purchasing an airline ticket) and Markovian reward functions assigning real-valued rewards to situations. E.g.,

\begin{align*}
&\text{reward}(v, s) \equiv \\
&\text{(at}(\text{EDI}, s) \land \text{date}(s) < 729 \lor \text{date}(s) > 805) \land v = 200) \lor \\
&\text{(~(at}(\text{EDI}, s) \land \text{date}(s) < 729 \lor \text{date}(s) > 805) \land v = 0)
\end{align*}

This says that the reward, $v$, is 200 if we are in Edinburgh before July 29 or after August 5, and $v$ is 0 otherwise.

But Fiona cannot easily specify all her preferences as numeric Markovian rewards. A rich qualitative preference language that exploits temporal logic should help!

3 Preference Language

To personalize agent programs, we use a subset of a rich first-order language for expressing non-Markovian user preferences recently proposed in [3]. The semantics of this language is defined in the situation calculus.

3.1 Syntax

In this section we present the syntax of a first-order language for expressing qualitative, non-Markovian user preferences. Our language is a subset of the preference language we proposed in [3], which is a modification and extension of Son and Pontelli’s PP language [16]. Constraints on the properties of situations are expressed by Basic Desire Formulae (BDF). BDFs are combined into Qualitative Preference Formulae \(^4\), using a preference ordering $\preceq$.

**Definition 1 (Basic Desire Formula (BDF)).** A basic desire formula is a sentence drawn from the smallest set $B$ where:

1. $F \cup R \subseteq B$, where $F$ is the set of fluents and $R$ is the set of non-fluent relations;
2. If $a \in A$, the set of primitive actions, then $\text{occ}(a) \in B$, stating that action $a$ occurs;
3. If $f \in F$, then $\text{final}(f) \in B$;
4. If $\psi_1, \psi_2$ are in $B$, then so are $\neg\psi_1, \psi_1 \land \psi_2, \psi_1 \lor \psi_2, \text{conditional}\ \psi_1 : \psi_2$ (equivalent to $(\psi_1 \land \psi_2) \lor \neg\psi_1$), $(\exists x)\psi$, $(\forall x)\psi$, $\text{next}(\psi)$, $\text{always}(\psi)$, $\text{eventually}(\psi)$, and $\text{until}(\psi, \psi_2)$.

BDFs establish desired properties of situations. The first three BDF forms are evaluated with respect to the initial situation unless embedded in a temporal connective. By combining BDFs using boolean and temporal connectives, we are able to express a variety of properties of situations. In our travel example:

\begin{align*}
&\text{always}((\exists y, z)\text{bookFlight}(y) \land \text{arriveLate}(y) \land \\
&\text{closeToAirport}(z) \land \neg\text{occ(bookHotel(z))}) \quad (1) \\
&\text{always}(\neg(\text{at}(LR))) \quad (2)
\end{align*}

Again, BDFs enable a user to define preferred situations. To express preferences among alternatives, we define the notion of qualitative preference formulae.

**Definition 2 (Qualitative Preference Formula).** $\Phi$ is a qualitative preference formula if one of the following holds:

- $\Phi$ is a basic desire formula
- $\Phi = \Psi_1 \preceq \Psi_2$, with $\Psi_{1,2}$ qualitative preference formulae.

\(^1\)which we abbreviate to $do(\vec{a}, S_0)$ or $do([a_1, \ldots, a_n], S_0)$.

\(^2\)denotes the re-insertion of $s$ into fluent arguments of $a$.

\(^3\)denotes a list with first element $a$, and rest of list $r$.

\(^4\)Subsequently referred to as preference formulae.
\( \mathcal{S} \) is an Ordered And preference. We wish to satisfy both \( \Psi_1 \) and \( \Psi_2 \), but if that is not possible, we prefer to satisfy \( \Psi_1 \) over \( \Psi_2 \). Note that this is enough to also express conditional preferences of the form “if \( a \) then I prefer \( b \) over \( c \)”, as this can be transformed to \( (a : b) \wedge \mathcal{S} c \) which has the same semantics: if \( a \) holds, then I want to satisfy both \( b \) and \( c \) with a preference for \( b \). If \( a \) does not hold, \( a : b \) is immediately satisfied and it only remains to satisfy \( c \). Qualitative preference formuale may be arbitrarily long.

### 3.2 Semantics

Following our recent work [3], preference formulae are interpreted as situation calculus formulae and are evaluated relative to an action theory \( \mathcal{D} \). Since BDFs may refer to properties that hold over fragments of a situation history, we use the notation \( \varphi[s, s'] \), proposed in [12], to explicitly denote that \( \varphi \) holds in the sequence of situations originating in \( s \) and terminating in \( s' = do(\mathcal{A}, s) \). BDFs are interpreted in the situation calculus as follows:

- \( \varphi \in \mathcal{F} \), \( \varphi[s, s'] \overset{\text{def}}{=} \varphi[s] \)
- \( \varphi \in \mathcal{R} \), \( \varphi[s, s'] \overset{\text{def}}{=} \varphi'[s'] \)
- \( \text{final}(\varphi)[s, s'] \overset{\text{def}}{=} \varphi'[s'] \)
- \( \text{occ}(\mathcal{A})[s, s'] \overset{\text{def}}{=} \text{do}(\mathcal{A}, s) \subseteq s' \wedge \text{Pos}(\mathcal{A}[s], s) \)
- \( \text{eventually}(\varphi)[s, s'] \overset{\text{def}}{=} (\exists s_1 : s \subseteq s_1 \subseteq s') \varphi[s_1, s'] \)
- \( \text{always}(\varphi)[s, s'] \overset{\text{def}}{=} (\forall s_1 : s \subseteq s_1 \subseteq s') \varphi[s_1, s'] \)
- \( \text{next}(\varphi)[s, s'] \overset{\text{def}}{=} (\exists a) \cdot \text{do}(a, s) \subseteq s' \wedge \varphi[\text{do}(a, s), s'] \)
- \( \text{until}(\varphi, \psi)[s, s'] \overset{\text{def}}{=} (\exists s_2 : s \subseteq s_2 \subseteq s') \{ \psi[s_2, s'] \wedge (\forall s_3 : s \subseteq s_3 \subseteq s_2) \varphi[s_3, s'] \} \)

The boolean connectives are already defined in the situation calculus. Since each BDF is shorthand for a situation calculus expression, a simple model-theoretic semantics follows.

**Definition 3.** Let \( \mathcal{D} \) be an action theory, and let \( s \) and \( s' \) be two situations such that \( s \subseteq s' \). A basic desire formula \( \varphi \) is satisfied by the situation beginning in \( s \) and terminating in \( s' \) just in case that \( \mathcal{D} \models \varphi[s, s'] \).

Intuitively a qualitative preference formula \( \Phi = \Psi_1 \mathcal{S} \Psi_2 \) partitions the space of situations into four equivalence classes of preferred situations, in decreasing order of preference: (1) those satisfying both \( \Psi_1 \) and \( \Psi_2 \), (2) those only satisfying \( \Psi_1 \), (3) those only satisfying \( \Psi_2 \), and (4) those satisfying neither. The semantics of qualitative preference formulae are defined in a subsequent section using Golog constructs. Their semantics follows from the semantics of Golog.

### 4 Adding Preferences to DT-Golog

BDFs are the building blocks of our qualitative preference formulae. Like Golog programs, BDFs impose constraints on situations. As such, it is natural to integrate BDFs into Golog by translating them into (generally non-deterministic) Golog programs. Preference over the enforcement of BDFs is expressed by qualitative preference formulae. These preferences can be realized in Golog by the multi-program synchronization of BDF-induced Golog programs with the original agent program, and by prioritized execution of the resultant nondeterministic programs in a manner consistent with the defined preferences.

Synchronization of BDF-induced Golog programs with DT-Golog programs [5] results in a natural integration of agent programming under both qualitative preferences and quantitative utility theory. Since qualitative and quantitative expressions of preference are not immediately comparable, one has to decide how to rank them in case they are contradictory, i.e. favour different plans. In this paper we rank qualitative preferences over quantitative ones. As a result, we first try to find the quantitatively best plan within the set of most preferred plans, and only if no such plan exists, broaden our scope to less qualitatively preferred plans. Nevertheless, a different ordering or even several ‘layers’ would be easy to realize in the presented framework.

The outline of our approach is as follows: (1) compile BDFs into Golog programs such that any successful execution of that program will result in a situation that satisfies the BDF, (2) define multi-program synchronization to couple the execution of two programs so as to combine a given agent program with the compilation result, (3) based on this, define preferences over different subprograms.

#### 4.1 Compilation

This section describes how we compile BDFs into Golog programs. The compilation works by progression up to a given horizon. At each progression step, the mechanism produces a set whose elements consist of a possible program step that can be performed without violating the BDF, and a possibly modified BDF that remains to be satisfied. Recursively these remaining BDFs are processed. As a progression step may return more than one branch (program-step/remaining-formula combination), compilation produces a tree, where branches are linked using nondeterministic choice. This tree describes the set of all possible program traces, i.e. situations of the situation calculus, that satisfy the BDF.

**Example 1.** Consider the following BDF: \( \text{always(happy)} \wedge \text{final(rich)} \) and assume \( A \) is a list of all primitive actions in our domain theory. Then the following program describes all possible sequences of length \( \leq 2 \) that satisfy this BDF:

\[
\text{nondet}([\text{happy} \wedge \text{rich}],
\text{[happy?]; nondet(A); nondet([[\text{happy} \wedge \text{rich}]],
\text{[happy?]; nondet(A); nondet([[\text{happy} \wedge \text{rich}]])])])}
\]

That is, either I am happy and rich already, or I am happy, take some action and then am happy and rich, or again I am happy and take another step. In the end I always have to be happy and rich. Any successful execution of this Golog program will satisfy the BDF.

Again, BDFs define desired properties of situations. As such, the maintenance of BDFs restricts the set of actions that may be taken in a situation. This insight is key to our compilation approach. We call the constraints required to enforce
our BDFs situation constraints. We express a situation constraint in Golog by a test $\varphi$ that enforces a fluent/nonfluent and/or a nondeterministic choice of the actions available in the current situation. In many cases, this is all actions, $A$.

Recall that in Golog $\varphi$ states that the formula $\varphi$ has to hold in the current situation and that $\text{nondet}(L)$ is the nondeterministic choice among the elements of the list $L$. For example, the only possible next steps for $\text{nondet}([a, b])$ are taking action $a$ or taking action $b$. Thus, assuming the current situation is $s$, the set of possible successor situations are restricted to $\{do(a, s), do(b, s)\}$. The scope of situation constraints can be expanded over several situations by using temporal expressions. In the example, the constraint of being happy is extended over all situations using always. Observe that several BDFs are contributing situation constraints to the same situation. To combine several situation constraints we define the function $\chi$. Note that the BDFs $\psi$ are treated as syntactic entities in the context of our compilation and are syntactically manipulated accordingly.

- $\chi(\psi_1?, \psi_2?) = (\psi_1 \land \psi_2)$
- $\chi(\psi, \text{nondet}(L)) = (\psi, \text{nondet}(L))$
- $\chi((\psi_1?) \land \text{nondet}(L_1), (\psi_2?) \land \text{nondet}(L_2)) = ((\psi_1 \land \psi_2)? \land \text{nondet}(L_1 \land L_2))$

plus its reflexive completion, where the $\psi'$s are formulae of the situation calculus and the $L$'s are lists of actions. In our example, the temporal extent of always and final overlap. In these situations, the situation constraints imposed by the two BDFs are combined using $\chi$.

Let $A$ be the set of actions in our domain, $F$ the set of fluents, $R$ the set of non-fluent predicates, then, formally the compilation of a basic desire formula $\psi$ is defined using the predicate $C$: $\bar{C}(\psi, SC, \psi')$ holds iff $SC$ is a situation constraint whose execution will not violate $\psi'$, and further $\psi'$ is a BDF that needs to be satisfied in the future.

In the following we use STOP as a shorthand for $\bar{\text{aocc}}(a)$. $\bar{C}$ is defined by the following set of axioms.

- $\bar{C}(f, f?, TRUE, \forall f \in F \cup R)$
- $\bar{C}(\text{occ}(a), \text{nondet}([a]), TRUE), \forall a \in A$
- $\bar{C}(\text{final}(f), SC, \psi') \equiv (SC = \text{final}([] \cup \psi') \land \text{nondet}([a])) \land \text{nondet}(A) \land \psi_1 \land \psi_2$
- $\bar{C}(\psi \land \psi_2, SC, \psi') \equiv (\psi \land \psi_2, SC, \psi') \land SC = \chi(SC_1, SC_2) \land \psi' = \psi_1 \land \psi_2$
- $\bar{C}(\chi(\psi_1, SC_1, \psi_1') \land \bar{C}(\psi_2, SC_2, \psi_2)) \land SC = \chi(SC_1, SC_2) \land \psi' = \psi_1 \land \psi_2$
- $\bar{C}(\psi \lor \psi_2, SC, \psi') \equiv \bar{C}(\psi \lor \psi_2, SC, \psi') \lor \bar{C}(\psi_2, SC, \psi')$
- $\bar{C}((\exists x)\psi, SC, \psi') \equiv \bar{C}(\chi(\psi_1 \land \psi_2), \text{nondet}(\forall \psi_1), SC, \psi')$
- $\bar{C}((\forall x)\psi, SC, \psi') \equiv \bar{C}(\chi(\psi_1 \land \psi_2), \text{nondet}(\forall \psi_1), SC, \psi')$
- $\bar{C}(\text{next}(\psi), \text{nondet}(A), \psi')$

$\text{nondet}([])$ states that no action may be taken. Together with the remaining BDF STOP, it enforces immediate program termination.

$\bar{C}(\psi_1?) \equiv (\bar{C}(\psi, SC, \psi') \land \psi' = \text{STOP} \land (\psi'' = \text{STOP} \lor \psi'' = \text{TRUE})) \lor (\bar{C}(\psi \land \text{next}(\psi), SC, \psi'))$

$\bar{C}(\text{eventually}(\psi), SC, \psi') \equiv \bar{C}(\psi \lor \text{next}(\text{eventually}(\psi)), SC, \psi')$

$\bar{C}(\text{until}(\psi_1, \psi_2), SC, \psi') \equiv \bar{C}(\psi_2 \lor (\psi_1 \land \text{next}(\text{until}(\psi_1, \psi_2))), SC, \psi')$

$\bar{C}(\text{TRUE}, SC, \text{TRUE}) \equiv SC = \text{nondet}([]) \lor SC = \text{nondet}(A)$

Negation requires special treatment. Golog finds situations, i.e. action sequences, that satisfy a program, but to address negation it is not obvious how the complement, that is the situations that do not satisfy the program, would be computed. We address this by pushing the negation down to the atomic level. For parsimony we only show some less obvious cases:

- $\bar{C}(-f, -f?, TRUE, \forall f \in F \cup R)$
- $\bar{C}(-\text{occ}(a), SC, \psi') \equiv (SC = \text{nondet}([]) \land \psi' = \text{STOP}) \lor (SC = \text{nondet}(A \setminus \{a\}) \land \psi' = \text{TRUE}, \forall a \in A)$
- $\bar{C}(-\text{always}(\psi), SC, \psi') \equiv \bar{C}(\text{eventually}(\neg \psi), SC, \psi')$
- $\bar{C}(-\text{until}(\neg \psi_1, \psi_2), SC, \psi') \equiv \bar{C}(\neg \psi_2 \lor (\neg \psi_1 \lor \text{next}(\neg \text{until}(\neg \psi_1, \psi_2))), SC, \psi')$

Based on $\bar{C}$ we can define the following (second-order) formula that relates a BDF $\psi$ to a Golog program $P$ such that every successful execution of $P$ results in a situation that satisfies $\psi$ where $h$ is the maximal number of actions in any such execution.

$\exists(\psi, P, h) \equiv (\psi = \text{TRUE} \land P = (\text{nondet}(A))^*) \lor (\psi = \text{STOP} \land P = \text{nondet}(A)) \lor (h = 0 \land \exists x.\bar{C}(\psi, P, x) \land \exists y.\bar{C}(\psi, y) \land \exists z.\bar{C}(\psi, z)) \lor (h > 0 \land \psi \neq \text{TRUE} \land \psi \neq \text{STOP} \land \bar{C}(\psi, SC, \psi') \land \exists^*(\psi', P, h - 1) \land P = SC; \text{nondet}(P))$

$\exists^*(\psi, P, h) \equiv P \lor (P \land \exists \psi.\bar{C}(\psi, P, h))$

A constructive proof of $\exists P.\exists^*(\psi, P, h)$ then, as a side-effect, provides the program $P_h = \text{nondet}(P)$ that describes all possible execution traces, i.e. situations, of length $\leq h$ that satisfy the BDF. These definitions lead to a Prolog implementation, able to conduct the constructive proof, producing the corresponding Golog program (cf. Section 5). Some optimization of the generated code is advisable, but for parsimony we omit the rather technical details of this here.

**Soundness**
The soundness of our compilation method follows from the semantics of our preference language.

**Theorem 1. (Soundness)** Let $\psi$ be a basic desire formula and $P_h$ be the corresponding program for horizon $h$. Then for any situation $s_h = do(x_1, a_2, \ldots, a_k, s)$ such that $D \models Do(P_h^c, s, s_h)$, it holds that $D \models \psi[s, s_h]$.

**Proof Sketch:** The proof proceeds by double induction over the structure of basic desire formulae and the length of the situation term. The base case for the structural induction is:

- $f \in F$: as we have $\bar{C}(f, f?, TRUE)$ and by hypothesis know that $D \models Do(P_h^c, s, s_h)$ we have from the definition of $Do$ (Golog semantics) that $f[s]$ and thus $f[s, s_h]$;

- $\text{occ}(a)$: $\bar{C}(\text{occ}(a), \text{nondet}([a]), TRUE)$ enforces that $a_1 = a$ and thus $\text{occ}(a)[s, s_h]$:
transition semantics is axiomatized through two predicates intersecting the results. It is however, grams are evaluated completely first. This motivates the use of our new DT-Golog transition semantics. Unfortunately, space precludes us from stating all but an example of the necessary definitions:

\[ \text{BestTrans}(\text{nondet}(\{s\})), s, d, \pi, v, \text{prob}, B, D) \equiv \]
\[ B = \left[ \begin{array}{c} \text{BestTrans}(\text{nondet}(\{s', s\})), s', d, \pi, v, \text{prob}, B_1, D_1) \land \text{Final}(s') \land \text{Final}(s) \end{array} \right] \]
\[ D = \left[ \begin{array}{c} \text{BestTrans}(\text{nondet}(\{s\}), s', d, \pi, v, \text{prob}, B_2, D_2) \land \end{array} \right] \]
\[ \pi_1, v_1, \text{prob}_{11}, \pi_2, v_2, \text{prob}_{22} \]

The program sync allows the two programs to execute in a new situation \( s \) to a new situation \( s' \) if both programs \( \sigma_1 \) and \( \sigma_2 \) can perform a transition to \( s' \) or when \( s' = s \) and one of \( \sigma_1 \) and \( \sigma_2 \) can do a transition that does not affect the situation, for example evaluating a test. In both cases, the program that remains to be run will be the synchronous execution of the two remaining subprograms \( (\sigma_1', \sigma_2') \). To synchronize more than two programs we can use nesting, so for instance \( \text{sync}(\sigma_1, \text{sync}(\sigma_2, \sigma_3)) \) would synchronize three programs.

The following theorem follows immediately from above definitions.

**Theorem 3.** Let \( \sigma_n, \sigma_o \) be two Golog programs. Then for any \( S', D := \text{Do}(\sigma_n, S_0, S') \land \text{Do}(\sigma_o, S_0, S') \) if and only if \( D := \text{Do}(\text{sync}(\sigma_n, \sigma_o), S_0, S') \).

The theorem states that if there is a situation \( S' \) that describes a legal execution in both programs starting in \( S_0 \), then this is also a legal execution for the synchronization of the two programs. Further, the inverse also holds, saying that any legal execution of the synchronization is also legal for the two individual programs.

### 4.2 Multi-Program Synchronization

Now that we have a Golog program enforcing satisfaction of a BDF, we want to combine this with a pre-existing agent program or another BDF-induced program to eventually provide a semantics for our qualitative preference formulae. To this end, we define multi-program synchronization.

Roughly, we understand two programs to execute synchronously if they traverse the same sequence of situations. Thus, at each step we need to find a common successor situation for both programs. This can be done efficiently by determining the successors of both individually and then intersecting the results. It is however, not efficient if both programs are evaluated completely first. This motivates the use of a transition semantics as opposed to the evaluation semantics originally used to define DT-Golog.

A transition semantics for Golog was first introduced in [6] where, for the same reasons as above, it was used to define the concurrent execution of two programs. Roughly, a transition semantics is axiomatized through two predicates \( \text{Trans}(\sigma_1, s, \sigma_1', s') \) and \( \text{Final}(\sigma, s) \). The former defines for a program \( \sigma \) and a situation \( s \) the set of possible successor configurations \( (\sigma_1', s') \) according to the action theory. The later defines whether a program is final, i.e. successfully terminated, in a certain situation. For instance, for the program \( \sigma_1; \sigma_2 \), that is the sequence of actions \( \sigma_1 \) and \( \sigma_2 \), and a situation \( s \), \( \text{Trans}(\sigma_1; \sigma_2, s, \sigma_2, s) \) describes the only possible transition and is only possible if the action \( \sigma_1 \) is possible in situation \( s \) according to the action theory. Using the transitive closure of \( \text{Trans} \), denoted \( \text{Trans}^* \), one can define a new \( \text{Do} \) predicate as follows:

\[ \text{Do}(\delta, s, s') \triangleq \exists \delta'. \text{Trans}^*(\delta, s, \delta', s') \land \text{Final}(\delta', s'). \]

As is shown in [6], this definition is equivalent to the original \( \text{Do} \). Thus, all results for the one semantics hold equally for the other.

In transition semantics we can formally define the synchronization of two programs \( \sigma_1, \sigma_2 \) by a new Golog construct:

\[ \text{sync}(\sigma_1, \sigma_2) \]:

\[ \text{Trans}(\text{sync}(\sigma_1, \sigma_2), s, \text{sync}(\sigma_1', \sigma_2')) \]

\[ \land \left( \text{Trans}(\sigma_1, s, \sigma_1', s') \land \text{Trans}(\sigma_2, s, \sigma_2', s') \right) \land \text{Final}(s') \land \text{Final}(s) \]

The program \( \text{sync}(\sigma_1, \sigma_2) \) can perform a transition in a situation \( s \) to a new situation \( s' \) if both programs \( \sigma_1 \) and \( \sigma_2 \) can perform a transition to \( s' \) or when \( s' = s \) and one of \( \sigma_1 \) and \( \sigma_2 \) can do a transition that does not affect the situation, for example evaluating a test. In both cases, the program that remains to be run will be the synchronous execution of the two remaining subprograms \( (\sigma_1', \sigma_2') \). To synchronize more than two programs we can use nesting, so for instance \( \text{sync}(\sigma_1, \text{sync}(\sigma_2, \sigma_3)) \) would synchronize three programs.

The following theorem follows immediately from above definitions.

**Theorem 3.** Let \( \sigma_n, \sigma_o \) be two Golog programs. Then for any \( S', D := \text{Do}(\sigma_n, S_0, S') \land \text{Do}(\sigma_o, S_0, S') \) if and only if \( D := \text{Do}(\text{sync}(\sigma_n, \sigma_o), S_0, S') \).

The theorem states that if there is a situation \( S' \) that describes a legal execution in both programs starting in \( S_0 \), then this is also a legal execution for the synchronization of the two programs. Further, the inverse also holds, saying that any legal execution of the synchronization is also legal for the two individual programs.

### A Decision-Theoretic Transition Semantics

As stated above, DT-Golog is defined using an evaluation semantics and that does not suit our requirements. Thus, we have to redefine DT-Golog in an equivalent transition semantics, or, seen differently, extend the available transition semantics to decision-theoretic planning. The semantics follows intuitively from the established relationship between the two semantics. In this section we provide an overview of our new DT-Golog transition semantics. Unfortunately, space precludes us from stating all but an example of the necessary definitions:

\[ \text{BestTrans}(\text{nondet}(\{s\}), s, d, \pi, v, \text{prob}, B, D) \equiv \]
\[ \text{BestTrans}(\text{nondet}(\{s\}), s, d, \pi, v, \text{prob}, B, D_1) \land \text{Final}(s') \land \text{Final}(s) \]

The program \( \text{sync} \) allows the two programs to execute in a new situation \( s \) to a new situation \( s' \) if both programs \( \sigma_1 \) and \( \sigma_2 \) can perform a transition to \( s' \) or when \( s' = s \) and one of \( \sigma_1 \) and \( \sigma_2 \) can do a transition that does not affect the situation, for example evaluating a test. In both cases, the program that remains to be run will be the synchronous execution of the two remaining subprograms \( (\sigma_1', \sigma_2') \). To synchronize more than two programs we can use nesting, so for instance \( \text{sync}(\sigma_1, \text{sync}(\sigma_2, \sigma_3)) \) would synchronize three programs.

The following theorem follows immediately from above definitions.

**Theorem 3.** Let \( \sigma_n, \sigma_o \) be two Golog programs. Then for any \( S', D := \text{Do}(\sigma_n, S_0, S') \land \text{Do}(\sigma_o, S_0, S') \) if and only if \( D := \text{Do}(\text{sync}(\sigma_n, \sigma_o), S_0, S') \).

The theorem states that if there is a situation \( S' \) that describes a legal execution in both programs starting in \( S_0 \), then this is also a legal execution for the synchronization of the two programs. Further, the inverse also holds, saying that any legal execution of the synchronization is also legal for the two individual programs.

### 4.3 Expressing Preference in DT-Golog

In previous sections we showed how to compile BDFs into hard constraints, realized as Golog programs. To make these constraints soft and to rank these constraints to eventually create ordered preferences we need to introduce two more Golog constructs:

- **withPref\( (\sigma_{pr}, P_{pr}) \)**: run program \( \sigma_{pr} \) and try to synchronously run \( P_{pr} \), the result of compiling BDF \( \psi \). This is implemented by creating two branches one with the remaining program \( \text{sync}(\sigma_{pr}, P_{pr}) \) and one with \( \sigma_{pr} \). We devise
the interpreter so that the first branch will be explored first. Only if it fails is the second branch explored.

• \( \text{pref}(P^h_{\psi_1}, P^h_{\psi_2}) \): Let \( \psi_1, \psi_2 \) be two BDFs and \( P^h_{\psi_1}, P^h_{\psi_2} \) their corresponding Golog programs as acquired by compilation. Then \( \text{pref}(P^h_{\psi_1}, P^h_{\psi_2}) \) gives semantics to the qualitative preference formula \( \psi_1 \& \psi_2 \) by creating three branches of decreasing preference: \( \text{sync}(P^h_{\psi_1}, P^h_{\psi_2}), P^h_{\psi_1}, \) and \( P^h_{\psi_2} \). Again, the later branches are only explored if no plan is found for the first. Formally this intuition is captured by extending \( \text{BestTrans} \) such that it defines clusters of branches (and corresponding decisions) of equal degree of preference. Then all previously seen Golog constructs return exactly one cluster of (possibly multiple) branches and the above two constructs return two, respectively three clusters:

\[
\text{BestTrans} \left[ \text{withPref} \left( \sigma \right), P^h_{\psi_1}, P^h_{\psi_2} \right] \quad \text{satisfies} \quad \text{BestTrans} \left[ \text{withPref} \left( \sigma \right), P^h_{\psi_1}, P^h_{\psi_2} \right] = \begin{cases} \text{BestTrans} \left[ \text{withPref} \left( \sigma \right), P^h_{\psi_1}, P^h_{\psi_2} \right] & \text{if successful} \\ \text{BestTrans} \left[ \text{withPref} \left( \sigma \right), P^h_{\psi_1}, P^h_{\psi_2} \right] & \text{if failed} \end{cases}
\]

The preference over former clusters is formally defined in the evaluation strategy of clusters and branches:

\[
\text{BestTrans}^* (B, h) \equiv B = (\sigma, s, d, [\pi, v, prob]) \land \\
(d < h \land \text{BestTrans} (\sigma, s, d, [\pi, v, prob], B, h) \land \\
\forall (d > h \land \pi = nil \land \text{Reward}(\sigma)[p] \land v = r \land \text{prob} = 1)
\]

The following corollary follows from the soundness and completeness of our compilation. Theorem 3, and the correctness of above decision-theoretic transition semantics.

**Corollary 1.** Let \( \sigma \) be an arbitrary Golog program, \( \Psi \) a qualitative preference formula, and \( h \in \mathbb{N} \) a horizon. Let further \( P^h_{\Psi} \) be such that \( \Omega (\Psi, P^h_{\Psi}, h) \). Then any constructive proof of

\[ D \models \exists \pi, v, \text{prob} \quad \text{BestTrans}^* ((\text{withPref}(\sigma, P^h_{\Psi}), S_0, 0, [\pi, v, \text{prob}]), h) \]

as a side-effect returns a policy\(^9\) \( \pi \) which has the following properties:

• any successful execution of \( \pi \) leads to a situation that is most preferred among all possible situations, i.e., the set of situations of length \( \leq h \) which describe a legal execution trace for \( \sigma \) according to the action theory \( D \) and there is no situation \( s^* \) in this set that is more preferred;

• \( \pi \) maximizes the expected reward according to the utility theory.

In other words, \( \pi \) is the best we can do with respect to satisfying the hard constraints in the first place, generating the most qualitatively preferred plan in second place, and finally maximizing the quantitative expected reward in third place.

### 5 Implementation and Application

As noted previously, we have implemented the approach reported in this paper as an extension to Readagol [10]. We have also turned our travel agency example into a working application by creating wrappers for the flight and hotel pages of Yahoo!'s Travel. Recall the planning procedure from Section 2. The actions

\[ \text{searchFlight}(\text{From}, \text{To}, \text{OutDate}, \text{ReturnDate}) \]

\( ^9 \text{eco} \) is a fluent stating that the transportation is economical.

\( ^{10} \text{i.e.} \) a Golog program without any non-deterministic choices
With respect to the quality of the results generated from our implementation, our theoretical results and correctness of the implementation (which we do not prove) ensure that the travel plan generated is optimized with respect to a user’s quantitative preferences, within the best realization of their qualitative preferences. No benchmarks exist for the empirical evaluation of our system, nor was it our objective to optimize our implementation. Nevertheless, as an illustration of the power of our system, we argue that our implementation enables a level of customization of travel planning (and more generally, agent programming), heretofore unattainable in an automated system. For example, in the described case, for each of the 9 date combinations there are over 90 hotels with about 5 room types each and 9 flights. To gather all relevant information, the system issues more than 800 queries to Yahoo!-Travel, considers 36450 combinations, and returns the most preferred travel plan. Manually this would not be feasible and existing systems, although allowing customization to a certain extent, cannot account for the complex preferences a customer may have. We now can! We intend to make this application available as a service at our website.

6 Summary and Discussion

Motivated by the need to personalize agent programs to meet individual users’ preferences and constraints, we addressed the problem of integrating non-Markovian qualitative user preferences with quantitative decision-theoretic planning in Golog. We approached the problem by compiling preferences into Golog programs using a notion of multi-program synchronization which we introduced. This required the redefinition of DT-Golog using a transition semantics which, as a nice side-effect, enables the implementation of more efficient and any-time solution algorithms. We proved the soundness and completeness of our compilation. The resulting system is able to handle infinite state spaces and allows for an efficient programmatic restriction of planning tasks using Golog’s procedural expressiveness. Also, this is, to the best of our knowledge, the first work on integrating qualitative and quantitative preferences for temporal reasoning. We implemented our approach, and as a demonstration of its utility developed a customizable travel planner for the Web. The results in this paper are applicable to both symbolic and decision-theoretic agent programming systems, and may be used not only for the personalization of agent programs, but also for the realization of defeasible control strategies for planning.

References


10Technically speaking these are so-called sensing actions, but space preclude a thorough discussion of this issue. The interested reader is referred to the literature, e.g. [14].


**Action progression and revision in multiagent belief structures**

Andreas Herzig  
IRIT - Université Paul Sabatier  
31062 Toulouse Cedex, France  
herzig@irit.fr

Jérôme Lang  
IRIT - Université Paul Sabatier  
31062 Toulouse Cedex, France  
lang@irit.fr

Pierre Marquis  
CRIL - Université d’Artois  
62307 Lens Cedex, France  
marquis@cril.univ-artois.fr

---

**Abstract**

We present a model for progressing and revising doxastic belief states in a multiagent setting. The model is sophisticated enough to deal not only with both ontic and epistemic actions, but also to handle the case where it is not common belief that an action occurred or that an observation has been made. Our model includes a multiagent extension of AGM-style belief revision.

---

1 Introduction

Reasoning about action and change have received an enormous attention in the last fifteen years, and has resulted in several families of languages, such as propositional action languages or causal theories, the situation calculus, the fluent calculus, etc. All these languages have been extended so as to reason with incomplete knowledge and sensing.

The fact that the initial belief state may be incomplete and the actions may be nondeterministic leads to the necessity to distinguish formally between facts and beliefs. Epistemic or doxastic logic have been used in several places to reason about action and change, e.g. [Lakemeyer and Levesque, 1998; Baral and Son, 2001; Herzig et al., 2003; Scherl and Levesque, 2003]. A common limitation of this series of works is that they consider only one agent (and nature, since the agent evolves in a nondeterministic world).

Now, many domains involve several agents, who are expected to interact (or more specifically negotiate, cooperate, communicate). Interacting in an efficient way requires to reason about other agents’ beliefs and the evolution of these beliefs after the occurrence of some actions or events. What renders things complex when several agents are considered is the fact that agents’ beliefs must include not only first-order beliefs, i.e., beliefs about the world, but also higher-order beliefs, i.e., beliefs about the agents’ beliefs. Furthermore, agents must be able to make higher-order beliefs evolve in the light of new information, i.e., they must also possess update and revision procedures for such beliefs.

While some works exist so as to model the evolution of a multiagent belief model after some communication actions are performed, there are only few works where usual physical and sensing actions – as considered in cognitive robotics – are considered in a multiagent setting. This paper contributes to fill the gap. For the sake of the exposition we suppose there are only two agents. It is not a loss of generality, in so far as all problems related to mutual belief, common belief, and communication are already present with two agents and are conceptually no more complex than with more than two agents. Moreover, all definitions and results of this paper can readily be extended to the $N$-agent case.

As it is often the case in the literature, we assume for the sake of simplicity that the set of actions available to the agents is partitioned into two subsets: purely ontic (or physical) actions may change the state of the world but do not bring any feedback, whereas purely epistemic (or sensing) actions leave the state of the world unchanged and may only bring some feedback about it. This does not induce a loss of generality, since more general actions, with both effects on the world and feedback, can be decomposed into two actions, one being purely ontic and the other one purely epistemic.

Belief models are taken to be Kripke structures of 2-agent doxastic logic $\text{KD45}^2$ on which Section 2 gives some background. Section 3 considers ontic actions (without feedback); we first define the progression of a belief state by an action whose occurrence is assumed to be common knowledge, and then we consider the general case where this assumption is relaxed. Section 4 focuses on observations and sensing actions. We define the progression of a belief model by an epistemic action model where agents may perform sensing actions (and thus gather observations) while others do not. Since initial beliefs of the agents are not required to be correct, agents have to perform a (mutual) belief revision process; accordingly, our progression by sensing actions involves a genuine extension of AGM revision. Section 5 discusses related work.

2 Belief models

We first give some basics about 2-agent propositional doxastic logic $\text{KD45}^2$ (see e.g. [Fagin et al., 1995]).

**Definition 1 (language)** Let $\text{AtProp} = \{p_1,\ldots,p_n\}$ be a finite set of propositional symbols (atoms). The language $\text{L}^{1,2}_{\text{AtProp}}$ of $\text{KD45}^2$ is built inductively from $\text{AtProp}$, constant symbols $\top$ and $\bot$, the connectives $\neg, \land, \lor$ and modal operators $B_1, B_2, CB_{1,2}$ in the standard way.

---

6A preliminary version of this paper has been presented at the 6th Workshop on Logic and the Foundations of Game and Decision Theory (LOFT-2004).
Formulas of $\mathcal{L}^{1,2}_{\text{AtProp}}$ are denoted by capital Greek letters $\Phi$, $\Psi$, etc.; objective (i.e., modality-free) formulas are denoted by small Greek letters $\varphi$, $\psi$, etc. Objective formulas are interpreted in a classical way:

**Definition 2 (states)** States $= \mathcal{S}$ is the set of propositional valuations, or states. $\text{States}(\varphi)$ denotes the set of states which classically satisfy the objective formula $\varphi$.

States are denoted by $s$, $s'$ etc. Although states are formally sets of atoms, we prefer to denote them in the following form: if $\text{AtProp} = \{a, b, c\}$ then the state $\{a, c\}$ is denoted by $[a, \neg b, c]$ it assigns $a$ and $c$ to true and $b$ to false.

**Definition 3 (pointed belief models)** A pointed belief model (PBM) $\mathcal{M}$ for KD45 is a tuple $\mathcal{M} = (W, \text{val}, R_1, R_2, w^*)$ where:

- $W = \{w, v, \ldots\}$ is a nonempty set of possible worlds;
- $\text{val} : W \to \text{States}$ maps possible worlds to states;
- $R_1, R_2$ are binary relations on $W$ satisfying seriality $(\forall i, w, R_i(w) \neq \emptyset)$, transitivity, and Euclideanity (if $wR_iw'$ and $wR_iw''$ then $w'R_iw''$);
- $w^* \in W$ is a distinguished world (the actual world).

Note that $R_i$ is not required to be reflexive (which implies that agents may hold wrong beliefs).

The *subjective content* of a pointed belief model $\mathcal{M} = (W, \text{val}, R_1, R_2, w^*)$ is the tuple $\mathcal{M} = (W, \text{val}, R_1, R_2)$ and is simply called a (nonpointed) belief model.

We are now ready to define the notion of satisfaction of a formula by a pointed belief model:

**Definition 4 (satisfaction by a PBM)** Satisfaction of formulas of $\mathcal{L}^{1,2}_{\text{AtProp}}$ in a (nonpointed) belief model $\mathcal{M} = (W, \text{val}, R_1, R_2)$ at world $w \in W$ is defined inductively in the usual way:

- if $\Phi$ is objective then $M, w \models \Phi$ iff $\text{val}(w) \models \Phi$.
- $M, w \models \neg \Phi$ iff $M, w \not\models \Phi$.
- $M, w \models \Phi \land \Psi$ iff $M, w \models \Phi$ and $M, w \models \Psi$ (and similarly for the other connectives);
- for $i \in \{1, 2\}$, $M, w \models \text{B}_i \Phi$ iff $M, w' \models \Phi$ for all $w' \in R_i(w)$.
- $M, w \models \text{CB}_1 \text{B}_2 \Phi$ iff $M, w' \models \Phi$ for all $w' \in R_{CB}(w)$, where the accessibility relation for common belief $R_{CB}$ is defined as usual by $R_{CB} = (R_1 \cup R_2)^*$ (* denotes reflexive and transitive closure).

Satisfaction of a formula $\Phi$ of $\mathcal{L}^{1,2}_{\text{AtProp}}$ in the pointed belief model $\mathcal{M} = (W, \text{val}, R_1, R_2, w^*)$ is finally given by $\mathcal{M} \models \Phi$ iff $M, w^* \models \Phi$.

3 Progression of belief models by ontic actions

3.1 Ontic actions

A (purely) ontic action is characterized by the fact that it does not bring any feedback. Examples are: tossing a coin without observing the outcome, sending an email to somebody without knowing whether it will be received by the addressee. In terms of belief states, there is no need to distinguish between the projection of agent $i$’s belief state by the action before the action is performed, and $i$’s belief state after the action: what $i$ foresees is what she gets. An ontic action $\alpha$ is a “state transformer”, i.e., a transition relation on states, alias valuations: to every $\alpha$ there is associated a function $f_{\alpha}$ mapping valuations $s \in \text{States}$ to sets of valuations: $s^\alpha \subseteq \text{States}$ is the set of possible successor states of $s$ after $\alpha$.

**Definition 5 (ontic actions)** An ontic action $\alpha$ is a binary relation on $\text{States}$ s.t. the direct image $s^\alpha$ of every state $s \in \text{States}$ by $\alpha$ is a nonempty set. $\text{AtAct}_O$ denotes the set of all ontic actions.

For the sake of simplicity, we assume that actions are fully executable, that is, $s^\alpha \neq \emptyset$.

In the rest of the paper we make use of the action $\beta$ of switching $p$, defined on $\text{AtProp} = \{p\}$ by $[p]^\beta = \{[\neg p]\}$ and $[-p]^\beta = \{[p]\}$, and of the void action $\lambda$, which is defined by $s^\lambda = \{s\}$ for all $s \in \text{States}$.

3.2 Common knowledge of action occurrences

In this section we assume that the initial beliefs of the agents are expressed by a belief model $\mathcal{M}$, and that it is common knowledge to the agents that action $\alpha$ is being performed (for instance because one of them publicly announces that she is performing $\alpha$). The *progression* of $\mathcal{M}$ by $\alpha$ is the intended new belief model $\mathcal{M}^\alpha$ expressing the beliefs of the agents after action $\alpha$ has been performed. The intuition behind the progression of $\mathcal{M}$ by $\alpha$ is the following: $\mathcal{M}^\alpha$ is obtained from $\mathcal{M}$ by replacing in $\mathcal{M}$ each world $w$ by a set of worlds $(w, s)$, such that $s$ is a possible state resulting from the application of $\alpha$ in $\text{val}(w)$. Accessibility relations are then “transferred” from the old worlds to the new ones.

**Definition 6 (progression of a PBM by an ontic action)** Let $\mathcal{M} = (W, \text{val}, R_1, R_2, w^*)$ be a pointed belief model and $\alpha \in \text{AtAct}_O$. The progression of $\mathcal{M}$ by $\alpha$ is the set of pointed belief models

$$\mathcal{M}^\alpha = \{ (W^\alpha, \text{val}^\alpha, R_1^\alpha, R_2^\alpha, (w^*, s)) \mid s \in \text{val}(w^*)^\alpha \}$$

where

- $W^\alpha = \{ (w, s) \mid w \in W, s \in \text{val}(w)^\alpha \}$;
- $\text{val}^\alpha((w, s)) = s$;
- $(w, s) R_1^\alpha (v, t)$ if and only if $w R_1 v$.

The reason why $\mathcal{M}^\alpha$ is generally not a single pointed belief model but a set is that actions may be nondeterministic, so that the actual state $\text{val}(w^*)$ may have several possible successor states by $\alpha$. If $\alpha$ is deterministic, then obviously $\mathcal{M}^\alpha$ is a singleton.

3.3 The general case

In the previous section, we supposed that action occurrences are perceived completely and correctly by every agent, which is often unrealistic: some agents may be unaware that an action is being performed, or may just suspect that some action is being performed, or may know that an action from a given set is being performed, without knowing precisely which one (see [Baltag et al., 1998] for a extended discussion on this issue, for the case of epistemic actions). In order to relax this constraint we extend Baltag et al.’s ideas and use doxastic action structures.
Definition 7 (pointed doxastic action structures) A pointed doxastic action structure (PDAS) is a tuple 
\[ \mathcal{A} = (W_A, \text{act}, S_1, S_2, a^*) \]
where
- \( W_A = \{a, b, \ldots\} \) is a set of possible “action worlds”;
- \( \text{act} : W_A \to \text{AtAct}_O \) is a total function that maps possible action worlds to actions;
- \( S_1, S_2 \) are binary relations on \( W_A \) that are serial, transitive, and Euclidean.
- \( a^* \in W_A \) is the actual action world – the occurrence of which some of the agents may be unaware.

Subsection 3.2 corresponds to the particular case where \( W_A = \{a^*\} \) and \( S_1(a^*) = S_2(a^*) = \{a^*\} \).

\( S_i \) relates an action to an agent \( i \)'s “subjective versions” of \( a \): if \( a_i b \) and \( a \) occurs then in \( i \)'s view \( b \) is one of the actions that might have happened. In this way one can model incomplete and erroneous perception.

Example 1 Let \( \beta = \text{switch}(p) \) and \( \lambda \) as defined in Section 3.1, and \( \mathcal{A} = (W, \text{val}, R_1, R_2, w^*) \) by a PDAS \( \mathcal{A} = (W_A, \text{act}, S_1, S_2, a^*) \) is a set of pointed belief models
\[ \mathcal{M}^d = \{\{W_A, \text{val}^{A_1}, R_1^{A_1}, R_2^{A_1}, \langle w^*, a^*, s \rangle \} \mid s \in \text{val}(w^*)^{\text{act}(a^*)}\} \]
where
- \( W_A = \{\langle w, a, s \rangle \mid w \in W, a \in W_A, s \in \text{val}(w)\text{act}(a)\} \)
- \( \text{val}^A(\langle w, a, s \rangle) = s \)
- \( \langle w, a, s \rangle R_i^A(\langle w, b, t \rangle) \iff w R_i v \) and \( a S_i b \).

Intuitively, the world \( \langle w, a, s \rangle \) encodes that the execution of action \( a \) in state \( w(a) \) results in state \( s \).

In order to make sure that this definition is well-founded we first have to check that each element of \( \mathcal{M}^d \) is actually a PBM, i.e., \( W_A \) is nonempty and each \( R_i^A \) is serial, transitive, and Euclidean.

Proposition 1 \( \mathcal{M}^d \) is a set of a KD45 point belief models.

Example 2 Let \( \text{AtProp} = \{p, q\} \), \( \mathcal{A} \) as in Example 1, and \( \mathcal{M} = (W, \text{val}, R_1, R_2, w^*) \) by a PDAS \( \mathcal{M} = (W_A, \text{act}, S_1, S_2, a^*) \) is a set of pointed belief models
\[ \text{val}(w_0) = p; \text{val}(w_1) = \neg p; R_1(w_0) = \{w_0\}; R_2(w_0) = \{w_1\}; R_1(w_1) = R_2(w_1) = \{w_1\} \]

The progression of \( \mathcal{M} \) by \( \mathcal{A} \) is the singleton \( \mathcal{M}^d = \{\langle W^d, \text{val}^{A_1}, R_1^{A_1}, R_2^{A_1}, \langle w_0, a_0, \neg p \rangle \rangle \} \) depicted on the figure below; the left part of the figure is \( \mathcal{M} \), the upper part \( \mathcal{A} \), and the bottom-right part is \( \mathcal{M}^d \); actual worlds are labeled by \( * \).

In \( \mathcal{M}^d \), both 1 and 2 (wrongly) believe \( p \) because he used to correctly believe \( p \) and believes nothing happened, and 2 because he used to believe \( \neg p \) and knows that \( p \) has been switched. But this is not common belief since 1 believes that 2 believes \( \neg p \) and 2 believes that 1 believes \( \neg p \).

The progression of a belief model by a doxastic action structure recovers single-agent progression as a particular case, and slightly more generally, if in \( \mathcal{M} \) the agents have full common belief that the objective formula \( \phi \) holds, and have common knowledge that action \( a \) occurs, then in \( \mathcal{M}^d \) the agents commonly believe \( \text{prog}(\phi, \alpha) \), where \( \text{prog} \) refers here to classical propositional progression at the syntactic level. (Semantically, we have \( \text{States}(\text{prog}(\phi, \alpha)) = \bigcup_{s \in \text{States}(\phi)} s^a \).

4 Progression by epistemic actions

4.1 Observation actions

We now consider a set \( \text{AtAct}_E \) of elementary epistemic actions, or observation actions, of the form \( \text{observe}(\Phi) \), where \( \Phi \) is a formula of \( L^{1,2} \) the action of observing that \( \Phi \) holds.

Observations can be made either “spontaneously” by the agent, or after performing a sensing action such as, typically, a test \( \text{sense}(\Phi) \) sending back the truth value of \( \Phi \). (We shall see later that observation actions can express sensing actions).

We consider the possibilities of observing (and sensing) both objective formulas, as in cognitive robotics (such as “test whether this solution is an acid”), and subjective formulas (such as “ask agent 2 whether she believes the solution is an acid”, or “ask agent 2 whether she believes whether agent 3 knows whether the solution is acid or not”).

Importantly, the observation \( \Phi \) is executable only in worlds where \( \Phi \) is true (we’ll come back on this later.) This restriction merely expresses that observations are reliable.

We now introduce pointed observation structures (POS), which are similar to PDAS, except for one thing: while it is meaningful to talk about the “actual” ontic action being performed, this is no longer so for epistemic actions (observations), since they are intrinsically “subjective”, that is, it is meaningless to talk about an observation (resp. a sensing action) without referring to the agent who performs it. However, we may talk about the actual epistemic action performed by a given agent \( i \). This leads us to consider the following pointed observation structures, which resemble our PDAS of Section
Definition 9 (pointed observation structures)  
A pointed observation structure (POS) is a tuple $\mathcal{O} = \langle W_O, obs, T_1, T_2, o^* \rangle$, where  

1. $W_O$ is a set of observation worlds, denoted $o, o'$ etc.  
2. $T_1$ and $T_2$ are serial, transitive and Euclidean relations on $W_O$;  
3. $obs : W_O \times \{1, 2\} \rightarrow \text{AtAct}_E$ maps each observation world and each agent to an observation action;  
4. $o^* \in W_O$.

$\mathcal{O}$ must satisfy the following constraints, for all $o \in W_O$:  
1. $\text{obs}(o \land 1) \land \text{obs}(o \land 2)$ is consistent;  
2. for all $i \in \{1, 2\}$, if $oT_1 o'$ then $\text{obs}(o', i) = \text{obs}(o, i)$.

For $i \in \{1, 2\}$ and $o \in W_O$, $\text{obs}(o, i)$ is the observation made by agent $i$ at $o$. In particular, $\text{obs}(o^*, i)$ is the observation actually performed by $i$.

The constraint that $\text{obs}(o, 1) \land \text{obs}(o, 2)$ be consistent is required by our hypothesis that observations are truthful. It is not a sufficient condition: truthfulness is actually not expressible in a POS, since it refers as well to the states; it will only be manifest in the product of a POS and a belief model (see further). The second constraint ensures that every agent knows the observation she performs.

Although observations are truthful, the observation structure is not necessarily reflexive: while an agent is correct about her own observations, she might be wrong about other agents’ observations. A nonreflexive epistemic observation structure may for instance represent the fact that agent 1 believes that agent 2 is sensing $p$ while she is actually not – this is totally different from getting a wrong observation.

Pointed observation structures on observation actions can represent sensing actions. For instance, the POS $\mathcal{O} = \langle W_O, obs, T_1, T_2, o^* \rangle$ where $W_O = \{o_1, o_2\}$, $o^* = o_1$, $T_1(o_1) = T_1(o_2) = \{o_1, o_2\}$, $T_2(o_1) = \{o_1\}$, $T_2(o_2) = \{o_2\}$, $\text{obs}(o_1, 1) = 1$, $\text{obs}(o_2, 2) = 0$, $\text{obs}(o_2, 1) = 1$, $\text{obs}(o_2, 2) = 0$, $\text{obs}(o_2, 2) = \neg p$ represents a situation where 1 does not observe anything and knows that 2 tests the truth value of $p$ (and all this is common knowledge) – which means that she observes either $p$ or $\neg p$, according to the actual truth value of $p$.

The latter will be ensured by the fact that it is impossible to observe $\Phi$ when $\Phi$ is false.

We are now in position of defining the progression of a PBM by an POS. We first work out the case where agents have correct beliefs (i.e., $\mathcal{M}$ is a $\mathbb{SS}_5^C$ pointed model), because it is simpler and comes to a syntactical progression operator. The case of possibly incorrect beliefs needs a belief revision phase, and will be considered later.

4.2 Progression by epistemic actions in $\mathbb{SS}_5^C$

We assume in this section that $\mathcal{M}$ is a $\mathbb{SS}_5^C$ PBM model (both $R_1$ and $R_2$ are reflexive), and $\mathcal{O}$ is a $\mathbb{SS}_5^C$ POS-model (both $T_1$ and $T_2$ are reflexive). We note $M$ the (nonpointed) belief model associated to $\mathcal{M}$. Progression of $\mathcal{M}$ by $\mathcal{O}$ amounts to construct a restricted product of $\mathcal{M}$ and $\mathcal{O}$. Unlike in the case for ontic actions, the result is a unique pointed belief model.

Definition 10 (progression of a $\mathbb{SS}_5^C$ PBM by a POS)  
Let $\mathcal{M} = (W, val, R_1, R_2, w^*)$ be a pointed $\mathbb{SS}_5^C$-model and $\mathcal{O} = (W_O, obs, T_1, T_2, o^*)$ a pointed observation structure such that $\mathcal{M} \models obs(o^*, i)$ for $i = 1, 2$. The progression of $\mathcal{M}$ by $\mathcal{O}$ is

$\mathcal{M}^o = \langle W^o, val^o, R_1^o, R_2^o, w^stocks^o \rangle$

where  
1. $W^o = \{w, o | w \in W, o \in W_O, and i = 1, 2, M, w \models obs(o, i)\}$,  
2. $val^o(\langle w, o \rangle) = val(w)$,  
3. $\langle w, o \rangle R_i^o \langle w', o' \rangle$ if and only if $wR_iw', oT_1o'$, and $M, w' \models obs(o, i)$,  
4. $w^stocks^o = \langle w^stocks, o^* \rangle$.

Hence from all possible combinations of worlds and observations, only those from $W^o$ which have truthful observations are retained. (As by hypothesis none of the agents can make erroneous observations this set is nonempty.) Moreover, for a given world $\langle w, o \rangle$ only those worlds $\langle w', o' \rangle$ are accessible for $i$ where $i$’s observation $\text{obs}(o', i)$ is true in $w$.

In this way the accessibility relations are constructed by filtering out accessible worlds (for agent $i$) where the observed formula (by agent $i$) is false: it makes that after observing $\Phi$, agent $i$ believes $\Phi$ (which is, again, justified by the assumption that observations are reliable).

For instance, consider a world where 1, after asking 2 whether she knows whether Mozart was left-handed or not (m.h.), 2 answers “yes, I know” (i.e., the observation is B$_2$ m.h. $\lor$ B$_2$ $\neg$ m.h.), then after progression, 1 knows that 2 knows whether Mozart was left-handed or not.

Proposition 2 If both $\mathcal{M}$ and $\mathcal{O}$ are $\mathbb{SS}_5^C$ belief models then $\mathcal{M}^o$ is an $\mathbb{SS}_5^C$ belief model.

Example 3 Suppose AtProp = $\{p\}$. Let $W = \{w, w\_p\}$, $val(w_p) = [p]$, $val(w\_p) = [\neg p]$ and for all $w \in W$, $R_1(w) = \{w\}$, and $R_2(w) = W$. Let $w^stocks = w_p$. Hence 1 knows that $p$, while 2 ignores whether $p$. The latter is common knowledge.

Let $W_O = \{o_p, o\_p\}$ with $\text{obs}(o_p, 1) = 1$ and $\text{obs}(o_p, 2) = \varphi$, for $\varphi = p, \neg p$; $T_1(o_p) = \{o_p\}$, and $T_2(o_p) = W_O$. Suppose $o^* = o_p$. Hence 1 knows that 2 is testing whether $p$.

We obtain $W^o = \{\langle w, o_p, [p] \rangle, \langle w\_p, o\_p, [\neg p] \rangle\}$, and $R_1^o(\langle w, o, s \rangle) = R_2^o(\langle w, o, s \rangle) = \{\langle w, o, s \rangle\}$ for all $\langle w, o, s \rangle \in W^o$. As expected, after the revision there is common belief that $p$. 
4.3 Progression by epistemic actions in KD45\textsuperscript{C}

We now consider the case where initial beliefs may be incorrect. At that time, this requires to make a simplifying assumption on the nature of epistemic actions, namely, that what is observed are only facts (and not beliefs): an observation (an element of ATAct\textsubscript{\varphi}) is an objective formula \varphi.

We consider an AGM preference structure, i.e., a collection of preference relations \preceq_x for every nonempty subset of states \(X \subseteq S\) verifying the faithfulness condition: \(\forall x \in X, \forall x \in X, \forall y \in S, x \preceq y\). From that structure an AGM revision operation can be defined as follows.

**Definition 11 (AGM revision on objective formulas)**

\(\varphi \preceq \psi\) is the propositional formula unique up to logical equivalence such that

\[
\text{States}(\varphi \preceq \psi) = \min_{\preceq \text{States}(\varphi)}(\text{States}(\psi)).
\]

Such preference-based revision operators are characterized by the AGM postulates [Gärdenfors and Makinson, 1988; Katsuno and Mendelzon, 1991].

Similar to the case of action structures, the revision of a belief model by an observation structure is done by revisiting the belief states of each agent according to her view of the observation.

**Definition 12 (revision of a PBM by a POS)**

Let \(\mathcal{M} = (\mathcal{W}, \text{val}, R_1, R_2, w^*)\) be a PBM and \(\mathcal{O} = (\mathcal{W}_0, \text{obs}, T_1, T_2, o^*)\) a POS such that \(\mathcal{M} \models \text{obs}(o^*, i)\) for \(i = 1, 2\). Let \(\preceq\) be any preference relation. Then

\[
\mathcal{M}^{\mathcal{O}} = (\mathcal{W}^{\mathcal{O}}, \text{val}^{\mathcal{O}}, R_1^{\mathcal{O}}, R_2^{\mathcal{O}}, w^{*^{\mathcal{O}}})
\]

where

- \(\mathcal{W}^{\mathcal{O}} = \{(w, o, s) \mid w \in \mathcal{W}, o \in \mathcal{W}_0, \text{and for } i = 1, 2, s \in \text{States}(\text{obs}(o, i))\};\)
- \(\text{val}^{\mathcal{O}}(\langle w, o, s \rangle) = s;\)
- \(R_i^{\mathcal{O}}(\langle w, o, s \rangle) = \{(w', o', s') \mid w' \in R_i(w), o' \in T_i(o), \text{and } s' \in \min_{\preceq \text{val}(R_i(w))}(\text{States}(\text{obs}(o, i)))\};\)
- \(w^{*^{\mathcal{O}}} = \langle w^*, o^* \rangle.\)

In the definition of \(\mathcal{W}^{\mathcal{O}}\), the condition that \(s \in \text{States}(\text{obs}(o, i))\) guarantees that observations are truthful. (Again, as by hypothesis none of the agents can make erroneous observations the set \(\mathcal{W}^{\mathcal{O}}\) is nonempty.) In the definition of \(R_i^{\mathcal{O}}\), the minimization condition implements preference-based revision. As for \(\mathcal{S}^{\mathcal{O}}\), the result is a unique pointed belief model.

Our definition is well-founded since:

**Proposition 3** \(\mathcal{M}^{\mathcal{O}}\) is a KD45\textsuperscript{C} belief model.

The next proposition establishes that AGM belief revision is a particular case.

**Proposition 4** Let \(\varphi\) be an objective formula, and let \(\mathcal{M}(\varphi) = (\mathcal{W}, \text{val}, R_1, R_2, w^*)\) be a belief model such that \(\mathcal{W} = \text{States}(\varphi), \text{val}(w) = w, R_i(w) = \mathcal{W}\).

Let \(\preceq\) be a preference relation, and let \(\mathcal{O} = (\{o\}, \text{obs}, T_1, T_2, o^*)\) be a pointed observation structure such that for \(i = 1, 2, \text{obs}(o, i) = \psi, T_i(o) = \{o\}, \text{and } o^* = o.\)

Suppose \(o^*\) is reliable, i.e. \(\mathcal{M} \models \psi\), and let \(\mathcal{M}^{\mathcal{O}} = (\mathcal{W}^{\mathcal{O}}, \text{val}^{\mathcal{O}}, R_1^{\mathcal{O}}, R_2^{\mathcal{O}}, w^{*^{\mathcal{O}}})\) be the revised model. Then

\[
\text{States}(\varphi \preceq \psi) = R_1^{\mathcal{O}}(w^*) = R_2^{\mathcal{O}}(w^*).
\]

**Example 4** Suppose \(\text{AtProp} = \{p\}\), and suppose \(\preceq\) is based on the Hamming distance \(\text{dist}(h(s, s'))\), i.e. the number of symbols on which \(s\) and \(s'\) differ. For \(S \subseteq X\), \(\text{dist}(h(s, X)) = \min_{x \in X} \text{dist}(h(s, x))\), and \(\preceq\) is defined by:

\[
\text{states } s \preceq s' \text{ iff } \text{dist}(h(s, X)) \leq \text{dist}(h(s', X)).
\]

Let \(W = \{w_p, w_{\neg p}\}, \text{val}(w_p) = [p], \text{val}(w_{\neg p}) = [-p],\) and for all \(w \in W\), \(R_1(w) = [w]\) and \(R_2(w) = [w_{\neg p}]\). Let \(w^* = w_p\). Then \(\mathcal{M} \models B_1(p \land B_2(CB_{1,2p}))\). Let \(\mathcal{O}\) such that \(W_0 = \{o_p\}, \text{obs}(o_p, i) = p\) and \(T_1(o_p) = \{o_p\}\), \(i = 1, 2\).

Then \(\mathcal{W}^{\mathcal{O}} = \{(w_p, o_p, [p]), (w_{\neg p}, o_p, [p])\}\), and for all possible worlds \(\xi \in \mathcal{W}^{\mathcal{O}}, R_1^{\mathcal{O}}(\xi) = \{\xi\}\) and \(R_2^{\mathcal{O}}(\xi) = \{(w_{\neg p}, o_p, [p])\}\). Hence \(\mathcal{M}^{\mathcal{O}} \models CB_{1,2p}\).

5 Related work and conclusion

A number of papers have considered belief change operators on belief structures based on multiagent KD45 or S5.

A series of works considers multiagent belief expansion, starting with [Fagin et al., 1995], who consider public announcements in S5, and express examples such as the Muddy Children Puzzle (cf. Section 4.2). The issue is then further studied in [Gerbrandy and Groeneveld, 1997] who develop a language for reasoning about information change relative to group announcements (public announcements within a given subset of agents.) [van Ditmarsch et al., 2004] show that under the restriction to positive formulas public announcement coincides with expansion, and that they differ in general.

[Baltag et al., 1998] go much more general and define complex epistemic actions as Kripke frames where worlds are valued by actions, to account for the case where agents have different information about which action is taking place. Our doxastic action and observation structures are taken from theirs (with a few differences explained further). See also [van Ditmarsch, 2002] who consider a more elaborate language of epistemic actions allowing for concurrent execution of epistemic actions by the agent.

[Baltag et al., 1998] only partially allow for incorrect beliefs: more precisely, the initial belief model is S5, whereas complex epistemic actions are KD45: agents are assumed to hold initial correct beliefs, but may have misperceptions or wrong suspicions of communication actions between other agents. The fact that initial beliefs should be correct is important, since it allows for a simple belief expansion process, which filters out the worlds where the observation just made does not hold. However, expanding an S5 model by a KD45 complex epistemic action does generally not result in an S5 model: the final beliefs of the agents might be incorrect – which makes the process impossible to iterate. This lead Aucher [Aucher, 2004] to extend the latter approach so as to deal with possibly erroneous beliefs. His belief models are graded S5 models, consisting of an S5 model together with a function expressing the relative plausibility of the worlds. The revision process then works by retaining, for each set of indistinguishable worlds, the most plausible ones among those which satisfy the observation. The result is still a graded S5 model. The only problem with Aucher’s semantics is that by enforcing that any conceivable world has some plausibility, models are extremely large, even for simple single-agent examples; furthermore, it may be difficult to assign plausibilities in the initial belief model. Our preference relations play...
a similar role to Aucher’s plausibilities, and remain more general and closer to the original AGM framework.

[Tallon et al., 2004] propose an account of multi-agent revision, where agents may have initial erroneous beliefs, under the strong hypothesis that agents communicate publicly all their beliefs, i.e., their belief state as a whole.

The approach in [van der Meyden, 1994], older than and unrelated to the latter ones, allows for incorrect beliefs and revision, but remains very general and does not commit to a precise family of belief revision operators. Observations are common knowledge such as in group announcements.

So far, all reported approaches focus on deterministic epistemic actions, namely observation actions; typical observation actions in these approaches are communication actions consisting in an agent telling another (or a group of others) something she claims to believe. With this class of actions, (a) the objective state of the world does not change and (b) actions are deterministic: they have preconditions (such as, in the case of sincere communication, an agent believing what she tells); when the precondition is satisfied, the action results in the same world, and when the precondition is not satisfied, the action is not executable. Therefore the progression problem (for ontic or sensing actions) is not considered.

Ontic actions, as well as sensing actions (which differ from observation actions in many aspects) are almost never considered in multiagent frameworks. An exception is [Martin et al., 2004] who consider ontic, sensing and communication actions but without any account for higher-order beliefs (more precisely, there is a distinguished agent who holds beliefs about other agent’s beliefs about facts, and nothing more). They assume that beliefs are correct and that actions are public; on the other hand, this work contains many results about the practical computation or the new beliefs. [Shapiro et al., 1998] consider a multiagent extension of the situation calculus where communications actions are taken into account, but agents hold no beliefs about other agents. [Demolombe and Parra, 2002] consider a multiagent extension of the situation calculus with higher-order beliefs, building on the single-agent approach [Shapiro et al., 2000]; they do not give a general model but study a few particular cases.

We mention three other approaches dealing with the dynamics of mutual beliefs; unlike the previous ones, they are not based on general Kripke structures but on simpler structures. [Kfir-Dahav and Tennenholtz, 1998] and [Su et al., 2004] assume that each agent observes (at each instant) a specific subset of propositional variables, and who observed what is common knowledge, and [Liu and Williams, 2001], that each agent chooses to open some parts of its knowledge base to some other agents (knowledge migration consists then in revising one’s knowledge base when accessing other agents’ accessible knowledge.)

Further work will mainly focus on the practical computation of progression and regression, that ultimately would enable logic-based planning in multiagent domains.

References


Using Ranking Functions to Determine Plausible Action Histories

Aaron Hunter and James P. Delgrande
School of Computing Science
Faculty of Applied Sciences
Simon Fraser University
{amhunter, jim}@cs.sfu.ca

Abstract
We use ranking functions to reason about belief change following an alternating sequence of actions and observations. At each instant, an agent assigns a plausibility value to every action and every state; the most plausible world histories are obtained by minimizing the sum of these values. Since plausibility is given a quantitative rank, an agent is able to compare the plausibility of actions and observations. This allows action occurrences to be postulated or refuted in response to new observations. We demonstrate that our formalism is a generalization of our previous work on the interaction of revision and update.

1 Introduction
When reasoning about epistemic action effects, it is useful to draw a distinction between ontic actions and epistemic actions. Ontic actions are actions that change the state of the world, whereas epistemic actions are actions that change the beliefs of an agent without changing the world. Several formalisms have been proposed to represent action domains in which an agent may perform both ontic actions and epistemic actions [SPLL00; HLM04]. These formalisms have focused primarily on the epistemic effects of a single action. In this paper, we consider belief change in the context of alternating sequences of ontic and epistemic actions. The formalism that we introduce is a generalization of our work in [HD05].

Informally, we are interested in alternating sequences of updates and revisions of the form

\[ K \circ A_1 \circ O_1 \circ \cdots \circ A_n \circ O_n \]

where each \( A_i \) is an ontic action and each \( O_i \) is an observation represented by a set of possible worlds. We are particularly interested in the case where action histories and observation histories may both be fallible. In this context, it is necessary for an agent to have some means for resolving conflicts between observations and perceived action histories. For example, if no world in \( O_n \) is possible following the action sequence \( A_1, \ldots, A_n \), then there are two options.

1. Reject \( O_n \).
2. Accept \( O_n \), and modify \( A_1, \ldots, A_n \).

The first option intuitively corresponds to the case where \( O_n \) is less plausible than the action history, and the second option corresponds to the case where it is more plausible. In order to determine which option is preferable for a specific problem, an agent effectively needs to be able to compare the plausibility of \( O_n \) with the plausibility of each \( A_i \). This kind of comparison is only possible if there is a single plausibility ranking over actions and observations.

In this paper, we propose that Spohn-style ranking functions can be used to define a flexible formalism for reasoning about belief change over alternating sequences of actions and observations. The idea is simply to give a subjective ranking of actions and observations at each point in time. By looking at this sequence of rankings, an agent is able to determine the most plausible world histories. We demonstrate the utility of our new formalism by example and by comparison with related formalisms. We also demonstrate that this is indeed a generalization of our previous work, and we suggest that this more general approach makes the role of our so-called interaction postulates more explicit.

2 Preliminaries
We introduce some terminology and formal machinery that is commonly used for reasoning about action effects [GL98]. We are interested in action domains that can be described by a set of fluent symbols \( F \) and a set of action symbols \( A \). Informally, fluent symbols represent properties of the world that may change in response to the execution of the actions in \( A \). Formally, the effects of actions are given by transition systems.

**Definition 1** A transition system is a pair \( (S, R) \) where \( S \subseteq 2^F \) and \( R \subseteq S \times A \times S \).

A transition system is simply a directed graph where the nodes represent states and the edges are labeled with action symbols. In this paper, we assume that every action is always executable, so we restrict attention to transition systems where every state has an outgoing edge for each action symbol. We also restrict attention to actions with deterministic effects.

We define a belief state to be a set of interpretations over \( F \), informally the set of interpretations that an agent considers possible. An observation is also a set of interpretations.
The observation $\alpha$ is interpreted to provide evidence that the actual world is in $\alpha$.

The process in which an agent changes their beliefs in response to a predicted change in the state of the world is called belief update. One of the standard approaches to belief update is given in [KM92], where a set of formulas is updated by another formula. By contrast, we define belief update operators that map a belief state and an action to a new belief state. In particular, a transition system defines a belief update operator as follows.

Definition 2 Let $T = (S, R)$ be a transition system. The update function $\circ : 2^S \times A \rightarrow 2^S$ is given by $\alpha \circ A = \{ s | (s', A, s) \in R \text{ for some } s' \in \alpha \}$.

The process in which an agent changes their beliefs in response to new information about a static world is called belief revision, and one of the standard approaches is the AGM approach [AGM85]. Again, we diverge slightly from the standard approach in that we do not deal with formulas; instead we think of revision as an operation in which a belief state and an observation are mapped to a new belief state.

3 Motivating Example

We briefly introduce a simple, commonsense example in which an agent needs to compare the plausibility of certain actions with the plausibility of observations. We will return to this example periodically as we introduce the formal machinery.

We consider a simple action domain involving four agents: Bob, Alice, Eve, and Trent. Bob places a chocolate chip cookie on his desk and then leaves the room; he believes that no one is likely to eat his cookie while he is gone. At time 1, Bob knows that Alice is at his desk. At time 2, Bob knows that Eve is at his desk. At time 3, Trent comes and tells Bob that a bite has been taken from the cookie on his desk.

Given the preceding information, Bob can draw three reasonable conclusions: Alice bit the cookie, Eve bit the cookie, or Trent gave him poor information. If Bob has no additional information about the world, then each conclusion is equally plausible. However, we suppose that Bob does have some additional information. In particular, suppose that Alice is a close friend of Bob and they have shared cookies in the past. Moreover, suppose that Bob believes that Trent is always honest. Bob’s additional information about Alice and Trent provides a sufficient basis for determining which of the three possible conclusions is the most plausible.

Informally, at time 2, Bob believes that his cookie was unbiten at all earlier points in time. After Trent tells him the cookie is bitten, he must determine the most plausible world history consistent with this information. In this case, the most plausible solution is to conclude that Alice bit the cookie. Note that this conclusion requires Bob to alter his subjective view of the action history. There is a non-monotonic character to belief change in this context, because Bob may be forced to postulate and suppress actions over time in response to new observations. The ramifications of changing the action history are determined by the underlying transition system.

4 Plausibility Functions

At each point in time, an agent needs a plausibility ordering over all actions and all states. Moreover, in order to resolve inconsistency at different points in time, each of the plausibility orderings must be comparable. One natural way to create mutually comparable orderings is by assigning quantitative plausibility values to every action and state at every point in time. Towards this end, we define plausibility functions.

Definition 3 Let $X$ be a non-empty set. A plausibility function over $X$ is a function $r : X \rightarrow N$.

If $r$ is a plausibility function and $r(x) \leq r(y)$, then we say that $x$ is at least as plausible as $y$. We remark that we will typically be interested in plausibility functions over finite sets, where there is always a non-empty set of maximally plausible elements.

Plausibility functions are inspired by Spohn’s ordinal conditional functions [Spo88], but there are some important differences. First, we allow plausibility functions over an arbitrary set $X$, rather than restricting attention to propositional interpretations. This allows us to treat partially observable actions in the same manner that we treat observations. Another important difference is that ordinal conditional functions must always assign rank 0 to a non-empty subset of elements of the domain. Plausibility functions are not restricted in this manner; the minimal rank for a given plausibility function may be greater than 0. This distinction is based on our underlying intuition that some observations provide more reliable information than others.

In order to illustrate the application of plausibility functions, we continue our simple example.

Example (cont’d) We describe how the cookie problem can be represented with plausibility functions.

Let $F = \{ BiteTaken \}$ and let $A = \{ BiteAlice, BiteEve \}$. Both actions have the same effect, namely they both make the fluent BiteTaken become true. We represent the problem with 3 plausibility functions: $a_1$, $a_2$, and $a_3$.

1. $a_1$ is a plausibility function over actions at time 1
2. $a_2$ is a plausibility function over actions at time 2
3. $a_3$ is a plausibility function over observations at time 3

Informally, each function should obtain a minimum value at the event that Bob considers the most plausible at the given point in time. Since Bob initially believes that no one will eat his cookie, both $a_1$ and $a_2$ should obtain a minimum value at the null action $\lambda$. The observation that the cookie has been bitten at time 3 is represented by defining $a_3$ with a minimum at the set of worlds where the cookie has a bite out of it. The additional soft constraints are used to determine the magnitude of the values for each event. Define $a_1$ and $a_2$ by the values in the following table.

<table>
<thead>
<tr>
<th>$\lambda$</th>
<th>BiteAlice</th>
<th>BiteEve</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_1$</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>$a_2$</td>
<td>0</td>
<td>10</td>
</tr>
</tbody>
</table>
The fact that Alice is more likely to bite the cookie is represented by assigning a low plausibility value to BiteAlice at time 1. Define $o_3$ as follows.

<table>
<thead>
<tr>
<th>$o_3$</th>
<th>BiteAlice</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
</tr>
</tbody>
</table>

Hence, the observation \{BiteTaken\} is assigned the minimum plausibility value, and the only alternative observation is assigned a very high plausibility value. This reflects the fact that Trent’s report is understood to supersede the assumption that Alice and Eve do not bite the cookie.

5 Graded World Views

Graded world views are a formal tool for determining a maximally plausible world history, given an alternating sequence of ontic actions and epistemic actions. Intuitively, a graded world view simply consists of a sequence of plausibility functions. In the general case, we need two plausibility functions at each point in time. One function assigns a plausibility value to every action symbol, and the other function assigns a plausibility value to every state. The following definitions extend the observation trajectories and action trajectories of [HD05].

Definition 4 A graded observation trajectory of length $n$ is an $n$-tuple of plausibility functions over $2^F$.

Definition 5 A graded action trajectory of length $n$ is an $n$-tuple of plausibility functions over $A$.

Using graded trajectories, we get the following notion of a graded world view.

Definition 6 A graded world view is a pair $\langle ACT, OBS \rangle$ where $ACT$ is a graded action trajectory and $OBS$ is a graded observation trajectory of the same length.

Informally, at each point in time, an action is performed and it is followed by an observation. At time $i$, the most plausible action is given by the $i^{th}$ plausibility function in $ACT$ and the most plausible observation is given by the $i^{th}$ plausibility function in $OBS$.

We remark briefly on the intuition behind graded action trajectories. The plausibility value assigned to $A$ represents the plausibility that $A$ is executed at a given instant. Hence, the lowest plausibility values will be assigned to actions that an agent actually performs. The highest values will be assigned to exogenous actions that an agent believes are unlikely to occur. In this paper, we do not explicitly consider failed actions. Instead, we simply note that failed actions can be added to our formalism by allowing non-deterministic actions and attaching a plausibility value to possible effects, as in [Bou95]. We leave such an extension for future work.

Implicitly, the initial belief state in every graded world view is $2^F$. However, if the initial plausibility function in $ACT$ assigns a small minimum value to the null action $\lambda$, then one can think of the initial element of $OBS$ as the initial belief state. In this manner, the plausibility of the initial belief state is treated in exactly the same manner as the plausibility of any subsequent observation.

We formally define the notion of a history.

Definition 7 Let $T = \langle S, R \rangle$ be a transition system. A history over $T$ is a tuple $\langle w_0, A_1, \ldots, A_n, w_n \rangle$ where for each $i$:
1. $w_i \in S$,
2. $A_i \in A$,
3. $\langle w_i, A_i, w_{i+1} \rangle \in R$.

A history is simply an alternating sequence of interpretations and actions that represents a possible evolution of the world. Let $HIST_n$ denote the set of histories involving $n$ actions.

Given a graded world view, the main computational task for an agent is to determine the most plausible histories. This is similar to the process of belief extrapolation with mixed scenarios [DdSCL02], with two main differences. First, in belief extrapolation, there is a single plausibility ordering over histories rather than $2^n$ orderings over actions and states. Second, belief extrapolation operators are intended for action domains in which individual fluents may change values in an arbitrary manner. In our framework, every change must be caused by some action defined by the underlying transition system.

The plausibility of a history with respect to a graded world view is calculated by summing the plausibility at each instant.

Definition 8 Let $\langle ACT, OBS \rangle$ be a graded world view. The plausibility of a history $h = \langle w_0, A_1, \ldots, A_n, w_n \rangle$ with respect to $\langle ACT, OBS \rangle$ is the sum

$$plaus(h) = \sum_{i=1}^{n} ACT_i(A_i) + OBS_i(w_i).$$

It is useful to introduce an operator that maps graded world views to the set of histories that are assigned the minimum sum of plausibility values.

Definition 9 Let $WV$ denote the set of graded world views of length $n$ for a fixed action signature. Define $\Phi : WV \rightarrow 2^{HIST_n}$ as follows:

$$\Phi(\langle ACT, OBS \rangle) = \{ h \mid \text{plaus}(h) \leq \text{plaus}(g) \text{ for all } g \in HIST_n \}.$$ We revisit the earlier example with this new notation.

Example (cont’d) In order to give a complete representation of the problem, we need to define a graded world view. We define a graded action trajectory and a graded observation trajectory by extending the tables given previously.

<table>
<thead>
<tr>
<th>$\lambda$</th>
<th>BiteAlice</th>
<th>BiteEve</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_0$</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>$a_1$</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>$a_2$</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>$a_3$</td>
<td>0</td>
<td>10</td>
</tr>
</tbody>
</table>

Note that we now have plausibility functions at time 0 and at time 3. At each of these times, the null action is given the minimum plausibility to reflect that no action occurs. We need to add time 0 in order to restrict the initial belief state to $\emptyset$. Strictly speaking, we do not need to add time 3, we
simply add it to remain consistent with the timeline in the initial
problem description.

We are interested in finding $\Phi(\langle a_0, \ldots, a_3 \rangle, \langle o_0, \ldots, o_3 \rangle)$. By inspection, we find that the minimum plausibility is obtained by the following history:

$$ h = (0, \lambda, 0, \text{BiteAlice, BiteTaken}, \lambda, \text{BiteTaken, } \lambda, \text{BiteTaken}). $$

This history represents the sequence of events in which Alice bites the cookie at time 1. Intuitively, this is the correct solution: given the choice between Alice and Eve, Bob believes that Alice is the one who is more likely to help herself to the cookie.

### 6 Basic Properties

#### 6.1 Pointwise Dominance

Suppose that the underlying set $F$ of fluent symbols and the underlying set $A$ of action symbols are both finite. Let $W = \langle ACT, OBS \rangle$ be a graded world view with

$$ ACT = \langle ACT_1, \ldots, ACT_n \rangle $$

and

$$ OBS = \langle OBS_1, \ldots, OBS_n \rangle. $$

The simplest way to find a plausible world history is to simply take the most plausible action and most plausible worlds at each point in time. The following definition makes this notion more precise.

**Definition 10** Let $h = \langle w_0, A_1, \ldots, A_n, w_n \rangle$. We say $h$ is a pointwise minimum for $\langle ACT, OBS \rangle$ if, for all $i$,

1. for all $A \in A$, $ACT_i(A_i) \leq ACT_i(A)$, and
2. for all $w \in 2^F$, $OBS_i(w_i) \leq OBS_i(w)$.

Note that histories are restricted in that each world must be the outcome of the preceding action. As such, it is possible that a graded world view will have no pointwise minimum. However, if there are any pointwise minima, then clearly they will be the most plausible histories. We state this simple fact more formally.

**Proposition 1** Let $W = \langle ACT, OBS \rangle$ be a graded world view and let $M$ be the set of pointwise minima for $W$. If $M \neq \emptyset$, then $\Phi(W) = M$.

This observation suggests that checking for pointwise minima may be a good first step in the search for plausible world histories. Finding pointwise minima is not easy in the general case.

**Proposition 2** For a fixed graded world view $W$, determining if $W$ has a pointwise minimum is NP-complete.

### 6.2 Equivalence

Clearly it is possible for two distinct graded world views to have the same set of minimally ranked world histories. In fact, it is possible for two distinct graded world views to induce the same preference ordering over histories. In this section, we define a natural equivalence relation over graded world views with an eye towards categorical representations. We start by defining a relation on plausibility functions.

**Definition 11** Let $P_1$ and $P_2$ be plausibility functions over a set $X$. We say that $P_1 \cong P_2$ if, for every $x, y \in X$,

$$ P_1(x) - P_1(y) = P_2(x) - P_2(y). $$

It is clear that $\cong$ is an equivalence relation. It is also clear that this relation can be extended to graded world views.

**Definition 12** Let $WV_1$ and $WV_2$ be graded world views over histories for a fixed action signature. We say that $WV_1 \cong WV_2$ if, for every pair of histories $g$ and $h$,

$$ \text{plaus}_1(g) - \text{plaus}_1(h) = \text{plaus}_2(g) - \text{plaus}_2(h). $$

Let $P_1$ be a plausibility function. We say that $P_2$ is obtained from $P_1$ by a translation if there is some $n$ such that, for all $x$, $P_1(x) = P_2(x) + n$. It is easy to see that, whenever $P_1 \cong P_2$, it must be the case that $P_2$ is obtained by a translation on $P_1$. If a graded world view $WV_2$ is obtained from $WV_1$ by uniformly translating every component, then clearly $WV_1 \cong WV_2$. However, it is straightforward to construct equivalent graded world views that are not obtained by translations.

### 7 Comparison with Related Formalisms

#### 7.1 Representing Belief States

We introduce some notation that allows belief states to be represented by plausibility functions. If $K$ is a set of interpretations and $c$ is an integer, let $K \uparrow c$ denote function defined as follows:

$$ K \uparrow c \left( w \right) = \begin{cases} 0 & \text{if } w \in K \\ c & \text{otherwise} \end{cases} $$

If $c$ is a positive integer, then $K \uparrow c$ denotes a plausibility function in which the elements of $K$ are the most plausible, and everything else is equally implausible. Plausibility functions of the form $K \uparrow c$ will be called simple.

If $c < 0$, then $K \uparrow -c$ does not actually define a plausibility function. However, allowing negative values leads to a simple symmetry in our notation. In the following proposition, $\overline{K}$ denotes the complement of $K$.

**Proposition 3** For any belief state $K$ and positive integer $c$

$$ K \uparrow c \cong \overline{K} \uparrow -c. $$

Note that translating $\overline{K} \uparrow -c$ by an integer greater than $c$ gives another equivalent plausibility function. However, this equivalence does not mean that $K \uparrow c$ is interchangeable with translations of $\overline{K} \uparrow -c$ in a given world view. The magnitude of the largest plausibility value is different, which can be significant when determining minimal sums.

Suppose that

$$ ACT = \langle ACT_1, \ldots, ACT_n \rangle $$

<table>
<thead>
<tr>
<th>$i$</th>
<th>0</th>
<th>{BiteTaken}</th>
</tr>
</thead>
<tbody>
<tr>
<td>$o_0$</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>$o_1$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$o_2$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$o_3$</td>
<td>10</td>
<td>0</td>
</tr>
</tbody>
</table>
where each $ACT_i$ and $OBS_i$ is simple, with maximum plausibility $c$. Hence, we essentially have belief states with no plausibility ordering. In this case, it is easy to show that $\langle w_0, A_1, \ldots, A_n, w_n \rangle \in \Phi(\langle ACT, OBS \rangle)$ if and only if $|\{A_i | A_i \in ACT_1\}| + |\{w_i | w_i \in OBS_i\}|$ is maximal among all histories. In other words, the most plausible histories are those that agree with $\langle ACT, OBS \rangle$ at a maximal number of components.

The case in which there is no plausibility ordering is not very interesting from the perspective of belief change. However, it is easy to see that AGM belief revision operators can also be represented. In particular, let $r$ be a plausibility function over $X$ with minimum value $\min_x$. For any $n$, let $r[n]$ denote the set of complete, consistent theories that are satisfied by some $I$ with $r(I) \leq n$.

**Proposition 4** *The collection $R = \{r[n] | n \geq \min_x\}$ is a system of spheres centered on $r[\min_x]$.*

Now, by applying well-known results of Grove [Gro88], it is easy to construct a graded world view of length 2 corresponding to any AGM revision operator. In particular, if only null actions are permitted and the second observation is simple, then we essentially have AGM revision. This relationship is not surprising, since plausibility functions clearly induce an ordering over the the set of possible worlds.

We remark that graded world views bear a resemblance to the generalized belief change framework proposed by Liberatore and Schaerf [LS00]. However, there are some important distinctions. The Liberatore-Schaerf approach associates a “penalty” with state change, which is minimized when determining plausible models. As such, it is difficult to represent problems where non-null actions are strictly more plausible than null actions. By contrast, graded world views have no implicit preference for null actions. Moreover, since we define actions with respect to a transition system, graded world views are more suitable for the representation of actions with conditional effects.

### 7.2 Representing Belief Evolution Operators

Belief evolution operators have been introduced to represent sequences of alternating updates and revisions. We briefly sketch the approach, and refer the reader to [HD05] for the details. Let $A = A_1, \ldots, A_n$ be a sequence of action symbols and let $O = O_1, \ldots, O_n$ be a sequence of observations. Given an initial belief state $K$, the evolution operator $\circ$ roughly corresponds to the following iterated belief change:

$$K \circ \langle A, O \rangle = K \circ A_1 \circ O_1 \circ \cdots \circ A_n \circ O_n.$$  

Simply performing the updates and revisions in succession gives unintuitive results. As a result, we have specified a number of so-called interaction postulates, and the definition of $\circ$ is constructed in a manner that assures the postulates must hold.

There are two underlying assumptions in belief evolution that are not required in a representation by graded world views.

1. The plausibility of an observation is determined by recency.
2. The action history is assumed to be correct.

Both of these assumptions can be represented in a graded world view by setting up the plausibility functions appropriately. In particular, for each $i$, we define $OBS_i = O_i \uparrow 2^i$.

By incrementing the plausibility of false observations exponentially, we can assure that recent observations will be given greater credence. The fact that action histories must be correct is represented by setting $ACT_i = A_i \uparrow 2^{n+1}$ for every $i$. Recall that $\circ$ is defined with respect to an update operator $\circ$ and a revision operator $\ast$. As a result, in order to represent $\circ$ in a graded world view, we also need to encode the plausibility ordering implicit in $\ast$. Omitting the details of the construction, we get the following result.

**Proposition 5** *Let $\circ$ be a belief evolution operator obtained from $\circ$ and $\ast$. There is a graded world view $W_{ev}$ such that, if $K \circ \langle O, A \rangle = \langle K_0, \ldots, K_n \rangle$, then $\langle w_0, A_1, \ldots, A_n, w_n \rangle \in \Phi(W_{ev})$ if and only if for each $i$, $w_i \in K_i$.*

Proposition 5 demonstrates that graded world views can represent any belief evolution operator. So, the interaction postulates for belief evolution will be satisfied by a graded world view whenever the plausibility functions are defined as above. Hence, from the perspective of graded world views, the role of the interaction postulates is essentially to restrict the admissible plausibility functions.

### 7.3 Comparison With Belief Extrapolation

As noted earlier, the motivation underlying our formalism is similar to the motivation underlying belief extrapolation operators. In this section, we demonstrate some expressive differences between the two formalisms. In the interest of space, we refer the reader to [DdSCL02] for the required background on belief extrapolation. We remark that we will abuse notation by equating the trajectories of belief evolution with histories.

We need to give some simple terminology used in belief extrapolation. A scenario is a tuple of formulas. For $t$ less than the length of $\Sigma$, let $\Sigma(t)$ denote the $t^{th}$ formula in the scenario $\Sigma$. We say that a history $\langle w_0, \ldots, w_n \rangle$ satisfies a scenario $\Sigma$ if $w_i \models \Sigma_i$ for each $i \leq n$. The set of histories satisfying $\Sigma$ is denoted by $\text{Traj}(\Sigma)$.

We are interested in determining if all belief extrapolation operators can be represented by graded world views. First, we need to formalize the problem more precisely.

**Definition 13** *Let $\uparrow$ be a belief extrapolation operator. We say that $\uparrow$ is representable if, for every scenario $\Sigma$ of length $n$, there is a graded world view $\langle OBS, ACT \rangle$ of length $n$ such that $\text{Traj}(\Sigma \uparrow) = \Phi(\langle OBS, ACT \rangle)$.*
If \( \mathcal{H} \) is representable, then the behavior of \( \downarrow \) can be simulated with graded world views.

The following proposition indicates that belief extrapolation operators have an expressive advantage.

**Proposition 6** There is a belief extrapolation operator \( \downarrow \) that is not representable.

We remark that the proof of Proposition 6 is constructive and it demonstrates that there is a simple, concrete, inertial extrapolation operator that is not representable. Intuitively, the distinction is that a belief extrapolation operator is based on an ordering over trajectories rather than several orderings over actions and states.

There is also a sense in which graded world views are more expressive than belief extrapolation operators. In particular, they provide a mechanism for handling unreliable observations. One of the main assumptions underlying belief extrapolation is that every observation should be incorporated in the new scenario. By contrast, we are interested in applications where some observations may be incorrect. For example, the cookie problem can easily be modified by assigning a very high value to the observation reported by Trent. This would reflect the fact that Trent is not a reliable source of information, and it would lead to plausible histories in which that observation is simply ignored. In particular, if Trent is not reliable, then the most plausible history is the one in which nobody bites the cookie.

### 8 Discussion

We have introduced a formalism for reasoning about sequences of actions and observations. The formalism uses ranking functions at each instant to determine the most plausible action or observation, and determines the most plausible histories by summing over all instants. The formalism provably subsumes belief revision and belief evolution. The relationship with belief extrapolation is more subtle, with each formalism having expressive advantages and disadvantages.

The generality of graded world views can be seen in comparison with iterated revision. Papini illustrates two different approaches to iterated revision, one which gives greater credence to recent information and one which gives greater credence to old information [Pap01]. In the same manner, given a sequence of actions and observations \( A_1, O_1, \ldots, A_n, O_n \), any ordering of the actions and observations may be used to resolve conflicts. Clearly, any such ordering can be represented by a graded world view by assigning maximal plausibility values that increase as powers of 2. Moreover, repetitions of observations and actions can be used to make democratic decisions based on, for example, the number of times that a given observation occurs. For this kind of reasoning, the full generality of arbitrary plausibility functions is useful.

We consider graded world views to be the most general possible extension of belief evolution. The interaction postulates of belief evolution essentially formalize the fact that action histories are infallible. Hence, from the perspective of graded world views, the postulates serve to restrict the admissible plausibility functions. Alternative postulates could be proposed to give different restrictions for a different class of problems. In the future, we would like to look for a most general set of postulates for which we could prove a representation result for graded world views.

### References

Improving Recovery for Belief Bases

Frances L. Johnson and Stuart C. Shapiro
Department of Computer Science and Engineering; SNePS Research Group
Center for Multisource Information Fusion; Center for Cognitive Science
University at Buffalo, The State University of New York
201 Bell Hall, Buffalo, NY 14260-2000, USA
{fjl | shapiro}@cse.buffalo.edu

Abstract

The Recovery postulate for contraction says that any beliefs lost due to the contraction of some belief $p$ should return if $p$ is immediately re-asserted. Recovery holds for logically closed sets of beliefs, but it does not hold for belief bases (sets of beliefs that are not logically closed). This paper discusses the Recovery aspect of the belief base optimizing operation of reconsideration (which performs hindsight belief change) and compares it to the adherence to Recovery of traditional base and belief liberation contraction operations. We also discuss the similarities and differences between the belief base manipulations for belief liberation vs. those for reconsideration — because both approaches support the concept that removing a belief from a base might allow some previously removed beliefs to return.

1 Introduction

1.1 Motivation

Any agent reasoning from a set of beliefs must be able to perform basic belief change operations, including expansion, contraction and consolidation. Briefly, expansion is adding a belief to a set without concern for any inconsistencies it might raise; contraction of a set by a belief results in a set that does not entail (cannot derive) that belief — it is the removal of retraction\(^1\) of that belief; and consolidation of a set of beliefs produces a consistent subset of the original set. See Section 1.3 for more details.

The Recovery postulate for contraction\citep{AlchourronEtAl1985} says that a logically closed set $K$ is contained in the set that results from contraction of $K$ by a belief $p$ followed by union with \{p\} and deductive closure.

A belief base, according to the foundations approach (see discussion in \citep{Gardenfors1992} and \citep{Hansson1999}), is a finite set of core or base beliefs (also called hypotheses in \citep{MartinsShapiro1988}) that have independent standing and are treated differently from derived beliefs. A base is, typically, not logically closed.

Contraction of a belief base $B$ by some belief $p$ is successful if $p$ is absent from the resulting base and its logical closure. Although Recovery does not hold in general for belief base contraction (due to the lack of closure before contraction), it does hold in some specific cases.

The research defining belief liberation \citep{BoothEtAl2003} and reconsideration \citep{JohnsonShapiro2005} supports the concept that removing a belief from a base might allow some previously removed beliefs to return.\(^2\) Because of this common view, we could not present the Recovery aspect of reconsideration without first examining the similarities and differences between these works (see Section 3.4). We then use belief liberation terminology and reconsideration in separate formulations of the Recovery postulate; and we discuss and compare the specific cases that do (or do not) adhere to these formulations as well as to the traditional Recovery postulate for belief bases.

1.2 Notation and Terminology

For this paper, we use a propositional language, $\mathcal{L}$, which is closed under the truth functional operators $\neg, \lor, \land, \rightarrow$, and $\leftrightarrow$. Formulas of the language $\mathcal{L}$ are denoted by lowercase letters ($p, q, r, \ldots$). Sets and sequences are denoted by uppercase letters ($A, B, C, \ldots$). If set $A$ derives $p$, it is denoted as $A \vdash p$. $C_n$ is defined by $C_n(A) = \{p \mid A \vdash p\}$, and $C_n(A)$ is called the closure of $A$. A belief base $B$ is consistent iff $B \not\vdash \bot$, where $\bot$ denotes logical contradiction. A belief set (a.k.a theory), $K$, is a logically closed set of beliefs (i.e. $K = C_n(K)$) \citep{AlchourronEtAl1985}. We will use $B$ for a belief base and $K$ for a belief set.

Given a finite belief base, $B$, the set of $p$-kernels of $B$ is the set of $A \mid A \subseteq B, A \vdash p$ and ($\forall A' \subseteq A. A' \not\vdash p$) \citep{Hansson1994}. The $p$-kernels known to derive $p$ are called $p$'s origin sets in \citep{MartinsShapiro1988}.

A nogood in the ATMS literature \citep{deKleer1986, ForbusKleer1993} is a minimally inconsistent set $S$ s.t. $S \vdash \bot$, but for all $S' \subseteq S$, $S' \not\vdash \bot$.

\(^1\)The term retraction is also used in the literature to define a specific subclass of contraction. In this paper, we use the term retraction as a synonym for removal.

\(^2\)This is very different from the recovery of retracted beliefs during either saturated kernel contractions \citep{Hansson1994} or the second part of Hybrid Adjustment \citep{WilliamsSims2000}.
1.3 Background
This section briefly reviews the traditional belief change operations of expansion and contraction of a logically closed belief set \( K \) [Alchourrón et al., 1985] and expansion, kernel contraction and kernel consolidation of a finite belief base \( B \). [Hansson, 1994; 1999].

Expansion
\( K + p \) (the expansion of the belief set \( K \) by the belief \( p \)) is defined as \( Cn(K \cup \{p\}) \).
\( B + p \) (the expansion of the belief base \( B \) by the belief \( p \)) is defined as \( B \cup \{p\} \).

Kernel Contraction
The contraction of a base \( B \) [or set \( K \)] by a belief \( p \) is written as \( B \sim p \) [\( K \sim p \)].
For this paper, \( B \sim p \) is the kernel contraction [Hansson, 1994] of the belief base \( B \) by \( p \) (retraction of \( p \) from \( B \) and, although constrained by several postulates, is basically the base resulting from the removal of at least one element from each \( p \)-kernel in \( B \) — unless \( p \in Cn(\emptyset) \), in which case \( B \sim p = B \). Given a belief base \( B \), if \( K \) is the belief space for \( B \) (\( K = Cn(B) \)), then \( K \sim p = Cn(B \sim p) \).

Kernel Consolidation
Consolidation (the removal of any inconsistency) is defined for belief bases only. Any inconsistent belief set is the set of all beliefs (due to closure in classical logic), making inconsistency removal a non-issue — set operations focus on preventing inconsistencies from occurring.
\( B! \) (the kernel consolidation of \( B \)) is the removal of at least one element from each nogood in \( B \) s.t. \( B! \subseteq B \) and \( B! \not\perp \). This means that \( B! =_{df} B \sim \perp \).

1.4 Recovery
Recovery does not hold for kernel contraction when elements of a \( p \)-kernel in \( B \) are retracted during the retraction of \( p \), but are not returned as a result of the expansion by \( p \) followed by deductive closure. Not only do these base beliefs remain retracted, but derived beliefs that depend on them are also not recovered.

Example Given the base \( B = \{s, d, s \rightarrow q\} \), \( B \sim s \vee d = \{s \rightarrow q\} \), and \( (B \sim s \vee d) + s \vee d = \{s \vee d, s \rightarrow q\} \). Not only do we not recover \( s \) or \( d \) as individual beliefs, but the derived belief \( q \) is also not recovered.
We feel the assertion of \( s \vee d \) means that its earlier retraction was, in hindsight, not valid for this current state, so all effects of that retraction should be undone. There are various criticisms of Recovery in the literature (see [Hansson, 1999] and [Williams, 1994] for discussions and further references). We address these criticisms in [Johnson, 2005], but state our general argument below.
Our defense of Recovery is predicated on the fact that the recovered beliefs were at one time in the base as base beliefs. The recovery of those previously retracted base beliefs should occur whenever the reason that caused them to be removed is, itself, removed (or invalidated). In this case, the previously retracted beliefs should be returned to the base, specifically because they were base beliefs and the reason for disbelieving them no longer exists.

2 Reconsideration
2.1 Assuming a Linear Preference Ordering
In defining reconsideration, [Johnson and Shapiro, 2005] make the assumption that there is a recency-independent, linear preference ordering (\( \succeq \)) over all base beliefs. Thus, any base can be represented as a unique sequence of beliefs in order of descending preference: \( B = p_1, p_2, \ldots, p_n \), where \( p_i \succeq p_{i+1} \), \( 1 \leq i < n \). Note: \( p_i \succ p_j \) means that \( p_i \) is strictly preferred over \( p_j \) (is stronger than \( p_j \)) and is true iff \( p_i \succeq p_j \) and \( p_j \not\succeq p_i \).

2.2 The Knowledge State for Reconsideration
The knowledge state used to formalize reconsideration [Johnson and Shapiro, 2005] is a tuple with three elements. Starting with \( B_0 = \emptyset \), \( B_n \) is the belief base that results from a series of expansion and consolidation operations on \( B_0 \) (and the subsequent resulting bases: \( B_1, B_2, B_3, \ldots \)). \( B^0 = \bigcup_{0 \leq i \leq n} B_i \). \( X_n \) is the set of base beliefs removed (and currently dis-believed: \( B_n \cap X_n = \emptyset \)) from these bases during the course of the series of operations: \( X_n =_{df} B^0 \setminus B_n \).
The knowledge state is a triple of the form \( \langle B, B^0, \succeq \rangle \), where \( \succeq \) is the linear ordering of \( B^0 \), \( X = B^0 \setminus B \) and \( Cn(B, B^0, \succeq) = Cn(B) \). All triples are assumed to be in this form.
A numerical value for credibility of a base is calculated from the preference ordering of \( B^0 \) \( \succeq \) \( B^0 \setminus B \) and \( Cn(B, B^0, \succeq) = Cn(B) \). All triples are assumed to be in this form.
A numerical value for credibility of a base is calculated from the preference ordering of \( B^0 \) \( \succeq \) \( B^0 \setminus B \) and \( Cn(B, B^0, \succeq) = Cn(B) \). All triples are assumed to be in this form.
A numerical value for credibility of a base is calculated from the preference ordering of \( B^0 \) \( \succeq \) \( B^0 \setminus B \) and \( Cn(B, B^0, \succeq) = Cn(B) \). All triples are assumed to be in this form.
A numerical value for credibility of a base is calculated from the preference ordering of \( B^0 \) \( \succeq \) \( B^0 \setminus B \) and \( Cn(B, B^0, \succeq) = Cn(B) \). All triples are assumed to be in this form.
A numerical value for credibility of a base is calculated from the preference ordering of \( B^0 \) \( \succeq \) \( B^0 \setminus B \) and \( Cn(B, B^0, \succeq) = Cn(B) \). All triples are assumed to be in this form.
A numerical value for credibility of a base is calculated from the preference ordering of \( B^0 \) \( \succeq \) \( B^0 \setminus B \) and \( Cn(B, B^0, \succeq) = Cn(B) \). All triples are assumed to be in this form.
A numerical value for credibility of a base is calculated from the preference ordering of \( B^0 \) \( \succeq \) \( B^0 \setminus B \) and \( Cn(B, B^0, \succeq) = Cn(B) \). All triples are assumed to be in this form.

2.3 Optimal Base
Given a possibly inconsistent set of base beliefs, \( B^{\cup} = p_1, p_2, \ldots, p_n \), ordered by \( \succeq \), the base \( B \) is considered optimal with respect to \( B^{\cup} \) and \( \succeq \) if and only if \( B \subseteq B^{\cup} \) and \( \{\forall B' \subseteq B^{\cup} : B \succeq B'\} \succeq B' \). This favors retaining a single strong belief over multiple weaker beliefs.
As in [Johnson and Shapiro, 2005], an operation of contraction or consolidation produces the new base \( B' \) by using a global incision function \(^4\) that maximizes \( \text{Cred}(B', B^{\cup}, \succeq) \) w.r.t. the operation being performed. Note: maximizing \( \text{Cred}(B', B^{\cup}, \succeq) \) without concern for any specific operation would result in \( B' = B^{\cup} \).

Observation 2.1 The consolidation of a base \( B \) is the optimal subset of that particular base (w.r.t. \( B^{\cup} \) and \( \succeq \)) \( : B! \subseteq B \) and \( \{\forall B' \subseteq B : B! \succeq B'\} \succeq B' \).

2.4 Operations on a Knowledge State
The following are operations on the knowledge state \( B = \langle B, B^{\cup}, \succeq \rangle \).

\(^3\)Adding beliefs to a finite base by way of expansion followed by consolidation is a form of non-prioritized belief change called semi-revision [Hansson, 1997].

\(^4\)An incision function is the function that determines which beliefs should be removed during the operations of kernel contraction and kernel consolidation.

66
3.1 Basic Notation

In this section, we summarize \( \sigma \)-liberation [Booth et al., 2003] and compare it to reconsideration. Like reconsideration, liberation assumes a linear sequence of beliefs which is called \( \sigma = p_1, \ldots, p_n \). The sequence is ordered by recency, where \( p_1 \) is the most recent information, the agent has received (and has highest preference), and the set \( [\sigma] \) is the set of all the sentences appearing in \( \sigma \).

Since the ordering in this sequence is based on recency, for the remainder of this section, all comparisons between features of liberation and those of reconsideration are predicated on the assumption that both of their sequences are ordered by recency.²

3.2 A Belief Sequence Relative to \( K \)

In [Booth et al., 2003] the ordering of \( \sigma \) is used to form the maximal consistent subset of \([\sigma]\) iteratively by defining the following: (1) \( B_0(\sigma) = \emptyset \). (2) for each \( i = 0, 1, \ldots, n - 1 \): if \( B_i(\sigma) + p_{i+1} \vdash \bot \), then \( B_{i+1}(\sigma) = B_i(\sigma) + p_{i+1} \), otherwise \( B_{i+1}(\sigma) = B_i(\sigma) \). That is, each belief — from most recent to least — is added to the base only if it does not raise an inconsistency.

Definition 3.1 [Booth et al., 2003] Let \( K \) be a belief set and \( \sigma = p_1, \ldots, p_n \) a belief sequence. We say \( \sigma \) is a belief sequence relative to \( K \) iff \( K = Cn(B_n(\sigma)) \).

3.3 Removing a Belief \( q \) from \( K \)

In [Booth et al., 2003] the operation of removing the belief \( q \) is defined using the following: (1) \( B_0(\sigma, q) = \emptyset \). (2) for each \( i = 0, 1, \ldots, n - 1 \): if \( B_i(\sigma, q) + p_{i+1} \not\vdash q \), then \( B_{i+1}(\sigma, q) = B_i(\sigma, q) + p_{i+1} \), otherwise \( B_{i+1}(\sigma, q) = B_i(\sigma, q) \). Note that \( \sim B_n(\sigma) = B_n(\sigma, \bot) \) and \( B_n(\sigma, q) \) is the set-inclusion maximal amongst the subsets of \([\sigma]\) that do not imply \( q \)².

Given a belief sequence \( \sigma \) relative to \( K \), \( \sigma \) is used to define an operation \( \sim \) for \( K \) such that \( K \sim q \) represents the result of removing \( q \) from \( K \) [Booth et al., 2003]: \( K \sim q = Cn(B_n(\sigma, q)) \) if \( q \not\in Cn(\emptyset) \), otherwise \( K \sim q = K \).

Definition 3.2 [Booth et al., 2003] Let \( K \) be a belief set and \( \sim \) be an operator for \( K \). Then \( \sim \) is a \( \sigma \)-liberation operator (for \( K \)) iff \( \sim = \sim_\sigma \) for some belief sequence \( \sigma \) relative to \( K \).

Example [Booth et al., 2003] Suppose \( K = Cn(p \land q) \) and let \( \sigma = p \land q, \neg p \land \neg q \) be the belief sequence relative to \( K \) — where \( \neg p \land \neg q \) was originally blocked from inclusion in \( B_3(\sigma) \) by the inclusion of the more recent (and more preferred) belief \( p \). Suppose we wish to remove \( p \). We first compute \( B_3(\sigma, p) \). We have \( B_3(\sigma, p) = \emptyset, B_1(\sigma, p) = \{ p \rightarrow q \} = B_2(\sigma, p) \), and \( B_3(\sigma, p) = \{ p \rightarrow q, \neg p \land \neg q \} \). Hence \( K \sim p = Cn(B_3(\sigma, p)) = Cn(\neg p \land \neg q) \) Note how, when determining \( B_2(\sigma, p) \), \( p \) is nullified, which leads to the reinstatement, or liberation, of \( \neg p \land \neg q \).

3.4 Comments On Liberation

Key Difference from Reconsideration

Reconsideration was intended specifically to improve adherence to Recovery for belief base contraction. The research in

³We assume that if \( p \in B_3 \), the location of \( p \) in the sequence might change — i.e. its old ordering information is removed before adding \( \geq p \) and performing closure — but all other beliefs remain in their same relative order.

⁴We have reversed the ordering from that presented in [Booth et al., 2003] to avoid superficial differences when comparing their ordering with ours. We have adjusted the definitions accordingly.

⁵The differences between a recency-independent ordering and ordering by recency are discussed in Section 4.2.
belief liberation focuses on defining liberation operators for some belief set $K$ relative to some arbitrary $\sigma$. The focus is on $K$ and how it changes when a contraction is performed — whether there is any $\sigma$ that shows that a given contraction operation is an operation of $\sigma$-liberation. The authors do not advocate maintaining any one, specific $\sigma$. Although it is clearly stated that $\sigma$-liberation does not adhere to Recovery, the similarity between $\sigma$-liberation and reconsideration prompted us to compare them in detail.

### Similarities to Reconsideration

Assume $B^\cup = [[\sigma]]$ and is ordered by recency, and we refer to the belief set associated with $\sigma$ as $K_\sigma$. $B_n(\sigma)$ is the maximal consistent subset of $[[\sigma]]$ — i.e. $B_n(\sigma) = [[\sigma]] \setminus B^\cup$. Similarly, $B_n(\sigma, p)$ is the kernel contraction of $[[\sigma]]$ by $p$. In other words, $B_n(\sigma, p) = B^\cup \setminus p$. Thus, $K \sim_\sigma p = C_n(B^\cup \setminus p)$.

If $B = B^\cup \setminus B_n(\sigma)$, then we can define $\sigma_B$ to be a recency ordering of just the beliefs in $B_n(\sigma)$, and $K_\sigma = K_{\sigma_B}$. Now we can define contraction of an optimal knowledge state in terms of contraction for $\sigma$-liberation: $B \sim p = (K_\sigma \sim_\sigma p) \cap B$ and $C_n((B, B^\cup, \succ) \sim p) = K_\sigma \sim_\sigma p$.

Let us define $\sigma$-addition (adding a belief to $\sigma$) as follows: $\sigma + p$ is adding the belief $p$ to the sequence $\sigma = p_1, \ldots, p_n$ to produce the new sequence $\sigma_1 = p, p_1, \ldots, p_n$.

If $\sigma$ is the sequence for $B^\cup$, then the optimized addition of $p$ to any knowledge state for $B^\cup$ results in a base equivalent to the base for $p$ added to $\sigma$: Given $B^\cup \cup p = (B', B^\cup + p, \succ')$, then $B' = B_{n+1}(\sigma + p)$.

Likewise, $\sigma$-addition followed by recalculation of the belief set is equivalent to optimized-addition followed by closure: $K_{\sigma + p} = C_n(B^\cup + p)$.

### Cascading Belief Status Effects of Liberation

It is important to realize that there is a potential cascade of belief status changes (both liberations and retractions) as the belief set resulting from a $\sigma$-liberation operation of retracting a belief $p$ is determined; and these changes cannot be anticipated by looking at only the nogoods and kernels for $p$.

**Example** Let $\sigma = p \rightarrow q, p \neg \rightarrow q, r \rightarrow p \lor q, r \neg \rightarrow p$. Then, $B_n(\sigma) = \{p \rightarrow q, p, r \rightarrow p \lor q, r\}$. Note that $r \in K_\sigma$ and $\neg r \notin K_\sigma$. Therefore, $K_\sigma \sim_\sigma p = C_n\{p \rightarrow q, p \neg \rightarrow q, r \rightarrow p \lor q, r\}$.

Even though $r$ is not in a $p$-kernel in $[[\sigma]]$, $r \notin K_\sigma \sim_\sigma p$. Likewise, $\neg r$ is liberated even though $\neg \neg r \in N$ s.t. $N$ is a nogood in $[[\sigma]]$ and $\{\neg r, p\} \subseteq N$.

Reconsideration has an identical effect. If $B^\cup = \sigma$, and $B = B^\cup \setminus B_n(\sigma)$, then $(B, B^\cup, \succ) \cup \langle \neg p, \neg \succ p \rangle$, where $\succ p$ indicates $\neg p \succ p$, would result in the base $B_1 = \{\neg p, p \rightarrow q, p \neg \rightarrow q, r \rightarrow p \lor q, r\}$.

\footnote{Note: specifically not $B_n(\sigma, p) = B^\cup \setminus p$.}

\footnote{This is also the technique described in [Chopra et al., 2001] — though, again, we have reversed the order.}

\footnote{Our notation for the base associated with a $\sigma$-addition is not inconsistent with the notation of [Booth et al., 2003] for the base associated with a $\sigma$-liberation operation. Addition changes the sequence, so we are determining the base for the new sequence $(\sigma + p)$: $B(\sigma + p)$. The operation of $\sigma$-liberation changes the base used to determine the belief set (from $B(\sigma)$ to $B(\sigma, p)$), but the sequence $\sigma$ remains unchanged.}

### 4 Improving Recovery for Belief Bases

#### 4.1 Comparing Recovery-like Formulations

**Table 1** shows the cases where different Recovery formulations hold — and where they do not hold. There is a column for each formulation and a row for each case. The traditional Recovery postulate for bases $(C_n(B) \subseteq C_n((B \sim p) + p))$ is shown in column (TR). In column (LR), the recovery postulate for $\sigma$-liberation retraction followed by expansion (Liberation-recovery, LR, our term) is: $K \subseteq ((K \sim_\sigma p) + p)$.

In column (OR), the recovery-like formulation for kernel contraction followed by optimized-addition is: $K \subseteq C_n((B \sim p) + p)$ (called Optimized-recovery, OR). We also claim $B \subseteq ((B \sim p) + p)$, which is more strict than Recovery: base beliefs are recovered in the base, itself, not just its closure. For column (OR-i), we assume that the ordering for $B^\cup$ and $B_n^\cup$ is recency. For column (OR-ii), we assume that the ordering is not recency-based, $p \in B^\cup$ (not applicable for Case 3), and optimized-addition returns $p$ to the sequence in its original place (i.e. $\geq \succ \sigma$). Note that (OR) is not a true Recovery axiom for some contraction operation; because it can be rewritten as $K \subseteq C_n((B \sim p) + p) \setminus K$, where reconsideration is performed after the expansion but before the closure to form the new belief space.

YES means the formulation always holds for that given case; NO means it does not always hold; NA means the given case is not possible for that column’s conditions. The second entry indicates whether the base/set is optimal w.r.t. $B_n^\cup = B^\cup + p = (\sigma + p)$ and its linear order. If not optimal, then a designation for consistency is indicated. Recall that optimality requires consistency.

**Theorem 4.1** Expansion of an optimal knowledge state by a belief that is consistent with the base (and is not being relocated to a lower position in the ordering) results in a new and optimal knowledge state: Given $B = (B, B^\cup, \succeq)$, where $B = B^\cup \setminus X$ and $X = B^\cup \setminus B$, then $(\forall p s.t. B + p \not\subseteq \bot)$: $B + (p, \not\subseteq p) = \{B + p, B^\cup + p, \not\subseteq p\} = \{B + p, B^\cup + p, \not\subseteq p\}$. (Provided: if $p = p_i \in B^\cup = p_j \in (B^\cup + p)$, then $j \leq i$; otherwise $\not\subseteq p$ might remove $p$.)

**Proof:** $B = B^\cup, (\forall x \in X) : B + x \not\subseteq \bot$ (and $\not\subseteq B' \subseteq B$ s.t. both $(B' \setminus B) + x \not\subseteq \bot$ and $(\forall b \in B') x \not\subseteq b$. Therefore, since $B + p \not\subseteq \bot$, then $\not\subseteq B'' \subseteq (B^\cup + p) : (B + p) \not\subseteq B''$. □

**Case 1** In this simple case, $\{p\}$ is the sole $p$-kernel in $B$. For all formulations, $p$ is removed then returned to the base, therefore all cases hold.

**Case 2** Since there are $p$-kernels in $B$ that consist of beliefs other than $p$, beliefs other than $p$ must be retracted during contraction of $p$. For (TR), if $B = (p \not\subseteq q)$, then $B \sim p = \emptyset$ and $(B \sim p) + p = \{p\}$. Therefore, $K \not\subseteq C_n((B \sim p) + p)$, and (TR) does not hold. For (LR), if $p = p \not\subseteq q$, then $K \sim_\sigma p = \emptyset$ and $(K \sim_\sigma p) + p = C_n(\{p\})$. So, (LR) also does

---

\[68\]
Table 1: This table indicates whether each of three different Recovery formulations (TR, LR and OR) holds in each of four different cases (which comprise all possible states of belief). $K = Cn(B)$ and $p = \{p; \geq p\}$. YES means the formulation always holds for that given case; NO means it does not always hold; NA means the given case is not possible for that column’s conditions. See the text for a detailed description. Note: If requiring contraction for consistency maintenance only, a column for adherence to either $B \subseteq (B +p \text{ } \neg p) +_3 p$ (ordered by recency) or $K_{\sigma} \subseteq K_{(\sigma + p)p}$ would match (OR-i). ¹¹

not hold. For (OR), since $p \in Cn(B)$, then $B +p \not\perp \perp$. Thus $B_1 = B +p$ (from Theorem 4.1), so $B \subseteq B_1$, and (OR) holds.

**Case 3** Since $p \notin Cn(B)$ and $B +p \not\perp \perp$, we know $p \notin B^{(3)}$ — otherwise, $(B+p) \succ_{P} B$ and $B \neq B^{(3)}$! as it was defined. Column (OR-ii) has NA (for “Not Applicable”) as its entry, because (OR-ii) assumes that $p \notin B^{(3)}$. For the other columns, $B \sim p = B, K \sim_{\sigma} p = K = Cn(B)$, and $B \sim p = B$. Clearly, (TR) holds and (LR) holds. (OR-i) also holds (Theorem 4.1).

**Case 4** Because $p \notin Cn(B)$, $B \sim p = B$ and $K \sim_{\sigma} p = K = Cn(B)$. Since $B +p \perp \perp$ and both (TR) and (LR) produce inconsistent spaces, they both hold. For (OR), $B \sim p = B$. For (OR-i), the optimized-addition puts $p$ at the most preferred end of the new sequence (most recent), so $p \in B_1$ forcing weaker elements of $B$ to be retracted for consistency maintenance during reconsideration (recall $B +p \perp \perp$). Therefore (OR-i) does not hold. ¹¹ For (OR-ii), optimized-addition returns $p$ to the same place in the sequence that it held in $B^{(3)}$ (recall $B^{(3)}_1 = B^{(3)}$ and $\geq \equiv \geq_1$). Therefore, $B = B_1$ and (OR-ii) holds.

4.2 Discussion

When comparing the traditional base recovery adherence (in column TR) to optimized recovery adherence (shown in the OR columns), the latter produces improved adherence, because:

1. when the retraction of $p$ is truly “undone” (column OR-ii), $B$ is recovered in all applicable cases;
2. using a recency-based ordering (OR-i), $B$ is recovered in all cases where $p \in Cn(B)$;
3. if expansion by $p$ traditionally makes the final base inconsistent (TR, 4), although $B$ is not recovered, the final base is consistent and optimal (OR-i, 4).

Reconsideration eliminates the results of any preceding contraction, because $B^{(3)}$ is unaffected by contraction: $B \sim p)_3 = B^{(3)}$. Likewise, optimized-addition also eliminates the results of any preceding contraction: $\forall q : (B \sim q)_3 +_3 p = B +_3 p$.

If we consider contraction for consistency-maintenance only (assuming ordering by recency), the recovery-like formulation $B \subseteq (B +_3 p) +_3 p$ would have column entries identical to those in the column under (OR-i). Likewise, the entries in a column for $K_{\sigma} \subseteq K_{(\sigma + p)p}$ would also be identical to the entries for column (OR-i). ¹²

We also note that the improved Recovery aspect that reconsideration provides does not involve the addition of extra beliefs to the belief base. A belief base can “adhere” to Recovery if the contraction operation to remove $p$ also inserts $p \rightarrow q$ into the base, for every belief $q$ that is removed during that retraction of $p$. However, this deviates from our assumption of a foundations approach, where the base beliefs represent the base input information from which the system or agent should reason. Not only would this technique insert unfounded base beliefs, but the recovery of previously removed beliefs would only show up in the belief space; whereas reconsideration actually returns the removed beliefs to the belief base.

If the linear ordering is recency-independent and $\geq_1 \neq \geq$, then there are cases where Optimized-recovery does not hold even though the resulting base will still be optimal. For Case 1, if $p$ is re-inserted into the ordering at a weaker spot, it might be retracted during reconsideration if it is re-asserted in a position that is weaker than the conflicting elements of some pre-existing nogood and the incision function favors retracting $p$. This could also happen in case 2, unless the elements of some p-kernel are all high enough in the order to force the retraction of the beliefs conflicting with $p$. In Case 3 all Recovery formulations always hold. In Case 4, if $p$ is inserted into the final ordering at a strong enough position, it could

¹²The (OR-i) results for $B \subseteq (B +_3 p) +_3 p$ and $K_{\sigma} \subseteq K_{(\sigma + p)p}$ show adherence to (R3) in [Chopra et al., 2002].
survive the reconsideration step of optimized-addition — in which case, (OR) would not hold.

5 Conclusions and Future Work

From Section 3.4, we see that a system that can implement $\sigma$-liberation can also implement reconsideration and vice versa.

Kernel consolidation of a finite belief base has adherence results for Optimized-recovery (OR) that are preferred over the adherence results for the traditional Recovery postulate for base contraction. Thus, reconsideration imparts a Recovery aspect to belief bases.

Although $\sigma$-liberation retraction was never intended to adhere to Recovery, if we require that contractions are for consistency maintenance only, it adheres identically as well as kernel contraction adheres to OR.

Ongoing work involves formalizing reconsideration for non-linear belief orderings (those lacking comparability and/or anti-symmetry) and exploring adherence to the recovery-like postulates in [Chopra et al., 2002] (altered using optimized-addition).

Acknowledgments

The authors are grateful for the support, insights and feedback of William J. Rapaport, Carl Alphonce, Ken Regan, David R. Pierce, Jan Chomicki, Samir Chopra, Thomas Meyer, and the SNePS Research Group.

References


Actions as a Basic Software Concept
in the Leonardo Computation System

Erik Sandewall
Department of Computer and Information Science
Linköping University
Linköping, Sweden
erisa@ida.liu.se

Abstract

The work reported here is performed in a broader context where we propose to change the overall software architecture (operating systems, programming languages, etc etc) in order to eliminate the considerable redundancy of concepts and constructs that contemporary software technology exhibits. This requires, among other things, a realignment so that some constructs that used to be placed on higher levels of software now become incorporated in a kernel on a much lower level.

In this framework, we propose in particular to use the construct of an action already in the kernel, whereby it becomes available for many applications as a conceptual and computational resource and in a uniform fashion. The article describes and discusses the ramifications of this approach, including how it relates to the current state of the art in logics of actions and change, as well as the non-monotonic character of one of its computational constructs.

The article has been written for the purpose of a workshop, so its contribution is in the range of ideas and as a discussion-starter. It does not pretend to report finished results.

1 Actions as a Basic Software Concept

Actions, in the sense of a process that changes its environment in a describable, and often goal-directed way, occur in several kinds of software systems. They are important constructs in 'intelligent agents' and they also occur in some simulation systems, for example. However, in all of these cases the actions occur in a relatively high layer of the overall software architecture of the computer. Lower layers include the operating system, the programming language and its environment, communication systems such as CORBA or OAA, possibly combined with database systems, and so on.

In this article we propose to change things around so that actions are introduced in a much lower level of the overall software architecture, in a way whereby they become available for many applications as a conceptual and computational resource and in a uniform fashion. This proposal is part of a broader idea concerning software reform in order to integrate the traditional concepts of operating system, programming language, database system, document formatting system, and several others. The reason for doing this is that the traditional overall structure of software contains a lot of conceptual redundancy: similar, but not equal conventions and constructions are introduced in different parts of the overall system. It ought to be possible to design the system in such a way that this redundancy is eliminated.

We are in the process of designing an experimental language and system, called Leonardo, for the purpose of investigating the feasibility of such a reform. One important aspect of Leonardo is that actions occur already in the system kernel. In fact, the proposed reorganization tends to change things around in more than one way, so that things that used to be thought of as 'high level' now become incorporated in the kernel, which is a natural consequence of the desire to remove conceptual redundancy.

The present article will first give a quick overview of the present version of Leonardo, with particular consideration of its action facilities, and then proceed to a discussion of how this relates and may relate to research about actions and change. Since this article is intended for a workshop, it combines presentation of some results achieved with a discussion of design issues that are still somewhat open, and of possibilities for future development in the cross-section between programming systems on one hand and logics of actions and change on the other.

The name of 'Leonardo' was chosen after Leonardo da Vinci, and since we believe in the need for a renaissance in software technology - a renaissance where many existing dogmas are rejected and where we return to some of the concepts that were invented long ago but have been forgotten meanwhile.

2 Functional Aspects of Leonardo

The Leonardo representation language can be thought of both as a programming language and as a knowledge representation language, and in fact it should also be used for the purpose that is commonly served by the 'shell' command language for the operating system. We believe that a single language kernel should be used for these purposes and several others, albeit with variations that adapt it to interpretive
or compiling environments. The basic design considerations are:

- to stay as close as possible to the notation of set theory and other discrete mathematics, while using the 8-bit standard ASCII character set,
- to favor powerful, orthogonal concepts and constructs,
- to view the entire Internet as the logical ‘memory’ (resource for storage and retrieval of data) of the language.

This language has a functional aspect and an action/agent aspect. The present section contains an outline of the functional aspect.

2.1 Expressions

The basic data thing in Leonardo is called an expression. We say ‘data thing’ since objects are another kind of thing. There are data expressions and text expressions which have different syntax, but each of them can be embedded inside the other, recursively, in some positions requiring an escape character. The following data expression:

\[
\text{[automobile: :brand Volvo :year 2005 :type sedan]}
\]

is a record, presumably denoting a description of a particular car, for example as a database query. The following text expression:

\[
<\text{[style: :bold t :font ariel] boldface text in ariel font}>\]

presumably denotes five words that are typeset in boldface ariel font. The first subexpression of the text expression is a record (hence, a data expression) specifying the formatting. Argument lists that appear in records, forms, and a few other constructs may have a few initial, untagged elements in prescribed order, followed by optional, tagged elements in arbitrary order (like in CommonLisp).

Leonardo allows several other kinds of data expressions besides those that were exemplified above. There are expressions for sets, sequences, mappings, texts, and a few more. We try to keep the notation as close to traditional set-theory notation as possible. The syntax for text expressions allows for basic markup. Sets can be specified both by enumeration of their members, by a characterizing property (“the set of all x such that ...”), and by standard operations on sets. Functions are viewed as mappings which are a kind of sets; recursive functions are characterized as set-valued solutions of equations in the obvious way.

We refer to sets, sequences, mappings, texts, etc as different sorts that are represented by data expressions, and to the variety of records that are denoted by the initial symbol in the record expression, as being different types. The Leonardo representation language specifies the syntax for each of the sorts, but types can be checked dynamically. Structure specifications for types are optional, and are not part of the system kernel.

2.2 Unification of Ontology and Computation

Leonardo expressions are intended to be used both as a representation language (corresponding to frames, XML, or semantic-web languages such as OWL) and as a programming language for defining computational processes. It is therefore adapted to ontology-oriented (i.e. data-model-based) programming, where one first defines the ontology for the application at hand and represents it formally, and then the elements of the ontology are used as carriers for the definitions of data types, procedures, and other expressions that are needed for defining and performing the computation.

From the perspective of programming languages, the Leonardo representation language is oriented towards functional programming, and it is set-theory-oriented in the same sense as e.g. Prolog is logic-oriented. From the perspective of representation languages, we believe that the possibility to define functions in the usual mathematical sense (mapping from arguments to values) is an important feature in any representation language, which adds to the good reasons for integrating the notations used for knowledge representation and for programming.

Besides the ontology aspect and the procedural aspect, there is also a document processing aspect in the Leonardo representation language. We consider documentation to be a fundamental part of any software system, and therefore documentation should be integrated as well as possible with the other aspects of the software. This is why ‘text’, including structured and marked-up text is also defined as expressions in the Leonardo representation language.

2.3 Locations and References

Expressions can be manipulated directly by computational processes in a Leonardo system, but they can also be deposited in locations. Locations are important for actions, which are our next topic, since computational actions often operate on the contents of locations. For example, the action of running a document through latex is considered as operating on a location containing both the source (.tex) and target (.pdf) version of the document, as well as other, related files.

From the programming-language perspective, on the other hand, locations are used both like ‘variables’ in conventional programming languages, and like filenames in a conventional programming system, and like URLs. In fact, the entire Internet ‘address space’ (thinking of a URL as an address in the universal computer) constitutes most of the location space from Leonardo’s point of view.

Each location is used for a particular sort, but record locations do not make any assumption about the type of their contents, in line with the interpretive character of the kernel Leonardo system as a whole.

A reference is a formula that specifies a location and that is used, in the language, for denoting the expression that is currently deposited in that location. For example, the following reference

\[
[?filerec: "C:/leo/doc/" section2 vfr]
\]

denotes a record that is stored in the location C:/leo/doc/section2.vfr according to the deposition method (“format”) specified by the name filerec. (This is not an actually existing deposition method, it is given by way of example). Similarly, the following reference:

\[
[?textfile "C:/leo/doc/" section2 txt]
\]
denotes a text in .txt format that is currently deposited in the location C:/leo/doc/section2.txt; the operator `textfile` specifies how it is accessed. Record references and text references are distinguished by the fact that the de- position method of the former ends with a colon character.

In these examples the path to the file in question was represented as a string. Other methods than `filename` and `textfile` would be used if the path is to be represented as a sequence instead.

These examples suggest that files in contemporary operating systems, such as Linux or Windows can be used as locations. This is certainly possible but not the only possibility; other basic storage facilities such as object-oriented ones may be more appropriate in the future. Leonardo's representation for references allows for several possibilities.

Reference expressions can be nested, like other expressions, so one can for example write

```
[?textfile
  [?textfile "C:/leo/" curdir txt] section2 txt]
```

whereby the system, when using this expression, will use the contents of the file C:/leo/curdir.txt as a directory name which is combined with the filename section2.txt to obtain a file whose contents are in turn obtained as a string.

References may be passed as arguments to functions or other computational entities. In this sense they are analogous to pointers in conventional programming languages. At some point the reference has to be resolved, for example by obtaining the contents of a textual location as an actual text. This is often done as a single operation like if, in a conventional programming language, a text file is read into working memory at one stroke and becomes a string.

3 Action and Change Aspects of Leonardo

3.1 Action Expressions, Agents, and Actions

We turn now to the topic of action and change. There are three related concepts in Leonardo, namely action expressions, agents, and actions. The following is an example of an action expression:

```
[fly-to! :agent witas-4
  :destination [geo-coord: 425 862]]
```

saying that the agent witas-4, which is an unmanned helicopter, shall fly to the point located at geographical coordinates (425, 862). The following action expression:

```
[add-to-account!
  :agent [?account-agent@ mybank]
  :account 634422
  :amount [money: ECU 4900]]
```

says that the agent [?account-agent@ mybank] should find some way of adding 4900 ECU to the current balance of account number 634422, by whatever means it can find of raising the money.

Each action expression must have an explicit or defaulted :agent field specifying what agent is responsible for performing the action. There is a variety of ways of specifying the agent, as these two examples have indicated. If an action expression is presented to its agent and that agent accepts to execute it, then an action arises, that is, a computational process that is usually a lightweight one and that proceeds through a sequence of steps. The action has a local state during and after its execution. The local state is always a record, and one part of the definition of an action specifies the next-state function that is applied in each step of its execution.

3.2 The Top Level of a Leonardo System

The top level of conventional imperative languages, such as Lisp, is a read-eval-print loop. The top level of Leonardo is instead an executive for actions, somewhat similar to an object-oriented simulator. It maintains a set of pending action expressions and a set of working actions. Pending action expressions refer to actions that have been requested to the system but that have not yet started to execute, for example because not all their preconditions are satisfied, or because of concurrency constraints. Working actions are actions that have started to execute but which have not yet arrived to a quiescent state.

In its normal main cycle, Leonardo first allows the user to enter an action expression and adds it to the pending set. After that, it checks for each member of the pending set whether its preconditions are satisfied, including concurrency restrictions, and if so a new action is initiated as specified by that action expression and added to the working action set. Finally, the executive visits the working actions and performs the update in each of them, according to the specifications that are given by the agent of the action in question. Actions that have reached a quiescent state are moved from the working set to an archive of past actions.

This general formulation of the top-level loop can be adapted, specialized, or extended in various ways. A conventional read-eval-loop can be implemented using an 'eval' verb whose actions always finish in one step. Background tasks such as fetching information from remote websites can be set up as actions with extended duration. Simulators of, and supervisors of physical robotic equipment, as well as servers can also easily be represented in the same structure. Actions are of course allowed to invoke sub-actions, the process of the main action being conditional on the process of its sub-actions. The subactions need not use the same agent as the invoking action; this is our counterpart of message-passing between agents.

3.3 Specifications of Actions

Actions in Leonardo are characterized using Cognitive Robotics Logic (CRL)5 which is a refined temporal logic[9], based on 'Features and Fluents'4 and closely related to Doherty’s Time and Action Logic[1] (TAL), and having many points in common with modern event calculus as presented by Shanahan[8].

The behavior and the effect of actions is defined by a combination of the action-verb and the agent performing the action. The external behaviors of actions are normally ex-
pressed in CRL and are specified in terms of preconditions, postconditions, and conditions characterizing concurrency restrictions and other aspects of intermediate states.

The internal behaviors of actions specify the details of their execution in terms of updates of its current state at successive (but not necessarily contiguous) timepoints. Each action has a current state, starting with the timepoint when it was initiated; this current state must always be a record. In addition, each agent has its current state, which is also a record. The next-state transformation defining one step in the execution of an action has access to both of these records and is able to update them both, modulo constraints that can arise by interference between concurrent actions. It can be expressed in a variety of ways:

- As a computational procedure
- As a finite-state or hybrid automaton
- By a combination of discrete state transitions and partial evaluations of state-expressions (details below)
- As a sequence or other temporal structure of subactions

In all these cases the behavior may refer either to computations that are performed within the computer at hand, or to actions performed by a robot under the active control of that computer, or to actions that are performed independently of the computer but are observed by it.

Definition using partial evaluation offers a structured but expressive way of characterizing 'hybrid' actions that combine gradual change of state with occasional qualitative changes. The current state of the action is expressed as a record, like for all actions in Leonardo. This record is an expression that may contain unevaluated forms. In each cycle, the system traverses that record, replacing variables by their values, when available, and evaluating forms (functions with arguments) when possible. Individual symbols are left as they are, and records, sequences, and sets are merely traversed, i.e. their components are evaluated but the structure of the record (etc) is retained. In particular, an expression not containing any variables or forms always evaluates to itself. After that, a set of qualitative state transitions is compared to the state at hand, and any applicable transition is performed.

3.4 Prediction and Planning

The design of Leonardo does not attempt to make it into an AI system from the start. Instead, the idea is to design a kernel that can be used as a platform for many common programming tasks, and one that is more powerful than what conventional software technology can offer. For this reason, the kernel Leonardo system does not contain full-fledged facilities for prediction and planning, but it does contain handles where such facilities can be plugged in effectively, in those applications where they are considered appropriate and useful.

We described above how the top level of the Leonardo system maintains a set of pending action expressions, and how in each cycle it selects those for which the preconditions are satisfied. The treatment of those action expressions for which the preconditions are not satisfied is a natural handle. The kernel system does not do anything about them and just leaves them in the pending set, but it is possible to define other handlers for precondition failures. Planning and plan execution capability can therefore be implemented by a routine that applies to precondition-failing action-expressions, selects a plan, and adds the plan to the set of pending actions, while considering the plan as a composite action.

3.5 Additional Topics

The full Leonardo design includes a number of additional aspects that are not covered here since they are less central for the question of the relationship to logics of action and change. Those additional aspects include, for example what to do if there is no applicable transition rule or invocation rule, and what to do if there are several concurrent action requests for the same agent. They also include questions of names, symbols, and namespace, questions of persistent objects and the use of locations, and of version management for the properties of objects. For the low-level part of robotic applications, there are questions of shared record fields or transfer of data streams, for use in the connections between sensors, controllers, and actuators on several abstraction levels. Forthcoming additional reports addressing these topics will be posted on our website (references at the end of this paper).

4 The Leonardo Timeline

Since actions are specified using Cognitive Robotics Logic (CRL) in Leonardo systems, each action that is performed there is characterized by two timepoints, namely its starting timepoint and its ending timepoint. The system also administers features where, as usual, Holds(t, f, v) expresses that the feature f has the value v at timepoint t, and actions can be characterized as depending on, and affecting the values of features.

4.1 Timepoints during Computation Sessions

Each computational session defines a sequence of timepoints that are numbered from 0 and up, and that are related to physical time as follows. Physical time is assumed to be metric and can be measured e.g. in milliseconds. The physical timeline is divided into two kinds of intervals that alternate, namely timepoints and action-periods. The ending-time of a timepoint is the starting-time of the succeeding action-period, and vice versa, and each timepoint and each action-period is an interval on the physical timeline.

Consider in particular the case where the Leonardo system operates a read-invoke-print loop, as described in subsection 3.2. One may then consider the physical time period where the user first thinks for a while, and then types in an action expression, as a timepoint in the Leonardo sense. For simple, single-cycle actions the following action-period will be the period when the action gets executed, and the next timepoint will be the physical period where the next action expression is decided and typed in.

If actions extend over several cycles, then each action-period can contain timeslots for several of the actions that go on at that time. However, the 'starting time' of an action from the point of view of the system will be the last timepoint (in our specific sense of that word) before the first action-period
where the action got to execute, and similarly the ‘ending
time’ of the action will be the first timepoint after the last
action-period where the action operated.

The aforesaid applies to computational actions within the
computer at hand. For robotic actions and other actions that
are performed outside that computer, actions can of course
usually be performed with true concurrency, and the duration
of an action will be defined in terms of when and how it was
controlled or observed.

Other ways of using the timeline are also possible and
useful. For example, in a natural-language dialog system it
may be appropriate to consider each input of a spoken phrase
into the system, as well as each output phrase that is pro-
duced by the system as an action in itself. In this case,
Leonardo timepoints should characterize the starting-time
and the ending-time of each input and output, instead of ‘con-
taining’ such inputs. Break-ins and other situations where the
user and the system perform concurrent speech acts or other
communication acts can then be represented in a natural man-
ner.

4.2 Computational Ramification

Since timepoints in Leonardo are defined in terms of the
physical timeline, it is not necessary to let all timepoints and
all action-periods have similar size; it is perfectly possible to
let them be different even by orders of magnitude.

Consider for example the following situation. The system
has decided to perform action A as a prerequisite for perform-
ing action B, where both A and B are physical actions by a
robot which require nontrivial time. The stated effects of ac-
tion A do not exactly match the preconditions for action B,
and a few steps of logical deduction are required to infer that
B can now be performed. Furthermore, these steps of logical
deduction also check that some other conditions still apply
and have not been invalidated while A was being performed.

The step from the immediate effects of A to the precondi-
tions needed by B may be considered as ramifications, and
in line with the usual treatment of ramifications in NRAC it
would be natural to do those deductions within the last action-
period of A, so that they are available in the timepoint that is
the ending-time for A. However, there is also another possi-
bility, namely considering those deductions as additional ac-
tions that take place after the action A has ended, albeit ac-
tions that execute very rapidly. The resulting timeline will
then contain some timepoints that are wide apart, in particu-
lar the timepoints characterizing many physical actions, but
it will also contain clusters of timepoints that physically oc-
cur in rapid succession, namely, timepoints that separate the
inference actions.

The latter approach entails some advantages, such as the
possibility of treating more or less complex inference activi-
ties as actions that can be subject to planning and other cogni-
tive activities. It also introduces some problems, in particular
the need to distinguish between cognitive (computation) time
on one hand, and real-world time on the other. Anyway, it is
an approach that makes computational sense, and that raises
some interesting problems for the corresponding logical sys-

5 Discussion

We proceed now to discussing the potential relevance of
research on nonmonotonic reasoning, actions and change
(NRAC)

5.1 The Relevance of Leonardo Systems for NRAC

The major reason why Leonardo may be relevant for rea-
soning about actions and change is by demanding extensions
to the logic while at the same time being precise about what
is required for the extension. The Leonardo design allows
for concurrent actions, subactions, delayed effects, and others
more. It also contains an explicit notion of an agent, and ways
of specifying whether an agent is able and willing to perform
an action requested from it. The list can be continued. These
are phenomena that ought to be represented in logics of ac-
tions and change, but which are incompletely understood at
present.

For these reasons, a Leonardo system may be seen as
a precisely defined model environment for a logic of actions
and change, that is, as an underlying semantics for it in the
sense that was introduced in ‘Features and Fluents’.[4]. In
that book I defined an underlying semantics and used it for as-
sessing the range of applicability of about a dozen nonmono-
tonic entailment methods for such logics. This underlying se-
manics was a kind of simulation, expressed in set-theoretic
concepts, that represents the actions of an agent in the world
being modelled. However, the underlying semantics of ‘Fea-
tures and Fluents’ does not model the sensor/actuator level in
any meaningful sense. (I intended to include a chapter with
those contents, but was not able to complete it in a way that
I was happy with at the time the book was being written). I
hope at present that the Leonardo language can be used for
defining a more expressive and realistic underlying seman-
tics.

The possibility of nonmonotonic partial evaluation is an
interesting one. We described above how the update of the
current state of an action is viewed as partial evaluation of
its local state record. This partial evaluation is robust with
respect to lack of information: it will then keep the unevalu-
ated variable or form, and try again in the next cycle. How-
ever, it would make sense to also allow operators under which
the partial evaluator is allowed to replace a form by a default
value even though the information required for standard eval-
uation is missing. This facility ought to be of interest both
for characterizing defaults due to timeout, and defaults on the
level of logical reasoning.

5.2 The Relevance of NRAC for Leonardo Systems

The other question is as follows: if the Leonardo proposal
is used in the design of a software system that in turn is used
as a platform for various applications, will it then be possible
to use existing logics of actions and change to characterize
that system or its applications, or even to help in building
them?

One feature of Leonardo is particularly important from
this point of view, namely the system support for distinct

2Here the acronym NRAC is used for the research area and does not specifically refer to the NRAC as a workshop series.
timepoints whose ontology is consistent with a logic of actions and change. In this way the current state of each action is easily accessible, describable, and available for inference.

To the extent that a logic of actions and change can be made to apply to a Leonardo system, several uses of it come to mind:

- Use of the logic for specifying the execution of actions in effective form, so that it can be used to define the behavior of agents;
- Use of the logic for characterizing actions for the purpose of planning, and possibly for diagnosis, for example by specifying pre- and postconditions for those actions;
- Use of the logic for specifying aspects of goal-directed or social behavior, for example, when to inform another agent about some facts (and when not to do it), how to handle the failure of an action, and so forth.

The last item is a particularly interesting one since it opens up the possibility of generic services that can apply across several domains. In this context it should be an advantage to work with a general-purpose notation, as offered by Leonardo, rather than a specific notation that has been developed for a particular application. The simplicity of the Leonardo notation, which is due to it staying close to the notation of set theory, will then also be an advantage in comparison with XML and other notations that originate in the semantic web initiative and that have a much more elaborate structure.

6 Related Work

The key idea in Leonardo that is particularly relevant for the present workshop, is preparing for a synthesis between logics of actions and change, and a high-level programming language. From the language point of view, almost all the constructs in the proposed Leonardo language are themselves well-known. It is not our intention to invent new and very original constructs for the language, but instead to compose selected, well-known concepts in a new way for the purpose of obtaining a basic software architecture that is powerful and exhibits as little conceptual redundancy as possible. This is necessary as a prerequisite to the "meeting" with logics of actions and change, or in fact, with any kind of logic.

This view must however be combined with the actions-and-change point of view where the idea of integrating a logic of actions and change with a programming language is relatively unique, probably the major exception being the use of Prolog for the event calculus, by Shanahan [7] and others. Leonardo differs from these in a number of ways: use of several sorts including records; more general use of set theory notation for defining functions; use of expressions for marked-up text and for web-level references, and so on. The Golog [3] language also represents a step from the situation calculus, as a logic of actions and change, towards a programmable system, but it seems to be even more remote from the expressiveness of a programming language than what one finds in Prolog.

On the programming-language arena, the most strongly 'related works' are the following. First of all, the Lisp language and system, and in particular the Interlisp system [10] and the software systems of the various Lisp machines that were designed in the 1980's. Interlisp pioneered the idea of a programming environment which has then been inherited by other languages and communities, and the Lisp machines showed that it was possible to integrate operating system and programming language in a strong way.

The Smalltalk language and system [3] has integrated concepts from Lisp and Simula [4], and seems to be the strongest follower of the Interlisp design philosophy today. The Perl language [5] shows in a modern setting how the facilities that are needed on the command level of the operating system can be extended into becoming a serious although still special-purpose programming language.

The first use of set theory for programming was to my knowledge the SETL language [6]. The style of defining functions by cases in Leonardo, including for the definition of recursive functions, is similar to the style in the Erlang [6] language which in turn obtained it from Prolog [7].

The definition of the top-level loop implements concurrency in a way where the current state of each concurrent action is open to inspection and can be referenced. By comparison, management of concurrency using ‘detach’ and ‘resume’ operators during an evaluation process requires complex stack management methods that are (intentionally) hidden from the program. The same applies for the use of continuations. The approach used by Leonardo may be perceived as more restrictive, but it is closer to the representation of current state in logics of actions and change, which is a distinct advantage from our point of view.

On another note, a number of agent systems and languages contain concepts and constructs that have been taken up in Leonardo. Among many, particular mention of RAPS [2] whose influence on Leonardo is evident.

7 Design History and Implementation

The core constructs in Leonardo have been implemented in CommonLisp, including functions for reading, printing, and evaluating Leonardo expressions, and for defining and executing actions. Several other facilities in Leonardo are similar to constructs that already exist in the DOSAR robotic dialog system that has been developed as part of the WITAS project [8], and we expect to be able to migrate them rapidly to the emerging Leonardo system. A description of DOSAR can be found on the website of the CASL research group [9]. Forthcoming additional articles about Leonardo will also be posted there.

The concepts that have been synthesized into Leonardo have been present in our own work during a long time and
have evolved gradually. Many aspects of the language design, including the proposed design of agents, have also been influenced by the experience of building the robotic dialog system and its auxiliary robot simulator, as a part of the WITAS project. This applies in particular for the view of the top-level executive of the system which bears some resemblance with the robot simulator that was implemented as a tool for the development of the dialog system.

The long-term research background for the present work is documented on the CAISOR website\textsuperscript{10}.

Acknowledgements

We gratefully acknowledge the following support. The WITAS project has been sponsored in full by a grant from the Knut and Alice Wallenberg Foundation. The concluding step of work leading to the present article was done while the author was visiting the Contraintes research group in the INRIA Research Center at Rocquencourt (Paris), France.

References


\textsuperscript{10}http://www.ida.liu.se/ext/caisor/
Action, Belief Change and the Frame Problem: A Fluent Calculus Approach

Richard B. Scherl
Computer Science Department
Monmouth University
West Long Brach, NJ 07764
U.S.A.
email: rscherl@monmouth.edu

Abstract

This paper develops a solution to the analogue of the frame problem that arises when the belief state of an agent is axiomatized in the presence of belief changing actions. It follows the work of Scherl and Levesque which adapted the approach to the frame problem of Reiter to the case of the analogue of the frame problem that arises when knowledge and knowledge producing actions are added to the situation calculus. For the case of belief, it is necessary to use the somewhat more expressive relative of the situation calculus, the fluent calculus which is a formalism that allows quantification over states and fluents.

1 Introduction

The frame problem was noted early on in the study of formalizing actions and their effects on the world [MH69]. In work in this area, axioms are used to specify the prerequisites of actions as well as their effects, that is, the fluents that they change. As noted in [MH69], it is in general also necessary to provide frame axioms to specify which fluents remain unchanged by the actions. Reiter [Rei91; Rei01] has given a set of conditions under which the explicit specification of frame axioms can be avoided.

In [SL03], this solution to the frame problem was extended to cover knowledge-producing actions, that is, actions whose effects are to change a state of knowledge. To incorporate knowledge-producing actions like these into the situation calculus, it is necessary to treat knowledge as a fluent that can be affected by actions. This is precisely the approach taken by Moore [Moo80]. With the presence of knowledge, there emerges a new analogue to the frame problem. It is necessary to ensure that after an action has taken place (whether it be a sensing or a non-sensing action), there are no unwanted losses or gains in knowledge.

In [SL03], knowledge and knowledge-producing actions are handled in a way that avoids this extended frame problem: they are able to prove as a consequence of their specification that knowledge-producing actions do not affect fluents other than the knowledge fluent, and that actions that are not knowledge-producing only affect the knowledge fluent as appropriate. In addition, they show that memory emerges as a side-effect: if something is known in a certain situation, it remains known at successor situations, unless something relevant has changed.

But this work only considers knowledge and knowledge producing actions. That is, it is assumed that the agent’s beliefs and sensor results are all correct. The approach simply fails when the agent being modeled acquires information that contradicts its knowledge. The agent then knows all sentences of the language since there will be no accessible possible worlds/situations as all accessible worlds/situations in which the new piece of information is false are eliminated. It is clearly unrealistic given the goals of cognitive robots to limit attention to agents who begin with only correct beliefs about the world.

What is needed is the incorporation of some sort of belief revision into the framework. In [SPLL00] the model of [SL93; SL03] is extended to include a process of belief revision. Additionally, [JT04] extend the closely related fluent calculus to incorporate belief revision. But they do not address the solution to the analogue of the frame problem that arises with belief.

This paper addresses the problem of dealing with these unwanted changes in belief when a new fact comes to be believed. In order to do this, we need greater expressivity than is allowed in the situation calculus. We need the ability to quantify over fluents and states. As this is allowed by the fluent calculus [Thi98; Thi00], the fluent calculus is utilized in this paper. In this paper, we limit our attention to knowledge of sentences in a propositional language.

Like [SPLL00] and [JT04], the approach developed here is able to incorporate both revision and update [KM91a] into a uniform framework. Changes in belief due to the incorporation of new information through sensing respect the revision postulates. Changes in belief due to changes in the world respect the update postulates. But the main contribution in this paper is an investigation of the analogue of the frame problem when belief is incorporated into situation/fluent calculus action theories.

2 The Fluent Calculus: A Language for Specifying Dynamics

The fluent calculus[Thi98; Thi00] is a many-sorted language with the sorts action, sit, fluent, and state. Fluents are reified.
In other words, they are terms. States are also terms which are constructed out of fluents with the binary function symbol \( \circ \). In the following, the letter \( f \) is used for fluent variables, the letter \( z \) for state variables, \( s \) for situation variables, and \( a \) for action variables. These letters may have superscripts or subscripts. Unless otherwise indicated, all variables in formulas are universally quantified.

Unlike the situation calculus, the fluent calculus separates the notion of state and situation. In the fluent calculus, situations contain a history of actions that have been performed and states contain the fluents that hold in that state. Each situation has an associated state.

The function symbol \( \circ \), used to construct the state terms, is axiomatized to be associative, commutative, and has a unit element \( \emptyset \). The following set of axioms (AC1) ensures these properties:

\[
\begin{align*}
(z_1 \circ z_2) \circ z_3 &= z_1 \circ (z_2 \circ z_3) \\
z_1 \circ z_2 &= z_2 \circ z_1 \\
z \circ \emptyset &= z
\end{align*}
\]

Additionally, unique name axioms for state terms are needed. These (EUNA) are given below:

\[
\begin{align*}
z &= f \rightarrow z \neq \emptyset \land \exists z' \circ z'' \rightarrow z' = \emptyset \lor z'' = \emptyset \\
z_1 \circ z_2 &= z_3 \circ z_4 \rightarrow (\exists z_a, z_b, z_c, z_d)[z_1 = z_a \circ z_2 = z_c \circ z_d \land z_3 = z_a \circ z_c \land z_4 = z_b \circ z_d]
\end{align*}
\]

We also have the foundational axiom

\[
\text{STATE}(s) \neq f \circ f \circ z
\]

which prohibits double occurrences of fluents in states. Additionally, we have the following abbreviations:

\[
\begin{align*}
\text{Holds}(f, s) &\overset{\text{def}}{=} \text{Holds}(f, \text{STATE}(s)) \\
\text{Holds}(f, z) &\overset{\text{def}}{=} (\exists z') z = f \circ z' \\
\text{Holds}(\neg \varphi, z) &\overset{\text{def}}{=} \neg \text{Holds}(\varphi, z) \\
\text{Holds}(\varphi \land \phi, z) &\overset{\text{def}}{=} \text{Holds}(\varphi, z) \land \text{Holds}(\phi, z)
\end{align*}
\]

We require that for each action \( A(\vec{x}) \), there is a precondition axiom of the form,

\[
\text{Poss}(A(\vec{x}), s) \equiv \pi(\vec{x}, s)
\]

Additionally, state update axioms are needed to specify the relationship between states at two consecutive situations. Below is the general form:

\[
\text{Poss}(A(\vec{x}), s) \rightarrow \text{state}(\text{DO}(A(\vec{x}), s)) \circ \text{v}^- = \text{state}(s) \circ \text{v}^+
\]

Here \( \text{v}^- \) are the negative effects and \( \text{v}^+ \) are the positive effects of action \( A \). An example is:

\[
\begin{align*}
\text{Poss}(\text{open}(\text{Door}_1), s) &\rightarrow \\
\text{state}(\text{DO}(\text{open}(\text{Door}_1), s)) &= \text{state}(s) \circ \text{closed}(\text{Door}_1)
\end{align*}
\]

After the execution of an open action, the door is no longer closed. It has been shown [Thi99] that a collection of state updates in this form constitute a solution to the frame problem.

### 3 Adding Belief to the Fluent Calculus

For belief we can adapt some of the machinery [Thi98; Thi00] developed for the case of knowledge. We have a predicate \( \text{BSTATE} \) of type \( \text{sit} \times \text{state} \) indicating that the second argument is a possible state of the situation in the first argument. Intuitively, something is believed in a situation if it holds in each of the belief states associated with that situation. We need an axiom similar to the foundational axiom given earlier:

\[
\text{BSTATE}(s, z) \rightarrow \forall f, z' z \neq f \circ f \circ z'
\]

**Believes**\( (\varphi, s) \overset{\text{def}}{=} (\forall z)\text{BSTATE}(s, z) \rightarrow \text{Holds}(\varphi, z) \)

where \( \text{Holds} \) is as defined previously.

Belief in the initial situation can easily be specified as follows:

\[
\text{Believes}(P, S_0) \quad \text{Believes}(\neg Q, S_0)
\]

We want to model actions that provide the agent information about the state of the world. For example, we might imagine a \( \text{SENSE}_P \) action for a fluent \( P \), such that after doing a \( \text{SENSE}_P \), the truth value of \( P \) is believed. We introduce the notation \( \text{Bwether}(P, s) \) as an abbreviation for a formula indicating that the truth of a fluent \( P \) is known (in the sense of belief) by the agent.

\[
\text{Bwether}(P, s) \overset{\text{def}}{=} \text{Believes}(P, s) \lor \text{Believes}(\neg P, s),
\]

Certainly, the effect of a \( \text{SENSE}_P \) action is \( \text{Bwether}(P, s) \).

The next step is to correctly axiomatize changes in the belief accessible states. The issue is what is the relationship between the states \( (z) \) for which \( \text{BSTATE}(s, z) \) is true and the set \( z' \) for which \( \text{BSTATE}(\text{DO}(a, s), z') \) is true. We might continue to follow [Thi98; Thi00] and develop belief update axioms of the form:

\[
\text{Bstate}(\text{DO}(a, s), z) \equiv \exists z' (\text{BSTATE}(s, z') \land \Psi(z, z', s))
\]

Here \( \Psi \) is a first-order formula expressing the relation between the two sets of belief states. The following is an example:

\[
\text{Poss}(\text{SENSE}_P, s) \rightarrow \\
\text{BSTATE}(\text{DO}(\text{SENSE}_P, s), z) \equiv \text{BSTATE}(s, z) \land \\
[\text{Holds}(P, z) \equiv \text{Holds}(P, s)]
\]

But the problem here is that the agent may already believe that \( \neg P \) holds and then there will not be a \( z' \) such that \( \text{BSTATE}(\text{DO}(\text{SENSE}_P, s), z') \). Then the agent’s beliefs will be in a state of contradiction as for any proposition \( Q \), both

\[
\text{Believes}(\neg Q, \text{DO}(\text{SENSE}_P, s))
\]

and

\[
\text{Believes}(Q, \text{DO}(\text{SENSE}_P, s))
\]

will hold. Revision must occur to prevent the agent from believing falsity.
4 Axiomatizing Changes in Belief

Here a successor state axiom is developed for specifying the belief set (i.e., those \( z \) such that \( \text{Bstate}(\text{DO}(a,s),z) \) holds) in terms of the belief set at the previous situation (i.e., the those \( z' \) such that \( \text{Bstate}(s,z') \) holds), the action \( a \) and the result of the sensing (if the action was a sensing action).

It is necessary to distinguish between 3 possible cases.

- The action was not a sensing action.
- The action was a sensing action and the result did not contradict the previous beliefs.
- The action was a sensing action and the result did contradict the previous beliefs.

To simplify matters, following [SL03], all actions will be either pure sensing actions that do not alter the world or ordinary actions that only alter the world and do not provide any information to the agent beyond the fact that the action has occurred. It is necessary to require that the axiomatization correctly distinguishes between sensing (information-producing actions) and ordinary actions that alter the state of the world. For every action \( a \), the axiomatization must entail either that \( \text{TYPE}(a) = "\text{SENSE}" \) or \( \text{TYPE}(a) \neq "\text{SENSE}" \).

The successor state axiom for Bstate requires some additional machinery as well. In general, there may be many information-producing actions, as well as many ordinary actions. To characterize all of these, we have a predicate \( \text{SR} \) (for sensing result), and for each action \( a \), a sensing-result axiom of the form:

\[
\text{SR}(a,s,z) \equiv \phi_a(s,z)
\] (8)

The following is a \( \text{SR} \) axiomatization for an action that determines accurately whether or not \( P \) is true in the current state.

\[
\text{SR}(\text{SENSEP},s,z) \equiv (\text{Holds}(P,z) \land \text{Holds}(P,s))
\] (9)

Since the situation is also an argument to \( \text{SR} \), it is possible to axiomatize functions for sensors that are not accurate, but rather give different results depending on the situation; regardless of the current state. For example:

\[
\text{SR}(\text{SENSEP},s,z) \equiv (\exists a,b \text{ s} = \text{DO}(a,\text{DO}(b,S_o)) \land \text{Holds}(P,z))
\] (10)

The idea is that if \( \text{SENSEP} \) is the third action to occur from the beginning of the history, the result of the sensing will be that \( P \) holds regardless of whether it actually does. There are many other possibilities.

The \( \text{SR} \) axiom for ordinary (non-sensing) actions are all a default with true for the \( \phi_a(s,z) \). For example

\[
\text{SR}(\text{PICKUP},s,z) \equiv \text{TRUE}
\] (11)

For ordinary actions, we need to have a correctly axiomatized state update function \( \text{SUF} \) of the following form:

\[
\text{SUF}(\text{PICKUP}(\text{obj}_1),z) = z' \equiv z' \circ z_2 = z \circ z_1
\] (12)

Consider the following two examples:

\[
\text{SUF}(\text{PICKUP}(\text{obj}_1),z) = z' \equiv z' = z \circ \text{HOLDING}(\text{obj}_1)
\] (13)

For sensing actions the \( \text{SUF} \) function needs to have no effect on the state and therefore the right hand side of the equivalence needs to be \( z = z' \) indicating that sensing actions have no effect on the world. For example:

\[
\text{SUF}(\text{SENSE},z) = z' \equiv z = z'
\] (15)

If the result of sensing does not contradict the agent’s previous beliefs, then it is necessary to perform update. In this case the result is similar to that of [SL03]. But the complicated case is when the sensing contradicts the agent’s previously held beliefs. In this case revision must occur.

Here an ordering on states is needed. Peppas, Foo, and Nayak [PFN00] develop a domain-independent criterion for measuring the similarity between two alternative belief states called PMA (Possible Models Approach) since it is based upon the Possible Models Approach for reasoning about actions[Win88]. The criterion of similarity is based upon the literals which are true in each model or state. Given two states \( w \) and \( r \), \( \text{Diff}(w,r) \) is the symmetric difference of the literals true in \( w \) and \( r \). This criterion is essentially that for a given state \( w \), a state \( r \) is more similar to \( w \) than \( r' \) if \( \text{Diff}(w,r) \subset \text{Diff}(w,r') \). See also [Dal88] and [KM91b].

Peppas, Foo, and Nayak [PFN00] follow Grove[Gro88] and imagine a system of spheres interpreted as a plausibility measure. Similarity is interpreted as differences in the truth of fluents. We imagine that there is a system of spheres centered around each possible world (state). Given a system of spheres \( S \) centered around \( w \) for any possible world \( r \), the smaller \( \text{Diff}(w,r) \) is, the closer \( r \) is to the center, i.e., to \( w \).

In other words, given any two models or worlds \( r \) and \( r' \), if \( \text{Diff}(w,r) \subset \text{Diff}(w,r') \) then there is a sphere \( U \in S \) that contains \( r \) and not \( r' \). Following, Grove when we want to revise a theory by \( \varphi \), the new theory is determined by the most plausible worlds satisfying \( \varphi \). The new worlds are precisely those in the sphere closest to the center that has worlds in which \( \varphi \) is true.

The proposition that state \( z^* \) is more similar to \( z' \) than to \( z \) is to \( z' \left( \text{Diff}(z,z^*) \subset \text{Diff}(z,z') \right) \) is expressed by the following formula:

\[
\forall f \text{ Holds}(f,z') \neq \text{Holds}(f,z^*) \rightarrow \text{Holds}(f,z) \neq \text{Holds}(f,z')
\] (16)

The formula states that every fluent in the symmetric difference of \( z' \) and \( z^* \) is also in the symmetric difference of \( z \) and \( z' \).

All of these notions are then incorporated into the successor state axiom for \( \text{Bstate} \) given below:
Successor State Axiom for Bstate

∀z Bstate(\text{DO}(a, s), z) \equiv

(\text{TYPE}(a) \neq \text{"SENSE"} \land
\exists' Bstate(s, z') \land \text{SUF}(a, z') = z)
\lor
\left(\text{TYPE}(a) = \text{"SENSE"} \land
(\text{POSS}(a, z) \land \text{SR}(a, s, z) \land \text{Bstate}(s, z))\right)
\lor
(\neg (\exists' Bstate(s, z') \land \text{POSS}(a, z') \land
\text{SR}(a, s, z'))\lor
(\exists' Bstate(s, z') \land \text{POSS}(a, z) \land \text{SR}(a, s, z) \land
\neg \exists z''(\text{SR}(a, s, z'') \land z'' \neq z \land \text{POSS}(a, z'') \land
\forall f \text{Holds}(f, z') \neq \text{Holds}(f, z''))\right)
\lor
\neg \exists' Bstate(s, z') \land \text{POSS}(a, z') \land \text{SR}(a, s, z) \land
\forall f \text{Holds}(f, z') \neq \text{Holds}(f, z')
\end{equation}

The idea here is that either the action is a sensing type action (i.e., an action that alters the state of belief, but not the world) or not a sensing type action (i.e., an ordinary action, one that alters the state of the world). If the action is an ordinary action then each belief accessible state must be updated in exactly the same way that the state associated with the situation is updated. This case works exactly as [SL03; Thi00] handle the combination of knowledge and ordinary actions. It is the first disjunct of the successor state axiom.

If the action is a sensing action then either the result of the sensing action contradicts the current state of knowledge or it does not. If it does not, then there must be at least one belief accessible state consistent with the result of the sensing action. This is the second disjunct of the successor state axiom. The belief accessible states accessible after the sensing action are precisely those which were accessible prior to the action and are consistent with the result of the sensing action. In this case things work very much as in the case of knowledge, although there is no guarantee that the fluents true in all of the belief accessible states are in fact true in the actual state. This case works exactly as [SL03] handles the combination of knowledge and sensing actions. It differs from [Thi00] in that the agent does know all of the axiomatized effects of actions.

Now, it very well may be the case that the result of the sensing action does contradict the current state of knowledge. If it does, then there will not be any belief accessible states consistent with the result of the sensing action. Then the belief accessible states are those states which are both consistent with the result of the sensing action and are minimally close to a state which was belief accessible prior to the sensing action. The minimal closeness is handled by the third disjunct. This ensures that if \( z \) is in the new Bstate, then there is no other belief state \( z'' \) in which \( \text{SR}(a, s, z'') \) holds, but which differs in fewer fluents than \( z \) from a \( z' \) in the initial belief state. Note that the \( z' \) in the initial belief state can not be in the new belief state because of the fact that

\[ \neg \exists' \text{Bstate}(s, z') \land \text{POSS}(a, z') \land \text{SR}(a, s, z') \]

holds. But it must be the case that for every \( z \) in the new belief state, there is a \( z' \) in the original belief state to which it is minimally close.

5 Example

An axiomatization of a domain needs the axioms AC1, EUNA, the foundational axiom for STATE, the foundational axiom for BSTATE, the successor state axiom for BSTATE (17), the abbreviations for Holds and Believes, the axiomatization of the initial situation, and for each action a precondition axiom, and a state update axiom, a SR axiom, and a SUF axiom. This set is called \( A \).

The following example is taken from [SPLL00]. There are two rooms \( R_1 \) and \( R_2 \). The agent has one sensor which detects whether or not the light is on in the room in which the agent is located. The other sensor indicates whether or not the agent is in \( R_1 \). The fluents \( \text{LIGHT}_1 \), \( \text{LIGHT}_2 \) indicating that the lights are on in rooms 1 and 2, and also \( \text{INR}_1 \) indicating that the agent is in room 1. If \( \neg \text{INR}_1 \) holds, then the agent is in room 2.

Initially, the lights in both rooms are on and the agent is in \( R_1 \). The agent believes that \( \neg \text{LIGHT}_1 \) and that \( \text{INR}_1 \) both hold. We have:

\[ \text{Holds}(\text{INR}_1, s_0) \land \text{Holds}(\text{LIGHT}_1, s_0) \]
\[ \text{Holds}(\text{LIGHT}_2, s_0) \]
\[ \text{Believes}(\neg \text{LIGHT}_1, s_0) \land \text{Believes}(\text{INR}_1, s_0) \]

We also need the SR axiomatization of the sense action.

\[ \text{Holds}(\text{INR}_1, s) \rightarrow \text{SUF}(\text{SENSE}\text{LIGHT}_1, s, z) \equiv \text{Holds}(\text{LIGHT}_1, z) \]

It follows that:

\[ \text{Believes}(\text{LIGHT}_1, \text{DO}(\text{SENSE}\text{LIGHT}_1, s_0)) \]

This was a case of revision.

The agent also has the capability of moving from one room to another with the LEAVE action. The following is the state update axiom for this action.

\[ \text{POSS}(\text{LEAVE}, s) \rightarrow \]
\[ \text{Holds}(\text{INR}_1, s) \land \text{STATE}(\text{DO}(\text{LEAVE}, s)) = \text{STATE}(s) - \text{INR}_1 \lor \]
\[ \neg \text{Holds}(\text{INR}_1, s) \land \text{STATE}(\text{DO}(\text{LEAVE}, s)) = \text{STATE}(s) + \text{INR}_1 \]

The following is the SUF axiom:

\[ \text{Holds}(\text{INR}_1, z) \rightarrow \text{SUF}(\text{LEAVE}, z) = z' \equiv z = \text{INR}_1 \land \\
\text{Holds}(\neg \text{INR}_1, z) \rightarrow \text{SUF}(\text{LEAVE}, z) = z' \equiv z' = z - \text{INR}_1 \]

The same information is repeated as it is needed in this form for the successor state axiom for Bstate (17). It follows that:

\[ \text{Believes}(\text{LIGHT}_1, \text{DO}(\text{SENSE}\text{LIGHT}_1, \text{DO}(\text{LEAVE}, s_0))) \]

This was a case of update.

6 Properties of the Result

In general, when a sensing action takes place, the result respects the AGM [AGM55; Gar88] postulates for revision. Additionally, when a world changing action occurs, the change in belief respects the postulates of Katsuno and
Mendelzon (KM) for update [KM91a]. The notation \( B_s \) is used to represent the set of sentences believed by the agent at situation \( s \).

\[
B_s = \{ \varphi \mid A \models \text{Believes}(\varphi, s) \} \tag{21}
\]

To make the comparison with the revision/update postulates, it is necessary (following [SPLL00]) to equate a belief set (of the AGM theory) or a knowledge base (of KM) with \( B_s \). Katsuno and Mendelzon (KM)[KM91a] have distinguished between update and revision by stating the AGM postulates for revision as postulates R1–R6 and their postulates for update as U1–U8.

**Theorem 1 (KM Postulates)** When \( B_s \) is viewed as a knowledge base, an axiomatization \( A \) conforms to postulates U1–U4 when update occurs and R1–R4 when revision occurs.

Space does not permit a detailed discussion of all of the postulates for revision and update [KM91a] and a comparison of the properties of the approach described here with the approaches of Shapiro et al. [SPLL00] and the approach of Jin and Thielscher [JT04].

The most important results here are that changes in belief are minimal in the sense of the analogue of the frame problem for belief. There are no unnecessary increases in things believed and decreases in things believed.

First note that for each action, there must be a formula \( \Pi_\alpha \) such that the axiomatization entails

\[
\forall s \ \text{POSS}(\alpha, s) \rightarrow \text{Holds}(\Pi_\alpha, s)
\]

We call the formula \( \Pi_\alpha \), the action precondition formula for action \( \alpha \).

For every sensing action \( \alpha \), there must be a formula \( \Sigma_\alpha \) such that the axiomatization entails

\[
\forall s, z \ \text{SR}(\alpha, s, \text{STATE}(s)) \rightarrow \text{Holds}(\Sigma_\alpha, s)
\]

We call the formula \( \Sigma_\alpha \), the sensed formula for \( \alpha \). For non-sensing actions \( \Sigma_\alpha \equiv T \).

The statement of the theorems to follow requires an additional definition based on \( B_s \):

\[
B_s^{-\varphi} = B_s - \{ \varphi \}
\]

as long as

\[
B_s^{-\varphi} \neq \varphi.
\]

But if

\[
B_s^{-\varphi} = B_s - \{ \varphi \} \models \varphi
\]

then

\[
B_s^{-\varphi} = B_s - \{ \varphi_1 \ldots \varphi_n \}
\]

where \( \varphi_1 \ldots \varphi_n \) is a minimal set of formulas such that

\[
\varphi_1 \ldots \varphi_n \models \varphi
\]

and

\[
B_s - \{ \varphi_1 \ldots \varphi_n \} \neq \varphi
\]

The notation \( B_s^{-\varphi} \) is needed to capture the circumstances under which a belief is irrelevant to the reasons that a revision needs to occur. If it is the case that

\[
\text{Believes}(\Sigma_\alpha \land \Pi_\alpha \rightarrow \text{FALSE}, s)
\]

then revision needs to occur. But if

\[
B_s^{-P} \cup \{ \Sigma_\alpha \} \cup \{ \Pi_\alpha \} \models \text{FALSE}
\]

then the belief in \( P \) is not relevant to the causes of the revision and therefore should continue to be believed after revision has taken place.

The following two theorems correspond to the two theorems with the same name in [SL03]. The difference here is that the two cases of revision and update need to be distinguished. If it is a case of update, then both lack of belief and belief in a literal \( P \) persists as long as the effect of the action is not to change \( P \). But if we have a case of revision, then not only must it be the case that the action have no effect on \( P \), but it must also be the case that \( P \) must not be a cause of the contradiction.

**Theorem 2 (Default Persistence of Ignorance)** For all literals \( P \), an action \( \alpha \) and a situation \( s \), if \( \neg \text{Believes}(P, s) \) holds and the axiomatization entails

\[
\forall s \ \text{Holds}(P, s) \equiv \text{Holds}(P, \text{DO}(\alpha, s))
\]

and if

\[
\neg \text{Believes}(\Sigma_\alpha \land \Pi_\alpha \rightarrow \text{FALSE}, s)
\]

holds then

\[
\neg \text{Believes}(P, \text{DO}(\alpha, s))
\]

holds as well. Otherwise, if

\[
\text{Believes}(\Sigma_\alpha \land \Pi_\alpha \rightarrow \text{FALSE}, s)
\]

and

\[
B_s^{-P} \cup \{ \Sigma_\alpha \} \cup \{ \Pi_\alpha \} \models \text{FALSE}
\]

then

\[
\neg \text{Believes}(P, \text{DO}(\alpha, s))
\]

holds as well.

**Theorem 3 (Memory)** For all literals \( P \) and situations \( s \), if \( \text{Believes}(P, s) \) holds then \( \text{Believes}(P, \text{DO}(\alpha, s)) \) holds as long as the axiomatization entails

\[
\forall s \ \text{Holds}(P, s) \equiv \text{Holds}(P, \text{DO}(\alpha, s))
\]

and it is not the case that

\[
\text{Believes}(\Sigma_\alpha \land \Pi_\alpha \rightarrow \text{FALSE}, s)).
\]

If

\[
\text{Believes}(\Sigma_\alpha \land \Pi_\alpha \rightarrow \text{FALSE}, s))
\]

then \( \text{Believes}(P, \text{DO}(\alpha, s)) \) holds as long as and

\[
B_s^{-P} \cup \{ \Sigma_\alpha \} \cup \{ \Pi_\alpha \} \models \text{FALSE}
\]

holds.

Note that if the agent begins believing \( R \), \( P \rightarrow Q \), and \( \neg Q \), and then senses \( P \), revision will occur. The new beliefs will be \( R \), \( P \), and \( Q \). This is completely intuitive and the agent continues to believe \( R \) since it is irrelevant to the reasons that revision must occur. Similarly, the agent begins without believing \( T \) or \( \neg T \). After revision has occurred, this state of non-belief is unchanged.
But the approach does lead to some unintuitive results in the case of iterative revision with faulty sensors. For example, assume the agent starts out believing $P \rightarrow Q$, and $\neg P \rightarrow R$. But the agent does not believe $P$ or $\neg P$. Then if $P$ is sensed as being true, the agent will correctly believe both $P$ and $Q$. But if the agent then senses $P$ and this time $P$ turns out to be false, the agent will believe $\neg P$ but will not believe $R$ since it has lost a belief in $\neg P \rightarrow R$, when the belief in $P$ was acquired.

7 Comparisons

Unlike [SL03; SPLL00], the approach presented here is unable to handle introspection. This feature is inherited from the framework of [Thi00]. On the other hand, unlike [SPLL00], the solution to the frame problem of [SL03; Thi00] is extended to the case of belief. Jin and Thielscher [JT04] extend the approach of [Thi00] to handle belief and belief revision, but without introspection. Both [JT04] and [SPLL00] utilize a numerical ranking of states/situations as a method of representing the relative believability of possible worlds. It may be the case that with the appropriate ranking, each of these methods could satisfy the theorems presented above. In this case, the work presented here can be seen as a general method of providing such a ranking. Additionally, it may be the case that the work described here can be augmented with a ranking of states to overcome the unintuitive results with regard to iterative revision with faulty sensors.

8 Summary and Future Work

This paper has presented a method for modeling agents with possibly false beliefs and belief producing actions in the situation calculus, while still preserving memory. Current and future work involves the extension of the work to consider knowledge of sentences in first-order logic, handling iterative revision with faulty sensors, the development of reasoning methods to work with this axiomatization, and the incorporation into an agent programming language such as GoLog or Flux.

Acknowledgements

This work was supported by the Knowledge Fusion Center of the Army Research Laboratory under contract number DAAD-03-2-0034 and by ARDA under the AQUAINT program. Thanks are due to the anonymous reviewers for NRAC who provided many useful suggestions. Additionally, helpful comments on an earlier draft were provided by Chitta Baral and Michael Gelfond.

References


Belief Change with Noisy Sensing and Introspection*

Steven Shapiro
Computer Science Inst.
University of Leipzig
Leipzig 04109
Germany
shapiro@informatik.uni-leipzig.de

Abstract

We model belief change due to noisy sensing, and belief introspection in the framework of the situation calculus. We give some properties of our axiomatization and show that it does not suffer from the problems with combining sensing, introspection, and plausibility update described in Shapiro et al. [2000].

1 Introduction

In this paper, we generalize the framework of Shapiro et al. [2000], where belief change due to sensing was combined with belief introspection in the situation calculus. In that framework, sensing was assumed to be infallible and the plausibilities of alternate situations (i.e., possible worlds) were fixed in the initial state, never to be updated. Here, we relax both assumptions. That is, we model noisy sensors whose readings may stray from reality and may return different values in subsequent readings. We also allow the plausibilities of situations to change over time, bringing the framework more in line with traditional models of belief change. We give some properties of our axiomatization and show that it does not suffer from the problems with combining sensing, introspection, and plausibility update described in Shapiro et al. In the next section, we present the situation calculus including the representation of beliefs, and Shapiro et al.’s framework. In Sec. 4, we present the formal details of our axiomatization of belief change. In Sec. 5, we present some properties of our axiomatization, and in Sec. 6, we conclude and discuss future work.

2 Situation Calculus

The basis of our framework for belief change is an action theory [Reiter, 2001] based on the situation calculus [McCarthy and Hayes, 1969], and extended to include a belief operator [Scherl and Levesque, 1993]. The situation calculus is a predicate calculus language for representing dynamically changing domains. A situation represents a snapshot of the domain. There is a set of initial situations corresponding to the ways the agent believes the domain might be initially. The actual initial state of the domain is represented by the distinguished initial situation constant, $S_0$. The term $do(a, s)$ denotes the unique situation that results from the agent performing action $a$ in situation $s$. Thus, the situations can be structured into a set of trees, where the root of each tree is an initial situation and the arcs are actions. The initial situations are defined as those situations that do not have a predecessor $Init(s) \equiv \neg\exists a, s' \cdot s = do(a, s')$.

Predicates and functions whose value may change from situation to situation (and whose last argument is a situation) are called fluents. For instance, we use the fluent $INR_1(s)$ to represent that the agent is in room $R_1$ in situation $s$. The effects of actions on fluents are defined using successor state axioms [Reiter, 2001], which provide a succinct representation for both effect axioms and frame axioms [McCarthy and Hayes, 1969]. For example, assume that there are only two rooms, $R_1$ and $R_2$, and that the action $\text{LEAVE}$ takes the agent from the current room to the other room. Then, the successor state axiom for $\text{INR}_1$ is:

$$\text{INR}_1(\text{do}(a, s)) \equiv ((\neg\text{INR}_1(s) \land a = \text{LEAVE}) \lor (\text{INR}_1(s) \land a \neq \text{LEAVE})).$$

This axiom asserts that the agent will be in $R_1$ after doing some action iff either the agent is in $R_2$ ($\neg\text{INR}_1(s)$) and leaves it or the agent is currently in $R_1$ and the action is anything other than leaving it.

Moore [1985] defined a possible-worlds semantics for a modal logic of knowledge in the situation calculus by treating situations as possible worlds. Scherl and Levesque [1993] adapted the semantics to the action theories of Reiter [2001]. The idea is to have an accessibility relation on situations, $B(s', s)$, which holds if in situation $s$, the situation $s'$ is considered possible by the agent. Note, the order of the arguments is reversed from the usual convention in modal logic.

Levesque [1996] introduced a predicate, $SF(a, s)$, to describe the result of performing the binary-valued sensing action $a$. $SF(a, s)$ holds iff the sensor associated with $a$ returns 1Here we assume that there is a single agent, however it would not be difficult to generalize the framework to handle multiple agents.

2We adopt the convention that unbound variables are universally quantified in the widest scope.
the sensing value 1 in situation \( s \). Each sensing action senses some property of the domain. The property sensed by an action is associated with the action using a guarded sensed fluent axiom [De Giacomo and Levesque, 1999]. For example, suppose that there are lights in \( R_1 \) and \( R_2 \) and that \( \text{LIGHT}_1(s) \) (\( \text{LIGHT}_2(s) \), resp.) holds if the light in \( R_1 \) (\( R_2 \), resp.) is on. Then:

\[
\begin{align*}
\text{INR}_1(s) & \supset (\text{SF}(\text{SENSE}_\text{LIGHT}, s) \equiv \text{LIGHT}_1(s)) \\
\neg \text{INR}_1(s) & \supset (\text{SF}(\text{SENSE}_\text{LIGHT}, s) \equiv \text{LIGHT}_2(s))
\end{align*}
\]

can be used to specify that the \( \text{SENSE}_\text{LIGHT} \) action senses whether the light in the room where the agent is currently located is on.

Shapiro et al. [2000] adapted Spohn’s ordinal conditional functions [Spohn, 1988; Darwiche and Pearl, 1997] to the situation calculus by introducing plausibilities over situations using a function \( pl(s) \) which returns a natural number representing plausibility of situation \( s \). The lower the number, the more plausible the situation is considered. The plausibilities were fixed in the initial situation and were not allowed to change, i.e., they used this successor state axiom for \( pl \):

\[
pl(\text{do}(a, s)) = pl(s).
\]

They adopted Scherl and Levesque’s [2003] successor state axiom for \( B \):

\[
B(s'', \text{do}(a, s)) \equiv \exists s'. B(s', s) \land s'' = \text{do}(a, s') \land (\text{SF}(a, s') \equiv \text{SF}(a, s)).
\]

The situations \( s'' \) that are \( B \)-related to \( \text{do}(a, s) \) are the ones that result from doing action \( a \) in a situation \( s' \), such that the sensor associated with action \( a \) has the same value in \( s' \) as it does in \( s \). In other words, after doing \( a \), the agent’s beliefs will be expanded to include what the value of the sensor associated with \( a \) is in \( s \). If \( a \) is a sensing action, the agent’s beliefs will also include the property associated with \( a \) in the guarded sensed fluent axiom for \( a \). If \( a \) is a physical action, then the agent’s beliefs will also include the effects of \( a \) as specified by the successor state axioms.

Shapiro et al. defined the beliefs of the agent to be the formula true in the most plausible accessible situations:

\[
\text{Bel}_S(\phi, s) \equiv \forall s''[(\forall s'. B(s', s) \land s'' = \text{do}(a, s') \land (\text{SF}(a, s') \equiv \text{SF}(a, s)))] \supset \phi[s'].
\]

Shapiro et al. thus modelled belief change with infallible sensors. If the agent senses a property \( \phi \) and \( \phi \) actually holds, then all the situations that satisfy \( \neg \phi \) and \( \phi \) will become inaccessible. For example, if the agent believes \( \neg \phi \) and senses \( \phi \), then all the most plausible, accessible situations will become inaccessible. A new set of accessible situations will become most plausible, all of which satisfy \( \phi \), yielding belief in \( \phi \).

However, Shapiro et al. did not allow for the possibility of the agent subsequently sensing \( \neg \phi \).

There are various ways of axiomatizing dynamic applications in the situation calculus. Here we adopt a simple form of the guarded action theories described by De Giacomo and Levesque [1999] consisting of: (1) successor state axioms for each fluent, and guarded sensed fluent axioms for each action, as discussed above; (2) unique names axioms for the actions, and domain-independent foundational axioms (we adopt the ones given by Levesque et al. [1998] which accommodate multiple initial situations, but we do not describe them further here); and (3) initial state axioms, which describe the initial state of the domain and the initial beliefs of the agent.

In what follows, we will use \( \Sigma \) to refer to a guarded action theory of this form.

3 Belief Change

Before formally defining a belief operator in this language, we briefly review the notion of belief change. Belief change, simply put, aims to study the manner in which an agent’s doxastic (belief) state should change when the agent is confronted by new information. In the literature, there is often a clear distinction between two forms of belief change: revision and update. Both forms can be characterized by an axiomatic approach (in terms of rationality postulates) or through various constructions (e.g., epistemic entrenchment, possible worlds, etc.). The AGM theory [Gärdenfors, 1988] is the prototypical example of belief revision while the KM framework [Katsuno and Mendelzon, 1991] is often identified with belief update.

Intuitively speaking, belief revision is appropriate for modeling static environments about which the agent has only partial and possibly incorrect information. New information is used to fill in gaps and correct errors, but the environment itself does not undergo change. Belief update, on the other hand, is intended for situations in which the environment itself is changing due to the performing of actions.

4 Belief Change with Noisy Sensors

Shapiro et al. [2000] modelled belief change due to sensing but it was assumed that the sensors were always accurate. This is quite a strong assumption which we will relax here. If sensing is exact, then the sensors will never be contradicted and so belief revision is limited to revising the agents initial beliefs. But once an initial belief is corrected, it will never change again. In this context it seems reasonable to have a fixed plausibility relation. However, if the sensors can return different results over time, this approach will not work because after sensing two contradicting values for the same formula, the agent will have contradictory beliefs (i.e., an empty accessibility relation).

To model noisy sensing, we add another distinguished predicate \( \text{SR}(a, s) \), which is similar to \( \text{SF} \) described previously. The idea is that while \( \text{SF}(a, s) \) describes the property of the world ideally sensed by action \( a \), the actual values returned by the sensor may not correspond exactly to the property described by \( \text{SF} \). So, we will use \( \text{SR}(a, s) \) to describe the
value actually returned by the sensor associated with action \( a \). Another way of describing \( SR \) is that it is the result of adding noise to the sensor described by \( SF \). How to specify \( SR \) is still an unresolved issue. We want \( SR \) to be related to \( SF \) but perhaps only related by a stochastic relation. This problem is reserved for future work.

As with Shapiro et al., we assume that the agent knows the history of actions it has taken. By that we mean the agent only considers a situation possible if it agrees with the history of actions in the actual situation.\(^5\) We further assume that the agent has privileged access to its sensors. That is, after the agent reads its sensor, it knows the value of the sensor and it remembers the sequence of sensor readings it has made to date. That is, in addition to knowing the history of actions that have occurred, the agent knows the history of sensor readings it has taken, and it only considers possible those situations that agree with the actual situation on the history of sensor readings.

## 5 Axiomatization

To model plausibilities that can change, we dispense with the \( pl \) predicate used by Shapiro et al. [2000], and instead add a plausibility to the accessibility relation. So, \( B(s', n, s) \) will denote that \( s' \) is considered a possible situation by the agent with plausibility \( n \) in situation \( s \). As before, the lower the plausibility level the more plausible the agent considers the situation to be. The beliefs of the agent are determined by the situations with plausibility \( 0 \):

\[
\text{Bel}(\phi, s) \equiv \forall s'. B(s', 0, s) \supset \phi[s']
\]

As previously mentioned, we have two further distinguished predicates: \( SF(a, s) \) and \( SR(a, s) \), both of which take an action and a situation as arguments. The former holds if the property ideally sensed by sensing action \( a \) holds in situation \( s \), and the latter holds if the sensor associated with \( a \) actually returns the value 1 in \( s \). We adopt the convention that if \( A \) is a non-sensing action, then \( \forall s.SF(A, s) \land SR(A, s) \) holds.

The dynamics of the agent’s beliefs are formalized by the successor state axiom for \( B \):

**Axiom 1**

\[
\begin{align*}
B(s'', n'', \text{do}(a, s)) & \equiv \\
\exists s', n'. B(s', n', s) \land s'' = \text{do}(a, s') \land \\
(SR(a, s') \equiv SR(a, s)) \land \text{Update}(n'', n', a, s'),
\end{align*}
\]

where \( \text{Update}(n'', n', a, s') \) (defined below) holds if \( n'' \) is the updated plausibility level due to action \( a \) for situation \( s' \) whose plausibility with respect to \( s \) is \( n' \).\(^6\) In other words, \( s'' \) will be accessible from \( \text{do}(a, s) \) with plausibility \( n'' \), if there exist \( s' \) and \( n' \) such that \( s' \) was accessible from \( s \) with plausibility \( n', s' \) and \( s \) agree on the value of the sensor associated with \( a \), and \( n'' \) is the result of updating the plausibility of \( s' \)

\( ^5\)A treatment of exogenous actions that are hidden from the agent was given by Shapiro and Pagnucco [2004].

\( ^6\)\( \text{Update} \) could be a function, however we found it more convenient to formulate it as a relation.

with respect to \( s \) due to \( a \). Note that situations that disagree with \( s \) on the value of the sensor associated with \( a \), (and those whose last action is not \( a \)) are discarded altogether from the accessibility relation. This means that the agent will never come to believe that it was mistaken about its sensor readings (or about the history of action occurrences).

We update the plausibilities as follows. We say a situation’s (\( s \)) sensor reading is correct with respect to the sensor associated with \( a \), if its \( SF \) and \( SR \) values agree, i.e., \( SR(a, s) = SF(a, s) \). Those situations whose sensor readings are correct will have their plausibility levels decreased (i.e., they will become more plausible) and the others will have their plausibility levels increased. For concreteness, we will use Darwiche and Pearl’s [1997] update function, but others are possible.

\[
\begin{align*}
\text{Correct}(a, s) & \equiv SR(a, s') \equiv SF(a, s') \\
\text{Good}(s', n', a, s) & \equiv B(s', n', s) \land (SR(a, s') \equiv SR(a, s)) \land \text{Correct}(a, s') \\
\text{Min}(n, a, s, n') & \equiv (\exists s^* \text{Good}(s^*, n, a, s)) \land \\
& \forall s', n'. \text{Good}(s', n', a, s) \supset n' \geq n \\
\text{Update}(n'', n', a, s', s) & \equiv \\
& (\text{Correct}(a, s') \supset \exists n^*. \text{Min}(n^*, a, s) \land n'' = n' - n^*) \land \\
& (\neg \text{Correct}(a, s') \supset n'' = n' + 1)
\end{align*}
\]

In other words, the situations whose sensor readings are incorrect have their plausibilities increased by 1. The situations whose readings are correct are updated by subtracting the \( \text{Min} \) value, which is the lowest plausibility among the accessible situations that agree with the actual situation on the sensor reading and the sensor reading is correct. The result is that the agent believes that its sensor reading is correct, since this will hold in all the 0-plausibility situations.

Following Shapiro et al., we want \( B(s', n, s) \) to hold only if \( s \) and \( s' \) have the same histories. This means that the agent knows what actions have occurred. To enforce this, we need the following axiom which says that the situations accessible from an initial situation are also initial.

**Axiom 2** \( \text{Init}(s) \land B(s', n, s) \supset \text{Init}(s') \).

We have \( B \) as a relation so that we can exclude certain situations altogether. We can think of these situations as completely implausible. However, for the situations that are assigned some plausibility, we want their plausibility to be unique.

**Axiom 3** \( \text{Init}(s) \land B(s', n, s) \land B(s', n', s) \supset n = n' \).

To ensure that the agent has positive and negative introspection, we need to impose a constraint on the situations accessible from initial situations. As is well known (see, e.g.,
Figure 1: Introspection, exact sensing, and updating plausibilities clash

Fagin et al. (1995), to get positive and negative introspection in contexts without plausibilities, it suffices for the accessibility relation to be transitive and Euclidean. Our constraint is a generalization of the combination of transitivity and Euclideanness that takes plausibilities into account. To get positive and negative introspection, we only need the accessibility relation to be transitive and Euclidean over situations with plausibility 0. However, since we are dealing with a dynamic framework, situations with higher plausibility levels could later have plausibility 0, therefore we enforce these constraints over all plausibility levels.

**Axiom 4**

\[
\text{Init}(s) \subseteq \left( B(s', n, s) \cup \forall s'' \cdot n''. B(s'', n'', s') \equiv B(s'', n'', s) \right).
\]

In other words, for initial \( s \), if \( s' \) is accessible from \( s \) with some plausibility, then \( s \) and \( s' \) have the same accessible situations with the same plausibilities.

Shapiro et al. described a conflict in their framework between preserving this constraint and updating plausibilities which is illustrated in Fig. 1. In this example, \( S_2 \) is accessible from \( S \) with some unspecified plausibility, and \( S_1 \) is accessible from both \( S \) and \( S_2 \) with plausibility \( n \). Note that this example satisfies the constraint described in Axiom 4 (if we assume the situations are all initial). Now recall that for Shapiro et al. sensing was assumed to be accurate. Therefore, if the agent senses \( \phi \), the plausibility level of \( S_1 \) with respect to \( S \) should increase because they disagree on the value of \( \phi \), whereas the plausibility level of \( S_1 \) with respect to \( S_2 \) should decrease because they agree on the value of \( \phi \). Therefore, the generalization of the constraint described in Axiom 4 that omits the condition that \( s \) be initial will not be satisfied after sensing \( \phi \). This means that the agent may loose full introspection.

The problem here is that in \( S \), the sensor says that \( \phi \) holds, while in \( S_2 \), the sensor says that \( \phi \) does not hold. So, loosely speaking, the agent in \( S_1 \) is being told to revise its beliefs with \( \phi \) by \( S \) and with \( \neg \phi \) by \( S_2 \). In our framework, this problem is avoided because all the situations that disagree with \( S \) on the value of the sensor will be dropped from the accessibility relation. In effect, the beliefs of the agent in all the surviving accessible situation are revised by the same formula. In other words, to avoid this problem, we (and Shapiro et al.) had to model agents that have privileged access to their sensors, i.e., they know the results of sensing. In the next section, we give a theorem which says that we have indeed avoided this problem.

Note that this problem only arises when beliefs are changed due to sensing (and the agent is introspective). When an agent senses \( \phi \), it is told whether \( \phi \) holds. In the traditional belief change setting, the agent is informed that \( \phi \) holds. The subtle, but crucial, difference is that in the former case, the content of the belief-producing action depends on the actual situation, but not in the latter. As we stated earlier, the problem illustrated in Fig. 1 is that in \( S \), the sensor says that \( \phi \) holds, while in \( S_2 \), the sensor says that \( \phi \) does not hold. If we were to model informing instead of sensing using the action INFORM(\( \phi \)), the value of \( \phi \) in \( S \) and \( S_2 \) would be irrelevant. In both situations, the agents beliefs would be revised with \( \phi \). While there have been previous approaches to belief revision with unreliable observations, e.g., [Aucher, 2005; Bacchus et al., 1999; Boutilier et al., 1998; Laverny and Lang, 2004] almost all of them use informing as the belief-producing action rather than sensing. Bacchus et al. (1999) model sensing (also in the framework of the situation calculus) as the nondeterministic choice of inform actions, one for each possible value returned by the sensor, but they do not address introspection. We think that there may be a problem modelling sensing this way, in the presence of introspection about future beliefs. If an agent believes \( \phi \) then it should also believe that it will believe \( \phi \) after sensing \( \phi \). This would not seem to hold in an approach like Bacchus et al.’s. Furthermore, we think it is more natural to model sensing as a primitive action rather than a nondeterministic choice of actions.

One issue that remains to be resolved is how to ensure that there is always at least one accessible situation. Since we are modelling noisy sensing, the agent’s sensors could say that \( \phi \) holds and later say that \( \phi \) does not hold. How do we then prevent the agent’s beliefs from lapsing into inconsistency? We need to ensure that regardless of the history of sensing results, for each action \( a \), there is always an accessible situation (but not necessarily a most plausible one) that agrees with actual situation on the value returned by the sensor associated with \( a \), and that value is correct, i.e.: \[
\forall a, s \exists s', n. B(s', n, s) \land (SR(a, s') \equiv SR(a, s)) \land \text{Correct}(a, s').
\]

We believe we can achieve this using an axiom similar to the one given by Lakemeyer and Levesque [1998], and we will investigate this in future work.

**6 Properties**

In this section, we give some properties of our axiomatization of belief change and show that it does not suffer from the problem discussed by Shapiro et al. Let \( \Sigma \) be the foundational axioms together with the axioms of the previous section. First, we can show that the constraints imposed on the initial state given in Axioms 3 and 4 are preserved over all sequences of actions.

**Theorem 1**

\[
\Sigma \models \forall n, n', s'. B(s', n, s) \land B(s', n', s) \supset n = n',
\]

\[
\Sigma \models \forall n, n', s, s'. B(s', n, s) \supset \forall s''. n'', B(s'', n'', s') \equiv B(s'', n'', s).
\]

The latter property ensures that the agent always has full introspection, and shows that we do not suffer from the problem of combining sensing, introspection, and updating plausibilities discussed in the previous section.
Corollary 2

\[ \Sigma \models \forall s. Bel(\phi, s) \supset Bel(Bel(\phi), s), \]
\[ \Sigma \models \forall s. \neg Bel(\phi, s) \supset Bel(\neg Bel(\phi), s). \]

Shapiro et al. discussed a possible solution to their problem with updating plausibilities by setting the plausibility levels from all accessible situations to be the same as they are in the actual situation. However, they showed that this solution was unsatisfactory by giving an example using this scheme that entailed a counterintuitive property, namely, that the agent believes \( \phi \), but thinks that after sensing \( \phi \), it will believe \( \neg \phi \). We can show that it is not possible to construct such an example in our framework that is reasonable. In particular, we show that in any such example, the agent believes that either its sensor is incorrect or that its beliefs will be inconsistent after sensing \( \phi \). The second alternative is clearly not reasonable. We would not want to model an agent that believes \( \phi \) but also believes that after sensing \( \phi \), its beliefs will become inconsistent. The first alternative does not make sense either because we are modelling agents that revise their beliefs according to what their sensors tell them. If the agent were to believe that its sensor is not correct, then it would not make sense to revise its beliefs according to what the sensor said. So, while the agent might be aware that its sensors are not always correct, we want to avoid situations where the agent actually believes that its sensor will return the wrong value. Accordingly, the next theorem says that if the agent believes \( \phi \) and it thinks that it will believe \( \neg \phi \) after sensing \( \phi \), and an action \( a \) is a sensing action for \( \phi \) that does not change the value of \( \phi \) if it holds initially, then the agent believes that either its sensor is incorrect or that its beliefs will be inconsistent after sensing \( \phi \).

Theorem 3

\[ \Sigma \models \forall a, s. Bel(\phi, s) \land Bel(Bel(\neg \phi, do(a, Now)), s) \land \]
\[ (\forall s'.SF(a, s') \equiv \phi(s')) \land \]
\[ (\forall s'.\phi(s') \supset \phi(do(a, s'))) \supset \]
\[ Bel(\neg Correct(a, Now) \lor \]
\[ Bel(\neg Correct(a, Now)))\land \]
\[ Bel(\neg Correct(a, Now)), s) \]

Next, we show that the agent will revise its beliefs appropriately. If an action \( a \) (ideally) senses a property \( \phi \), and the sensor indicates that \( \phi \) holds, then after sensing, the agent will believe that \( \phi \) held before the sensing occurred. We first define what it means for \( \phi \) to hold in the previous situation:

\[ \text{Prev}(\phi, s) \equiv \exists a, s'. a = do(a, s') \land \phi(s'). \]

Theorem 4

\[ \Sigma \models \forall a, s. (\forall s'.SF(a, s') \equiv \phi(s')) \land SR(a, s) \supset \]
\[ Bel(\text{Prev}(\phi), do(a, s))]. \]

If the agent also believes that \( a \) does change the value of \( \phi \), then the agent will believe \( \phi \) after doing \( a \).

Since the basis of our framework is a theory of action, belief updates are handled naturally as resulting from physical actions. We show that (as with Shapiro et al.) belief updates are handled appropriately. If \( a \) is a physical action (i.e., \( SF \) and \( SR \) are identically true) and situation \( s \) has at least one accessible situation with 0-plausibility, and the agent believes that \( a \) causes \( \phi \) to hold if \( \phi \) holds initially, then the agent will believe that \( \phi \) holds after doing \( a \) in \( s \), if it believes that \( \phi \) holds in \( s \).

Theorem 5

\[ \Sigma \models \forall a, s. (\exists s'. B(s', 0, s) \land \]
\[ (\forall s'.SF(a, s')) \land (\forall s'.SR(a, s')) \land \]
\[ Bel((\phi(\text{Now}) \supset \phi'(do(a, \text{Now}))), s) \land \]
\[ Bel(\phi, do(a, s)) \]

Finally, we can show that Shapiro et al.’s framework is a special case of ours. If we assume that sensing is always accurate, and for every action \( a \), every situation has an accessible situation that agrees with it on the value of the sensor associated with \( a \), then Shapiro et al.’s axioms for \( B \) and \( pl \) combined and translated into our notation follow from our axioms.

Theorem 6

\[ \Sigma \models (\forall a, s. \text{Correct}(a, s)) \land \]
\[ (\forall a, \exists s'. B(s', 0, s) \land \text{SR}(a, s') \equiv S(R(a, s'))) \supset \]
\[ \forall a, s, s', \phi(s', s'' \equiv \phi(a, s)) \equiv \]
\[ \exists s'. B(s', s''), s \land s'' = do(a, s') \land \]
\[ (\text{SF}(a, s') \equiv \text{SF}(a, s)). \]

7 Conclusions and Future Work

In this paper, we introduced a framework for modelling belief change as a result of noisy sensing in the situation calculus, where the agent has full introspection of its beliefs. Our framework allows the updating of plausibilities of situations, and we showed that we resolved the difficulty with combining all these elements discussed by Shapiro et al. [2000], and to achieve this, we had to endow the agent with infallible knowledge of the results of its sensing. As previously mentioned, there are some issues that are as yet unresolved. One is how to specify the \( SR \) predicate. Another is how to ensure that there are enough situations with the right properties to prevent the agent’s beliefs from lapsing into inconsistency. We would also like to investigate the extent to which our framework satisfies the standard belief change postulates [Darwiche and Pearl, 1997; Gardenfors, 1988; Katsuno and Mendelzon, 1991]. Lastly, we would like to extend the framework to handle unreliable physical actions, i.e., physical actions whose outcomes may be different from those expected by the agent.

References


The Wumpus World in INDIGOLOG: A Preliminary Report

Sebastian Sardina and Stavros Vassos
University of Toronto, Toronto, ON, M5S 3G4, Canada.
{ssardina, stavros}@cs.toronto.edu

Abstract

This paper describes an implementation of the Wumpus World [Russell and Norving, 2003] in INDIGOLOG with the objective of showing the applicability of this interleaved agent programming language for modeling agent behavior in realistic domains. We briefly go over the INDIGOLOG architecture, explain how we can reason about the Wumpus World domain, and show how to express agent behavior using high-level agent programs. Finally, we discuss initial empirical results obtained as well as challenging issues to be resolved.

1 Introduction

There has been extensive work on logical formalisms for dynamic domains. Action theories in the literature address a variety of issues ranging from the specification of actions’ effects and non-effects (i.e., the so-called frame problem [McCarthy and Hayes, 1969]) and the qualification problem, to more sophisticated topics such as the ramification problem, knowledge and sensing, incomplete information, concurrent action, continuous time and nondeterministic effects, to name a few. Recently, much research into reasoning about actions has been devoted to the design and implementation of agent languages and systems for Cognitive Robotics which are often built on top of existing rich formalisms of action and change. An agent is assumed to be equipped with a formal theory of the world and a high-level program describing its behavior up to some degree. This is indeed the view taken in INDIGOLOG, the last version of the University of Toronto GOLOG-like family of agent programming languages. INDIGOLOG provides a formal account of perception, deliberation, and execution within the language of the situation calculus. INDIGOLOG is implemented in PROLOG and has been used in real robotics platforms such as the LEGO MINDSTORM and the ER1 EVOLUTION robots.

In this paper, we show how to use INDIGOLOG to model the behavior of an agent living in the Wumpus World (see [Russell and Norving, 2003, Chapter 7]), a convincing and challenging abstraction of an incompletely known dynamic environment for logically reasoning agents. To that end, we explain how to practically reason in the Wumpus World by using a special kind of situation calculus-based action theories which soundly approximate incomplete knowledge by representing the dynamics of the possible values for the fluents. In addition, we explain how the behavior of an intelligent agent can be modeled with a high-level agent program that is intended to be executed incrementally. We provide empirical results which show the feasibility of our approach for this interesting scenario.

2 INDIGOLOG: An Interleaved Agent Architecture

The agent programming language to be used for modeling an agent that acts and reasons in the Wumpus World is INDIGOLOG, the most recent situation calculus based agent language in the GOLOG family. The situation calculus is a second-order language specifically designed for representing dynamically changing worlds [McCarthy and Hayes, 1969; Reiter, 2001]. All changes to the world are the result of named actions such as moveForward and pickup(x). A possible world history, which is simply a sequence of actions, is represented by a first-order term called a situation. The constant S0 is used to denote the initial situation and a distinguished binary function symbol do(a, s) is used to denote the successor situation to s resulting from performing action a. The features of the world are represented with (functional) fluents, functions denoted with a situation term as their last argument and whose values vary from situation to situation. There is also a special predicate Poss(a, s) used to state that action a is executable in situation s. In the presence of sensing actions, a special function SR(a, s) is used to state the sensing result obtained from executing action a in situation s. Non-sensing actions are assumed to always return 1 as their sensing outcomes. Also, to talk about both the actions and their sensing results we use the notion of a history, a sequence of pairs (a, µ) where a is a primitive action and µ is the corresponding sensing outcome.

Within this language, one can specify action theories that describe how the world changes as the result of the available actions in a principled and modular way (e.g., basic action theories [Reiter, 2001]). Using the theory, the agent can query the state of the world at each possible world history by solving the so-called projection task: given a sequence of actions together with their corresponding sensing outcomes, and a
formula $\phi(s)$ about the world, determine whether $\phi(s)$ is true in the situation resulting from performing these actions.

On top of these action theories, logic-based programming languages can be defined, which, in addition to the primitive actions of the situation calculus, allow the definition of complex actions. GOLOG [Levesque et al., 1997], the first situation calculus agent language, offers all the control structures known from conventional programming languages (e.g., sequence, iteration, conditional, etc.) plus some nondeterministic constructs. It is due to these last control structures that programs do not stand for complete solutions, but only for sketches of them whose gaps have to be filled later, usually at execution time. CONGOLOG [De Giacomo et al., 2000] extends GOLOG to accommodate concurrency and interrupts in order to accommodate the specification of reactive agent behavior. A summary of the constructs available follows:

- $\alpha$, primitive action
- $\phi?$, wait or test for a condition
- $\delta_1; \delta_2$, sequence
- $\delta_1 | \delta_2$, nondeterministic branch
- $\pi \cdot \delta(x)$, nondeterministic choice of argument
- $\text{do } \delta \text{ endIf}$, conditional
- $\text{while } \phi \text{ do } \delta \text{ endWhile}$, while loop
- $\delta_1 || \delta_2$, concurrency with equal priority
- $\delta_1 \triangleright \delta_2$, concurrency with $\delta_1$ at a higher priority
- $\delta^\dagger$, concurrent iteration
- $\langle \vec{x} : \phi(\vec{x}) \rightarrow \delta(\vec{x}) \rangle$, interrupt
- $p(\vec{\theta})$, procedure call

Note the presence of nondeterministic constructs, such as ($\delta_1 | \delta_2$), which nondeterministically chooses between programs $\delta_1$ and $\delta_2$, $\pi \cdot \delta(x)$, which nondeterministically picks a binding for the variable $x$ and performs the program $\delta(x)$ for this binding of $x$, and $\delta^\dagger$, which performs $\delta$ zero or more times. To deal with concurrency two constructs are provided: ($\delta_1 || \delta_2$) expresses the concurrent execution (interpreted as interleaving) of programs $\delta_1$ and $\delta_2$: ($\delta_1 \triangleright \delta_2$) expresses the concurrent execution of $\delta_1$ and $\delta_2$ with $\delta_1$ having higher priority. Finally, for an interrupt ($\langle \vec{x} : \phi(\vec{x}) \rightarrow \delta(\vec{x}) \rangle$), program $\delta(\vec{t})$ is executed whenever condition $\phi(\vec{t})$ holds (see [De Giacomo et al., 2000] for further details.)

Finding a legal execution of high-level programs is at the core of the whole approach. Originally, GOLOG and CONGOLOG programs were intended to be executed offline, that is, a complete solution was obtained before committing even to the first action. In contrast, INDI GOLOG, the next extension in the GOLOG-like family of languages, provides a formal logic-based account of interleaved planning, sensing, and action [Kowalski, 1995] by executing programs online. Roughly speaking, an incremental or online execution of a program finds a next possible action, executes it in the real world, obtains sensing information afterwards, and repeats the cycle until the program is finished. The semantics of INDI GOLOG is specified in terms of single-steps, using two predicates $Trans$ and $Final$ [De Giacomo et al., 2000]: $Final(\delta, s)$ holds if program $\delta$ may legally terminate in situation $s$; $Trans(\delta, s, \delta', s')$ holds if one step of program $\delta$ in situation $s$ leads to situation $s'$ with $\delta'$ remaining to be executed. It is important to point out also that the execution of a program strongly relies on the projection task. For example, to execute a test for condition $\phi?$ one needs the project $\phi$ at the current history w.r.t. the underlying theory of action.

The fact that actions are quickly executed without much deliberation and sensing information is gathered after each step makes the approach realistic for dynamic and changing environments. However, an online execution is deterministic in the sense that there is no provision for backtracking once an action has been selected. Yet there may be two actions $A_1$ and $A_2$ for which $Trans$ holds and yet only $A_2$ may lead ultimately to a legal successful termination. To deal with this form of non-determinism, INDI GOLOG contains a search operator $\Sigma$ to allow the programmer to specify when lookahead should be performed: executing $\Sigma(\delta)$ means executing $\delta$ in such a way that at each step there is a sequence of further steps leading to a legal termination. Unlike a purely offline execution, however, operator $\Sigma$ allows us to control the amount of lookahead to use at each step.

### 2.1 An Incremental Interpreter

A logic-programming implementation of INDI GOLOG has been developed to allow the incremental execution of high-level GOLOG-like programs [Sardina, 2004]. This system is fully programmed in PROLOG and has been used to control the LEGO MINDSTORM robot, the ER1 EVOLUTION robot and other soft-bot agents. The implementation provides an incremental interpreter of programs as well as all the framework to deal with the real execution of these programs in real platforms (e.g., real execution of actions, sensing outcome readings, exogenous actions, etc.)

The architecture, when applied to the Wumpus World scenario, can be divided into the following four parts: (i) the device manager software interfacing with the real world; (ii) the evaluation of test conditions; (iii) the implementation of $Trans$ and $Final$; (iv) and the main loop. The device manager for the Wumpus World is the code responsible for simulating a real-world Wumpus World environment. It provides an interface for the execution of actions (e.g., moveFwd, smell, etc.), the retrieval of sensing outcomes, and the occurrence of exogenous events (e.g., scream). In our case, the world configuration will be also displayed using a Java applet. We shall now briefly go over the other three parts.

### The evaluation of formulas

In order to reason about the world, we need to be able to specify the domain and be able to project formulas w.r.t. evolutions of the system, that is, to evaluate the truth of formulas at arbitrary histories. We use an extension of the classical formula evaluator used for GOLOG that is able to handle some kind of incomplete knowledge. To that end, the evaluator deals with the so-called possible values that (functional) fluents can take at certain history; we say that the fluent is known at $h$ only when it has only one possible value at $h$. For a detailed description and semantics of this type of knowledge-based theories we refer to [Vassos et al., 2005; Levesque, 2005].

We assume then that users provide definitions for each of the following predicates for fluent $f$, action $a$, sensing result $r$, formula $w$, and arbitrary value $v$:

- $\text{fluent}(f)$, $f$ is a ground fluent;
• action(a),  a is a ground action;
• init(f,v),  initially, v is a possible value for f;
• poss(a,w),  it is possible to execute action a provided formula w is known to be true;
• causes(a,f,v,w),  action a affects the value of f: when w is possibly true, v is a possible value for f;
• settles(a,r,f,v,w),  action a with result r provides sensing information about f: when w is known to be true, v is the only possible value for f;
• rejects(a,r,f,v,w),  action a with result r provides sensing information about f: when w is known to be true, v is not a possible value for f.

Formulas are represented in PROLOG using the obvious names for the logical operators and with all situations suppressed; histories are represented by lists of the form o(a,r) where a represents an action and r a sensing result. We will not go over how formulas are recursively evaluated, but just note that the procedure is implemented using the following four predicates: (i) kTrue(w,h) is the main and top-level predicate and it tests if the formula w is known to be true in history h; (ii) mTrue(w,h) is used to test if w is possibly true at h; (iii) subf(w1,w2,h) holds when w2 is the result of replacing each fluent in w1 by one of its possible values in history h; and (iv) mval(f,v,h) calculates the possible values v for fluent f in history h and is implemented as follows:

\[
\text{mval}(\text{f}, \text{v}, \lambda) \leftarrow \text{init}(\text{f}, \text{v}).
\]

\[
\text{mval}(\text{f}, \text{v}, [o(A, R) | H]) \leftarrow \text{causes}(A, F, \lambda, \lambda), !, \text{causes}(A, F, V, W), \text{mTrue}(W, H).
\]

\[
\text{mval}(\text{f}, \text{v}, [o(A, R) | H]) \leftarrow \text{settles}(A, R, F, V, W), \text{kTrue}(W, H), !, V = V1.
\]

\[
\text{mval}(\text{f}, \text{v}, [o(A, R) | H]) \leftarrow \text{mval}(\text{f}, \text{v}, H), \text{not}(\text{rejects}(A, R, F, V, W), \text{kTrue}(W, H)).
\]

So for the empty history, we use the initial possible values. Otherwise, for histories whose last action is a with result r, if f is changed by a with result r, we return any value v for which the condition w is possibly true; if a with result r senses the value of f, we return the value v for which the condition is known; otherwise, we return any value v that was a possible value in the previous history h and that is not rejected by action a with result r. This provides a solution to the frame problem: if a is an action that does not affect or sense for fluent f, then the possible values for f after doing a are the same as before.

The implementation of Trans and Final and the main loop
Clauses for Trans and Final are needed for each of the program constructs. The important point to make here is that whenever a formula needs to be evaluated, kTrue/2 is used. So, for example, these are the corresponding clauses for sequence, tests, nondeterministic choice of programs, and primitive actions:

\[
\text{final}(\text{nindet}(E1, E2), H) \leftarrow \text{final}(E1, H); \text{final}(E2, H).
\]

\[
\text{trans}([\text{nindet}(E1, E2), H, E, H1]) \leftarrow \text{trans}(E1, H, E, H1).
\]

\[
\text{trans}(\text{nindet}(E1, E2), H, E, H1) \leftarrow \text{trans}(E2, H, E, H1).
\]

The top part of the interpreter deals with the execution of actions in the world. It makes use of Trans/4 and Final/2 to determine the next action to perform and to end the execution.

\[
\text{indigo}(E, H) \leftarrow \text{handle_rolling}(H), !, \text{indigo}(E, []).\]

\[
\text{indigo}(E, H) \leftarrow \text{exog occur}(A), !, \text{indigo}(E, [A | H]).
\]

\[
\text{indigo}(E, H) \leftarrow \text{final}(E, H), !.
\]

\[
\text{indigo}(E, H) \leftarrow \text{trans}(E, H, E1, H), !, \text{indigo}(E1, H).
\]

\[
\text{indigo}(E, H) \leftarrow \text{trans}(E, H, E1, [A | H]), \text{execute}(A, H, S), !, \text{indigo}(E1, [o(A, S) | H]).\]

In the first clause, predicate handle_rolling/1 checks whether the current history H must be rolled forward (for example, if its length has exceeded some threshold). If it does, handle_rolling/1 performs the progression of the database and the execution continues with the empty history. In the second clause, the interpreter checks whether some exogenous action has occurred. In that case, the action in question is added to the current history and execution continues. Next, the third clause ends the execution whenever the current configuration is provably terminating. The fourth clause handles the case of transition steps that involve no action. Finally, the last clause performs an action transition step: predicate execute(A,H,S) is the interface to the real world and is responsible for the actual execution of action A in history H. Variable S is bound to the corresponding sensing outcome obtained from the environment after performing the action, and the execution program continues correspondingly. In our case, predicate execute/3 will interface with the Wumpus World device manager simulator through TCP/IP sockets.

3 Reasoning in The Wumpus World

The Wumpus World is a well-known example for reasoning and acting with incomplete knowledge (see [Russell and Norving, 2003, Chapter 7]). According to the scenario, the agent enters a dungeon in which each location may contain the Wumpus (a deadly monster), a bottomless pit, or a piece of gold. The agent moves around looking for gold and avoiding death caused by moving into the location of a pit or the Wumpus. The agent has an arrow which she can throw as an attempt to kill the Wumpus. Also, the agent can sense the world to get clues about the extent of the dungeon, as well as the location of pits, gold pieces, and the Wumpus.

Using the functionality described in Section 2.1, we construct the domain description $\Pi^W$ which captures the agent’s knowledge about the world. $\Pi^W$ follows closely the definitions in [Russell and Norving, 2003] except for the fact that the agent assumes a fixed predefined size for the dungeon.
The world is organized as an 8 x 8 rectangular grid, where $g(1,1)$ is the lower-left corner and $g(1,8)$ is the upper-left one. Predicate $\text{loc}(L)$ holds if $L$ is a valid grid location and $\text{dir}(D)$ holds if $D$ is one of the four directions up, down, left and right. The geometry of the grid is captured by predicate $\text{adj}(2)$, which holds if the arguments represent two adjacent locations, and one binary predicate for each direction such as $\text{adj}(L,L2)$, which holds if $L2$ is a location lying one step up of $L$.

$\Pi_W$ includes the following fluents, each of which captures the possible values for a dynamic element of the domain. $\text{locA}$, $\text{dirA}$ and $\text{hasArrow}$ represent the location of the agent, the direction that she is facing and whether she has the arrow. $\text{noGold}$ is used to keep track of the number of gold pieces gathered by the agent. $\text{isGold}(L)$ and $\text{isPit}(L)$ capture whether there is a gold piece or a pit at location $L$ and $\text{isVisited}(L)$ captures whether the location has already been explored. $\text{locW}$ and $\text{aliveW}$ capture the location of the Wumpus and whether it is alive. Finally, $\text{inDungeon}$ is used to capture whether the agent is in the dungeon. A few representative fluent/1 clauses follow.

fluent($\text{locA}$).
fluent($\text{isPit}(L)$): $\text{loc}(L)$.
fluent($\text{dirA}$).
fluent($\text{isGold}(L)$): $\text{loc}(L)$.

Note that all fluents are functional; those that capture propositions will be defined so that only true and false may be a possible value for them. Initially, the agent is in the dungeon at location $g(1,1)$ facing to the right. She possesses the arrow, but no gold pieces. There is only one possible value for the corresponding fluents and so there is complete information about the state of the agent in the empty history. On the contrary, a possible value for the location of the Wumpus is any valid grid location apart from $g(1,1)$ and similarly, any location other than $g(1,1)$ can possibly contain a pit or a gold piece. Also, initially the agent knows that the Wumpus is alive and that only $g(1,1)$ has been explored. The init/2 clauses for the fluents follow. We only omit the init/2 clauses for $\text{isGold}(L)$ which are identical to the ones for $\text{isPit}(L)$.

init($\text{inDungeon}$,true).
init($\text{locA},g(1,1)$).
init($\text{dirA}$,right).
init($\text{hasArrow}$,true).
init($\text{noGold}$,0).
init($\text{aliveW}$,true).
init($\text{locW},L$): $\text{loc}(L)$, not $L=g(1,1)$.
init($\text{isPit}(L)$,true): $\text{loc}(L)$, not $L=g(1,1)$.
init($\text{isPit}(L)$,false): $\text{loc}(L)$.
init($\text{isVisited}(L)$,true): $L=g(1,1)$.
init($\text{isVisited}(L)$,false): not $L=g(1,1)$.

The agent can always execute the sensing actions $\text{smell}$, $\text{senseBreeze}$, and $\text{senseGold}$ which give clues about the location of the Wumpus, the pits, and the gold, respectively. In order to move around, the agent can perform action $\text{turn}$ which represents a change of direction by 90 degrees clockwise and $\text{moveFwd}$ which represents a move, one step forward to the direction the agent is facing. This action can only be executed if it leads to a valid grid location. The agent can also perform actions $\text{pickGold}$ and $\text{shootFwd}$ with the intuitive meaning and preconditions. Similarly, actions $\text{enter}$, $\text{climb}$ represent that the agent goes in or leaves the dungeon. Some action/1 and poss/2 clauses follow.

action($\text{smell}$).
action($\text{pickGold}$).
action($\text{shootFwd}$).
poss($\text{smell}$,true).
poss($\text{pickGold}$, $\text{isGold}(\text{locA})$=true).
poss($\text{shootFwd}$, $\text{hasArrow}$=true).

As described in Section 2.1, the actions change the possible values of the fluents, updating in this way the agent’s knowledge about the world. The fluents that represent the position of the agent, $\text{dirA}$ and $\text{locA}$, are affected only by action $\text{turn}$ and $\text{moveFwd}$ respectively. For each of the four directions, $\Pi_W$ includes an appropriate causes/4 clause of the following form.

causes($\text{turn}, \text{dirA}, Y$, (dirA=up,Y=right))).

Since there is complete information about $\text{locA}$ and $\text{dirA}$ initially, these clauses make sure that each fluent has exactly one possible value in all histories. Similarly, $\text{inDungeon}$ is affected only by actions $\text{enter}$ and $\text{climb}$, $\text{hasArrow}$ by action $\text{shootFwd}$, $\text{noGold}$ by action $\text{pickGold}$, and $\text{isVisited}(L)$ by $\text{moveFwd}$. For these fluents too, there is complete information in all histories.

The rest of the fluents can capture incomplete information which may shrink whenever a sensing action is performed. Fluent $\text{locW}$ is sensed by a $\text{smell}$ action: if there is no stench (i.e. the sensing result is 0) then each of the robot’s adjacent locations is not a possible value for $\text{locW}$, otherwise the opposite holds. Fluent $\text{isGold}(L)$ is sensed by a $\text{senseGold}$ action which settles the value of the fluent depending on the sensing result. $\Pi_W$ includes the following clauses.

rejects($\text{smell}$,0,$\text{locW}$,$\text{adj}(\text{locA},Y)$).
rejects($\text{smell}$,1,$\text{locW}$,$\text{neg}(\text{adj}(\text{locA},Y))$).
settles($\text{senseGold}$,1,$\text{isGold}(L)$,true,locA=L).
settles($\text{senseGold}$,0,$\text{isGold}(L)$,false,locA=L).

Fluent $\text{isPit}(L)$ is sensed by a $\text{senseBreeze}$ action: if no breeze is sensed (i.e. the sensing result is 0), then $\text{isPit}(L)$ is settled to value false for the locations $L$ which are adjacent to the agent. Otherwise, if all the adjacent locations but one are known not to contain a pit, then the unknown one is settled to contain a pit. Note that this last rule is indeed limited for reasoning about pit locations in the sense that it is incomplete whenever there is uncertainty in more than one adjacent location. Finally, fluent $\text{aliveW}$ is affected by the $\text{shootFwd}$ action: if the shot was in the right direction, then the Wumpus dies.

We conclude this section with an example. In the empty history, the agent is located at $g(1,1)$ and senses the stench of the Wumpus. The disjunctive knowledge about fluent $\text{locW}$ then shrinks, so that the only possible values are the locations which are adjacent to the agent.

?- mTrue($\text{locW}=g(1,2)$, [o($\text{smell}$,1)]),
mTrue($\text{locW}=g(2,1)$, [o($\text{smell}$,1)]).
Yes.
?- $\text{loc}(L)$, not $\text{adj}(g(1,1),L)$,
mTrue($\text{locW}=L$, [o($\text{smell}$,1)]).
No.
Sensing can further limit the possible values to be exactly one. For example,

\[-k_{\text{True}}(\text{locW} = g(3,1), [o(\text{smell},0), o(\text{moveFwd},1), o(\text{turn},1), o(\text{moveFwd},1), o(\text{turn},1), o(\text{turn},1), o(\text{smell},1), o(\text{moveFwd},1), o(\text{smell},0)])].

Yes

### 4 Experimental Results in the Wumpus World

In order to test the feasibility of our approach, we defined an INDIGOLOG high-level controller, which is intended to run w.r.t. the domain description given in the previous section, and experimented on several random settings of the scenario. The controller we used is quite simple: it is implemented using three concurrent interrupts at three different levels of priority, which can be described as follows:

1. If the Wumpus is known to be alive at a location \(l\) which is aligned with the agent’s location, then execute procedure \(\text{shoot}\) with the direction at which the Wumpus is known to be. Procedure \(\text{shoot}(d)\) is in charge of aiming and shooting the arrow at direction \(d\); it is defined using a search block as follows:

   \[
   \text{proc shoot}(d) \\
   \Sigma(\text{turn}):(\text{dirA} = d)?;\text{shootFwd} \\
   \text{endProc}
   \]

2. If there is a gold piece at the current location, then pick it up.

3. If the agent is in the dungeon, then she senses the world and proceeds to explore an unvisited location, provided it is safe and necessary to do so. Otherwise, she returns to location \(g(1,1)\) and climbs out of the dungeon. The exploration is done using an iterative deepening procedure, \(\text{explore}\). Procedure \(\text{goto}(l)\) goes to location \(l\) traversing only visited locations.

So, here is the INDIGOLOG controller in question that is to be executed online:

\[
\text{proc mainControl} \\
(d,I,: \text{locW} = l \land \text{aliveW} = \text{true} \land \text{aligned(locA, dir, locW) \rightarrow \text{shoot}(d)}) \rangle \\
(\text{isGold(locA) = true \rightarrow \text{pickGold}}) \\
(\text{inDungeon = true \rightarrow}
 |
 \{\text{noGold = 0};\text{explore} \} \cup \{\text{goto}(g(1,1));\text{climb}}
) \\
\text{endProc}
\]

We performed a series of experiments where the world is an 8 x 8 grid. A setting \((n, p)\) represents a random configuration with \(n\) gold pieces and a probability \(p\) of a location containing a pit. For each of these settings, we tested our controller in 300 random scenarios. The experiments verify that the agent always exits from the dungeon alive.

Figure 1 summarizes our evaluation metrics for some representative settings. The column “Gold” specifies the number of the scenarios where the agent managed to get gold. The column “Imp.” specifies the number of scenarios in which a pit or the Wumpus was located just next to the initial location of the agent, and hence the problem was unsolvable for the agent.\(^1\) The rest of the columns show the average of the reward, the number of moves, and the time it took the agent to exit the dungeon. Time is the total running time of the INDIGOLOG controller in seconds and rounded; this includes the deliberation time, as well as the time needed for INDIGOLOG to execute the simulated actions. The reward is calculated as follows: move actions cost \(-1\), a shoot action costs \(-10\), getting the gold is \(+1000\), and dying is \(-1000\). Each number in parenthesis is the corresponding average calculated only over the scenarios which are not “impossible” in the aforementioned sense.

As the probability \(p\) for the pits becomes higher, the scenarios which are not impossible become rare and exploring becomes difficult because there are not many locations which the agent can conclude that they are safe to go. For scenarios with 1 gold piece, which are not impossible, the effectiveness of the agent in getting the gold ranges from 73.9% for \(p = 0.10\) down to 23.1% for \(p = 0.40\). As expected, as \(p\) increases, the average reward and the average number of moves decrease. Note that the average running time drops as \(p\) increases, but this is merely because in many scenarios the agent has to give

\(^1\) Our agent is risk-averse: when no safe exploration is possible, the agent exits the dungeon.
up exploring very early. A safer metric for running time is the average time calculated over the scenarios which are not impossible; this also decreases because of the same reason, but it is more reasonable as a metric for the time the agent needs to fully explore the world.

Finally, as the number of gold pieces increase in the grid, the effectiveness of the agent increases also, both in respect to the average reward and in respect to the average time. This is because when there are more than one gold pieces in the grid, the chances that there exists a risk-averse path to one of them increase. As one can observe, this also mitigates the effect of having pits with high probability.

5 Discussion

In this article, we have shown how to model an intelligent agent acting in the Wumpus World by using the INDI GOLOG agent programming language.

The only available Cognitive Robotics implementation of the Wumpus World that we are aware of is that of FLUX [Thielscher, 2004]. FLUX is a constraint logic programming method in which agents could be programmed relative to fluent calculus action theories. The fluent calculus [Thielscher, 2000] extends the situation calculus by making explicit the notion of states—a snapshot of the environment characterized by the set of fluents that are true in it and some extra constraints representing incomplete knowledge. By appealing to states, FLUX uses progression, rather than regression, as the computational mechanism for answering queries about the world and the agent’s knowledge. By using progression and appealing to constraint programming, FLUX offers a computationally attractive framework for implementation of agents with incomplete information.

The results of running the available FLUX Wumpus implementation on exactly the same scenarios used for INDI GOLOG are shown in Figure 2. As one can observe, the empirical results do not seem to differ much from the INDI GOLOG ones and many differences may, in fact, only reflect the different strategies used in the agent controllers rather than the actual programming framework (e.g., FLUX seems to perform more actions). Nonetheless, we think there are two major issues worth mentioning. First, the FLUX implementation strongly relies on constraint solvers. This makes the agent reasoning substantially faster than when plain Prolog technology is used, as with our current INDI GOLOG implementation. We believe that this could be even more explicit when larger grids are used and, hence, it suggests the convenience of investigating how constraint programming can be incorporated in INDI GOLOG too. Notice that, like ours, the FLUX implementation also assumes a fixed grid size. Second, it is difficult, however, to compare the formal accounts of execution in FLUX and INDI GOLOG. This is mainly because the FLUX controller is no more than a constraint logic program; FLUX does not come with a well-defined notion of what an agent program execution is (e.g., in terms of single-step semantics) and there is no formal interleaved account of sensing, deliberation, and execution. The

References


A SNePS Approach to The Wumpus World Agent
or Cassie Meets the Wumpus

Stuart C. Shapiro and Michael Kandefer
Department of Computer Science and Engineering and Center for Cognitive Science
University at Buffalo, The State University of New York
Buffalo, NY 14260-2000
{shapiro|mwk3}@cse.buffalo.edu

1 Introduction
We demonstrate the use of SNeRE, the acting component of the SNePS knowledge representation, reasoning, and acting system, by showing its use to implement a wumpus world agent [Russell and Norvig, 1995]. For this purpose, we use SNePS 2.6.2, which consists of SNePS 2.6.1 [Shapiro et al., 2004] plus some patch files. We usually name our SNePS-based agents Cassie [Shapiro, 1989; 1998; Shapiro and Ismail, 2003; Shapiro et al., 2000; Shapiro and Rapaport, 1987; 1991]. To distinguish Cassie in the role of the wumpus world agent, we will call her CassieW.

Our main motivation in developing intelligent systems is to model general human-level intelligence, not to maximize the use of computing power to optimize problem solving. CassieW has been developed accordingly.

2 The Wumpus World
The wumpus world consists of a rectangular world of cells, within which is a rectangular cave, the size of which can vary from run to run. The cells within the cave are considered to be rooms; the border of the cave is a wall formed by cells that are not rooms. Each cell is identified by its Cartesian coordinates, cell(x, y), −1 ≤ x ≤ maxx, −1 ≤ y ≤ maxy, where maxx and maxy are parameters that are fixed within a set of runs. cell(0, 0) is always in one corner of the cave, and is CassieW’s “home room”. where she starts, facing east.

Each room in the cave, other than cell(0, 0), has a 20% probability of containing a pit. A live wumpus and a bar of gold are also placed in the cave randomly. Neither can be in cell(0, 0) nor in a room with a pit, although they can be in the same room as each other.

If CassieW ever goes into a room containing a pit or the live wumpus, she dies. However, in each of the rooms adjacent to a room that contains a pit, CassieW can detect a breeze, and in each of the rooms adjacent to a room that contains the wumpus, she can detect a stench. She can also detect a glitter in the room with the gold.

CassieW starts out with one arrow. If she shoots the arrow, it will travel in the direction she is facing until it either hits the wumpus or the far wall. If it hits the wumpus, the wumpus dies and CassieW can hear it scream.

CassieW’s task is to find the gold, grab it, return home (to cell(0, 0)), and stop.

For the graphical aspects of CassieW and her wumpus world, we are using Byron Weber Becker’s Java implementation of Rich Pattis’ Karel the Robot. Due to constraints of this system, cell(0, 0) is in the north-west corner instead of its usual position in the south-west corner; cell(1, 0) is to its east, and cell(0, 1) to its south. Our wumpus world is shown in Fig. 1.

CassieW is capable of performing the following standard wumpus-world primitive acts.

- **go(d):** If dis left or right, CassieW turns 90° left or right, respectively. If dis forward, CassieW goes to the room in front of her. However, if there is a wall in front of her, she doesn’t move, but can detect that she has bumped into the wall.

- **do(grab):** CassieW grabs for the gold; if she’s in the room with the gold, she’s successful.

- **do(shoot):** If she still has her arrow, CassieW shoots it.

---

1As described on http://www.cl.inf.tudresden.de/~mit/LRAPP/wumpus/wumpus.htm

2see http://www.learningwithrobots.com/
• **do**(stop): CassieW terminates all her activity; if she’s at cell(0,0), she exits the cave.

• **senseFor**(percept): CassieW actively senses for one of the possible percepts: stench, breeze, or glitter. See §11 for how CassieW perceives bumps and screams.

CassieW can also do (nothing), which is the act of not doing anything.

When CassieW either stops or dies, she receives a score, which is printed. The total score is the sum of: -1 for each **go**(d), **do**(grab), or **do**(stop); -10 for **do**(shoot); -1,000 for dying; and +1,000 for being in cell(0,0) with the gold when CassieW stops.

### 3 Use of The GLAIR Architecture

CassieW is implemented following the GLAIR (Grounded Layered Architecture with Integrated Reasoning) architecture [Hexmoor et al., 1993; Hexmoor and Shapiro, 1997; Shapiro and Ismail, 2003], and uses the following layers.

The Knowledge Layer (KL) is the layer at which “conscious” reasoning takes place. The KL is implemented in SNePS [Shapiro, 2000: Shapiro et al., 2004], and its sub-system SNeRE (the SNePS Rational Engine) [Kumar, 1996; Kumar and Shapiro, 1994a; 1994b; Shapiro et al., 2004]. SNePS, in turn, is implemented in Common Lisp.

The Perceptuo-Motor Layer, Sublayer a (PMLa) contains the Common Lisp implementation of the actions that are primitive at the KL. PMLa is implemented in a way that takes into account the top-level design of the agent, but is independent of the implementation of the agent’s body.

The Perceptuo-Motor Layer, Sublayer b (PMLb) implements the functions of PMLa taking into account the particular implementation of the agent’s body and environment. CassieW’s PMLb uses Franz Inc.’s Allegro CL jLinker3 to link Common Lisp code to Java programs, in which the lower layers are implemented.

The Perceptuo-Motor Layer, Sublayer c (PMLc); The Sensori-Actuator Layer (SAL); and The Environment are implemented as a set of Java classes and methods that specialize the Java implementation of Karel the Robot, and which is responsible for the display in Fig. 1.

### 4 Some SNePS Basics

A SNePS knowledge base is seen as containing the beliefs of the agent, itself, rather than being information about the agent. In that sense, the SNePS KB contains first-person beliefs of the agent. Of course, an agent may have beliefs about other agents, and these nested beliefs can be represented in SNePS [Shapiro and Rapaport, 1991; Chalupsky and Shapiro, 1996], but this facility is not used for CassieW. Another aspect of first-person representation is what is criterial for a belief’s being in the KB is not that it is true in the world, but that the agent is justified in believing it.

Similarly, SNePS can be used to reason about the actions of other agents, but the primary use of the SNePS acting system, and the use presented in this paper, is for the agent, itself, to act: it is a first-person acting system. It is also an on-line acting system. That is, it’s primary use, and the use presented in this paper, is to control the agent’s current acting. CassieW acts in her world, and, when necessary, she reasons about what she should do next based on: her beliefs about the current state of the world; the evidence of her sensory apparatus; a set of small stored or inferred plans (recipes) for carrying out certain actions or for bringing about certain states. This sensing, reasoning, and inferring is done on-line, while CassieW is acting.

The contents of CassieW’s KL (her beliefs) will be shown using SNePSLOG [Shapiro et al., 2004], which is one of a set of interface languages used to interact with SNePS agents. The current SNePSLOG syntax does not allow a formula to be a simple atomic symbol. It must consist of at least one function or predicate symbol with at least one argument. For example, neither a proposition such as HaveGold nor an act such as shoot is legal. Instead, CassieW uses Have(gold) and do(shoot), respectively.

In SNePS, propositions are reified [Shapiro, 1993]. That is, they are considered first-class members of the domain. So it is not really the case that Have, as used above, is a predicate symbol, nor that Have(gold) is a sentence denoting a truth value. Instead, Have is a function symbol, and Have(gold) is a proposition-valued functional term. Similarly, the SNePSLOG expression Have(gold) and ¬Alive(wumpus) is a functional term denoting the proposition, “I have the gold and the wumpus is not alive.” A SNePS agent may contemplate or have beliefs about propositions that it does not believe. If CassieW believes that she is facing east, we will say that she believes Facing(east), or that Facing(east) is asserted. An explicit Holds predicate is neither needed nor used.

The designers of any reasoning system must face the issue of, when new information, p, is inferred, should it be saved in the knowledge base? This is a traditional space-time trade-off. However, it may also be that p is a necessary step of a much longer derivation of q, and storing p may shorten that later derivation. Focusing on this role for p, we will refer to derived information that might or might not be saved in the knowledge base as lemmas. SNePS has been designed to save lemmas in the knowledge base.

Another basic decision for the designers of an agent is whether to give the agent a model of time. By this we mean whether the agent will have beliefs that certain events happened, or that certain acts were performed, at certain times. Some previous versions of Cassie (e.g., [Shapiro, 1998; Ismail and Shapiro, 2000; Ismail, 2001; Shapiro and Ismail, 2003]) had models of time. The alternative to a model of time is to have the agent have only situation-independent beliefs and beliefs about the current situation. These can, of course, include beliefs about past events and acts as long as multiple past times needn’t be distinguished. CassieW does not have a model of time, but can believe propositions such as Visited(cell(2,3)), meaning “I have visited cell(2,3).”

The belief that a fluent, a situation-dependent proposition, held at a particular time, once believed, may remain believed. However, the belief that a fluent holds now must be disbe-
lieved once it no longer holds. If any lemmas were derived from such fluents, they must be disbelieved also. SNePS uses SNeBR [Martins and Shapiro, 1988], an assumption-based truth maintenance system, for such house-cleaning.

Since the developers of SNePS have been interested in modeling general human-level intelligence, we have not built any numerical processing into SNePS. Therefore CassieW has been given the explicit beliefs Isa(i, Number), \(-1 \leq i \leq \text{max}(\text{max}_x, \text{max}_y)\), and Successor(i, i+1), \(-1 \leq i < \text{max}(\text{max}_x, \text{max}_y)\).

5 Directions and State Constraints

CassieW has a sense of direction, for which she uses: the individual constants, north, south, east, and west denoting the four directions; the individual Direction denoting the category of directions; and the proposition Isa(\{north, south, east, west\}, Direction) denoting the proposition that north, south, east, and west are directions. The proposition that each of north, south, east, and west is a Direction follows from this by the SNePS method of reduction inference [Shapiro, 1991].

CassieW also needs to know how the directions are arranged around the compass, for which she uses: Clockwise(d1,d2) for the proposition that direction d2 is clockwise from direction d1. CassieW believes Clockwise(north, east), Clockwise(east, south), Clockwise(south, west), and Clockwise(west, north).

CassieW needs to know what direction she’s facing. For this, she uses the proposition Facing(d), for the proposition, “I am facing direction d”. as well as the belief that she’s always facing in exactly one direction: andor(1,1){Facing(north), Facing(south), Facing(east), Facing(west)}. In SNePSLOG, andor(1,j){P1, ..., Pn}, where \{P1, ..., Pn\} is a set of propositions, denotes the proposition that at least i and at most j of the n Pk are true. So andor(1,1){P1, ..., Pn} denotes the proposition that exactly one of the Pk is true, and constitutes a state constraint. The use of state constraints will be discussed in §8. Note also that andor(0,0) is generalized nor, and \sim p is an abbreviation of andor(0,0){p}. At the beginning, CassieW believes that Facing(east), from which it follows that she isn’t facing north, south, or west.

6 The Cells and Rooms

Each cell in the wumpus world is denoted by the functional term cell(x,y). At the beginning, CassieW is given the beliefs that Isa(cell(x,y), Room), \sim Isa(cell(x,-1), Room), and \sim Isa(cell(,-1,y), Room), where 0 \leq x \leq \text{max}_x, 0 \leq y \leq \text{max}_y, and Room is an individual constant denoting the category of rooms in the cave. A cell that is not a room is part of the wall surrounding the cave, so CassieW starts off knowing where the north and west walls are. She will have to discover where the south and east walls are by herself.

CassieW is also given complete adjacency information using Adjacent3(c1,c2,d) for the proposition that cell c2 is d-of-cell c1. That is, CassieW believes that each room in the cave, including rooms next to walls, is adjacent to four cells.

Sometimes it is sufficient for CassieW to know that two rooms are adjacent without thinking about which direction one is from the other. For this, she uses the proposition Adjacent(c1,c2), for the proposition that cell c1 is adjacent to cell c2. Adjacent(c1,c2) is derivable from Adjacent3(c1,c2,d) by reduction inference.

CassieW is always in some cell (that’s a room). Her belief that she’s in cell c is represented by In(c). CassieW is initialized with the belief that In(cell(0,0)), and the state constraint that andor(1,1){{..., In(cell(x,y)), ...}, for 0 \leq x \leq \text{max}_x, 0 \leq y \leq \text{max}_y}.

Each room can also contain the wumpus or a pit. Of course, some room contains the gold, but CassieW never reasons about that — when she detects a glitter, she just grabs the gold. For the belief that a particular room contains the wumpus or a pit, CassieW uses Contains(r,x), denoting the proposition that room r contains x.

For the second argument of Contains, we use one of the individual constants, wumpus or pit. Although wumpus denotes the one and only individual wumpus, there is no need to individuate particular pits, so pit is actually being used like a mass noun — one individual constant for all the pits. One might read Contains(cell(3,5), pit) as “Cell (3,5) contains pit.” At the start, CassieW believes that \sim Contains(cell(0,0), wumpus) and \sim Contains(cell(0,0), pit).

For conciseness, we also use Safe(c) for the proposition, “cell c is safe for me to enter”. Safety and containing a pit or the wumpus are connected by the beliefs that all(c) {\sim Contains(c,pit) => (\{Alive(wumpus), \sim Contains(c,wumpus)\})}, and all(c) {\sim Safe(c)} \vDash \{\sim Contains(c, pit), andor(1,2){\sim Contains(c, wumpus), \sim Alive(wumpus)}\}. Where Alive(wumpus), means that the wumpus is alive. In SNePSLOG, \{A1, ..., An\} \vDash \{C1, ..., Cm\} means that if any Ai is believed, then any Cj may be believed.

Recall that all the rooms in the cave are cells, and have four adjacent cells. CassieW distinguishes rooms from walls by believing that each room is a cell, c, for which Isa(c, Room), but that each wall-cell is a cell, c, for which \sim Isa(c, Room). She never knowingly goes into a wall-cell, but she does believe that all(c) (\sim Isa(c, Room) => Safe(c)), which helps her locate the pits and the wumpus.

7 Propositions, Acts, and Policies

SNeRE recognizes three particular types of domain entities: propositions, acts, and policies. Propositions are entities that can be believed and whose negations can be believed. Acts are entities that a SNeRE agent can perform. Policies connect propositions and acts. Two SNeRE built-in policies are used by CassieW:

\text{whendo}(p,a): \text{When I believe the proposition } p, I \text{ will perform the act } a.
wheneverdo(p,a): Whenever I believe the proposition p, I will perform the act a.

In each case, if the policy has been adopted, the agent performs a when forward inference causes p to be believed. Also a is performed if p is already believed when the policy is adopted with forward inference. The difference is that a whendo policy is unadopted after firing once, but a wheneverdo remains adopted until explicitly unadopted.

We call something that the agent can perform an act. An act consists of an action and zero or more arguments. For example, CassieW’s act of going one cell forward, expressed in SNePSLOG as go(forward), consists of the action of going (go) and the argument forward (forward).

Since the smallest well-formed SNePSLOG expression is a functional term consisting of a function symbol and at least one argument, we use the functional term do(a) to represent the act of performing the action a on no arguments.

Any agent has a repertoire of primitive actions it can perform. We will say that an act whose action is a primitive act is a primitive act. We will call other acts actions complex.

SNeRE comes with a set of preprogrammed primitive actions: mental actions, discussed in the next section, and control actions. The control actions used by CassieW (for the complete set, see [Shapiro et al., 2004]) are:

• do-all({a1, ..., an}): Perform all the acts a1, ..., an in random order.
• do-one({a1, ..., an}): Perform one of the acts a1, ..., an chosen randomly.
• prdo-one({pract(x1, a1), ..., pract(xn, an)}): Perform one of the acts aj, with probability xj/(x1 + ... + xn).
• snsequence(a1, a2): Perform a1, and then perform a2. For sequences of three acts, snsequence3(a1, a2, a3) is used.
• withsome(?x, p(?x), a(?x), da): Using backward inference, determine which of the pi hold. If any do, randomly choose one, say e, and perform a(e). If no entity satisfies p(?x), perform da. withsome/3 is like withsome, but with da defaulting to do nothing. withall/3 is like withsome/3, but performs a(e) on all e that satisfy p(?x)

Additional primitive acts must be defined by the agent designer, and implemented in the PML and SAL. The primitive acts used by CassieW were described in §2.

8 Mental Acts

The two mental actions are believe and disbelieve.

When a SNeRE agent performs the mental act disbelieve(p), the result is that if p is a believed proposition, it is no longer believed, and if p is an adopted policy, it is no longer adopted. Note that disbelieving a proposition does not cause its negation to be believed, and that when p is disbelieved, SNeBR causes any lemmas that depended on p to also be disbelieved (see §4).

When a SNeRE agent performs the mental act believe(p), the result is: if p is a policy, it is adopted; if p is a proposition, it is believed as an hypothesis; and, in either case, forward inference is done with p. The forward inference may cause other propositions to be believed, policies to be adopted, and adopted policies to trigger.

However, before believe changes the belief status of a proposition p, it performs a limited form of belief revision [Alchourrón et al., 1985]:

1. If andor(0,0){...,p,...} is believed, disbelieve it.
2. If andor(i,1){p,q,...} and q are believed, disbelieve q.

Case 2 is the way state constraints (§5) are used.

9 (p => q) vs. whendo(p,believe(q))

When CassieW enters a room, r, she wants to remember that she has visited it, using the proposition Visited(r). andor(0,0)

In that way, when Visited(cell(2,3)) and Safe(cell(2,3)) are believed, they are asserted as hypotheses, and remain so even when CassieW moves to another room. She starts off believing that Visited(cell(0,0)), and ~Visited(c), for all other cells, c.

10 The SNeRE Execution Cycle

An abbreviated version of the SNeRE execution cycle makes use of these predefined proposition-forming functions:

• ActPlan(a1,a2): A plan for performing the complex act a1 is to perform the complex act a2 (which is usually structured using one or more of the SNeRE control acts);
• Effect(a,p): An effect of performing the act a is that the proposition p which could be of the form ~q) will hold. The abbreviated execution cycle is:

To perform the act a:

Use backward inference to find propositions pe such that Effect(a,pe);
if a is primitive, execute its implementation;
else Use backward inference to find acts a2 such that ActPlan(a,a2), and perform one of them;
for all pe, perform believe(pe).

We intend to extend this to a more unrestricted form of belief revision in the future.

We ignore preconditions in this paper, since CassieW doesn’t use them.
Active perception is accomplished by a sensory act, whose performance leads to a perception. Active perception happens when the PML triggers an act. These trigger the policies shown in Fig. 2, and allow CassieW to eventually locate the pits and the wumpus, if she explores the cave sufficiently. The formula \( \text{next}(\{x\}) \{\{P(x)\} : \{Q(x)\} \} \) denotes the proposition that, of the \( \{x\} \) individuals a that satisfy \( P(a) \), at least \( i \) and at most \( j \) also satisfy \( Q(a) \) [Shapiro, 1979].

Upon performing \text{senseFor}(x), where \( x \) is either breeze or stench, CassieW’s PMLa performs a believe either on \text{Feel}(x) or \~\text{Feel}(x). These trigger the policies shown in Fig. 2, and allow CassieW to eventually locate the pits and the wumpus, if she explores the cave sufficiently. The formula \( \text{next}(\{x\}) \{\{P(x)\} : \{Q(x)\} \} \) denotes the proposition that, of the \( \{x\} \) individuals a that satisfy \( P(a) \), at least \( i \) and at most \( j \) also satisfy \( Q(a) \) [Shapiro, 1979].

Upon performing \text{senseFor}(x), CassieW’s PMLa performs a believe either on \text{Feel}(x) or \~\text{Feel}(x). These trigger the policies shown in Fig. 2, and allow CassieW to eventually locate the pits and the wumpus, if she explores the cave sufficiently. The formula \( \text{next}(\{x\}) \{\{P(x)\} : \{Q(x)\} \} \) denotes the proposition that, of the \( \{x\} \) individuals a that satisfy \( P(a) \), at least \( i \) and at most \( j \) also satisfy \( Q(a) \) [Shapiro, 1979].

11 Active and Passive Perception

Perception is accomplished in GLAIR agents by the PML performing a believe on a proposition that some object or phenomenon has been perceived. As mentioned in §8, this could cause inferences to be drawn and acts to be performed via adopted policies.

Active perception is accomplished by a sensory act, whose performance leads to a perception. Active perception is done by CassieW with the sensory acts \text{senseFor}(stench), \text{senseFor}(breeze), and \text{senseFor}(glitter). They are combined into one complex act, \text{ActPlan}(\text{do}(\text{perceive}), \text{do-all}({\text{snif}(\text{if}(\neg \text{Have(gold)}, \text{senseFor(breeze)})), \text{snif}(\text{if}(\text{Have(gold)}, \text{senseFor(glitter)})), \text{snif}(\text{if}(\text{Alive(wumpus)}, \text{senseFor(stench)}))})).

so that she bothers to \text{senseFor}(stench) only if she believes that the wumpus is still alive, and to \text{senseFor}(glitter) and \text{senseFor(breeze)} only if she doesn’t already have the gold. (As we’ll see in §15, she doesn’t have to worry about the pits when she is on her way home with the gold.) She starts out believing that Alive(wumpus) and \~\text{Have(gold)}.

Upon performing \text{senseFor}(x), where \( x \) is either breeze or stench, CassieW’s PMLa performs a believe either on \text{Feel}(x) or \~\text{Feel}(x). These trigger the policies shown in Fig. 2, and allow CassieW to eventually locate the pits and the wumpus, if she explores the cave sufficiently. The formula \( \text{next}(\{x\}) \{\{P(x)\} : \{Q(x)\} \} \) denotes the proposition that, of the \( \{x\} \) individuals a that satisfy \( P(a) \), at least \( i \) and at most \( j \) also satisfy \( Q(a) \) [Shapiro, 1979].

Upon performing \text{senseFor}(x), CassieW’s PMLa performs a believe either on \text{Feel}(x) or \~\text{Feel}(x). These trigger the policies shown in Fig. 2, and allow CassieW to eventually locate the pits and the wumpus, if she explores the cave sufficiently. The formula \( \text{next}(\{x\}) \{\{P(x)\} : \{Q(x)\} \} \) denotes the proposition that, of the \( \{x\} \) individuals a that satisfy \( P(a) \), at least \( i \) and at most \( j \) also satisfy \( Q(a) \) [Shapiro, 1979].

Upon performing \text{senseFor}(x), CassieW’s PMLa performs a believe either on \text{Feel}(x) or \~\text{Feel}(x). These trigger the policies shown in Fig. 2, and allow CassieW to eventually locate the pits and the wumpus, if she explores the cave sufficiently. The formula \( \text{next}(\{x\}) \{\{P(x)\} : \{Q(x)\} \} \) denotes the proposition that, of the \( \{x\} \) individuals a that satisfy \( P(a) \), at least \( i \) and at most \( j \) also satisfy \( Q(a) \) [Shapiro, 1979].

12 Dead Reckoning

CassieW knows that she starts in \text{cell}(0,0) facing east, but she has to keep track of her position and facing afterwards by dead reckoning. She can keep track of the direction she’s facing by her knowledge of the effects of turning:
Bored(1), Bored(2), Bored(3), Bored(4)]

Figure 4: CassieW’s plans for believing where the east and south walls are.

\[
\text{all}(r) \land (\text{In}(r)) \Rightarrow \text{all}(d) \land (\text{Facing}(d))
\]

\[
\Rightarrow (\text{all}(d) \land (\text{Facing}(d)) \Rightarrow (\text{all}(r,d) \land (\text{Adjacent3}(r,d)))
\]

Figure 5: CassieW’s belief about the effects of going forward.

\[
\text{all}(d_1,d_2,d_3) \land \{\text{Facing}(d_1,d_2), \text{Clockwise}(d_2,d_3)
\land \Rightarrow (\text{Effect}(\text{go}(\text{right}), \text{Facing}(d_3)),
\text{Effect}(\text{go}(\text{left}), \text{Facing}(d_1)))
\]

\[
\text{(The formula } \{P_1, \ldots, P_n\} \Rightarrow \{Q_1, \ldots, Q_m\} \text{ denotes the proposition that if all the } P_i \text{ are true, then so are all the } Q_j.\]

CassieW’s belief about the effects of going forward is shown in Fig. 5. It has three parts: 1) after going forward, she is in the room in front of her; 2) she visited this new room from the previous room; 3) if she earlier visited this new room from some other room, just remember that occurrence. The accuracy of these effects relies on the fact that, if this act of going forward resulted in CassieW’s feeling a bump, the effects of the bumping will be believed before these effects of going forward. Therefore, the cell in front of her will not be a room, and these effects will not be believed. The set of VisitedFrom(r,r1) beliefs (meaning “I visited room r from room r1”) will form a trail of “crumbs” CassieW will follow after she finds the gold (see §15). Avoiding new visits in favor of old visits cuts loops in this trail.

Note that the construct withsome(?n1, Bored(?n1), do(raiseBoredom)) is the autoepistemic policy, perform a if you know of no ?x for which you believe p(?x).

13 Finding the Gold

CassieW’s strategy to find the gold is to explore the cave semi-randomly. Her explore(cave) plan uses a three-way categorization of the rooms: rooms she has already visited are “old rooms”; rooms she has not yet visited are “new rooms”; new rooms that she knows are safe are “safe new rooms.” CassieW’s rule for categorizing rooms is shown in Fig. 6. The proposition RoomType(r,r2,d) means that r2 is of the given RoomType, and is just d-ward of room r.

CassieW’s semi-random exploration is further controlled by her level of boredom, which is represented by Bored(i), denoting the proposition “My level of boredom is i”, for \(0 \leq i \leq \text{max}_b\). Currently \(\text{max}_b\) is 4, so CassieW begins with the state constraint andor(1,1)\{Bored(0), Bored(1), Bored(2), Bored(3), Bored(4)\} and the initial belief that Bored(0). She increases her level of boredom, up to \(\text{max}_b\), with the act do(raiseBoredom):

\[
\text{ActPlan}(\text{do}(\text{raiseBoredom}),
\text{snif}(\text{if}(\text{Bored}(4),
\text{withsome}(?n1, \text{Bored(?n1)},
\text{withsome}(?n2, \text{Successor(?n1,?n2), believe(\text{Bored(?n2))}))))))
\]

CassieW’s plan for exploring the cave is shown in Figure 7. If she can move to a safe new room, she’ll do that, and set her boredom level to 0; if she can’t find such a room, and she’s not totally bored, she’ll go to an old room (she needn’t do (perceive) there), and increase her boredom level; if she can’t find a safe new room, and she’s totally bored, she’ll move to any new room (even though she might die), and set her boredom level to 0; in any other case, she’ll make a random move. After making one move, CassieW continues exploring.

CassieW makes a random move by going forward 50% of the time, and right or left 25% of the time each. However, moving randomly is boring:

\[
\text{ActPlan}(\text{do}(\text{random}), \text{snsequence}(
\text{prdo-one}((\text{pract}(50, \text{move(} \text{forward})),
\text{pract}(25, \text{go(left)})),
\text{pract}(25, \text{go(right)})),
\text{do}(\text{raiseBoredom}))))
\]

CassieW’s plan for explore(cave) uses the act turn(d), where d is some direction. CassieW’s plans for turning are shown in Fig. 8

14 Shooting the Wumpus

CassieW does not go to any particular effort to try to shoot the wumpus. She explores the cave, looking for the gold. If she happens to locate the wumpus before getting the gold, she adopts the policy that if she happens to be in a room in the same row or column as the wumpus, then, if she still has her arrow, she should turn toward the wumpus and shoot:

\[
\{\text{Have(gold), Alive(wumpus)}\} \Rightarrow \{
\text{all}(r,d) \land (\text{WumpusAhead}(r,d) \Rightarrow
\text{whendo}(\text{In}(r), \text{snif}(\text{if}(\text{Have(arrow)}),
\text{snsequence}(\text{turn}(d), \text{do(} \text{shoot}))))))
\]

WumpusAhead(r,d) denotes the proposition that the wumpus is somewhere d-ward of room r. It can be
all(r)(Isa(r, Room) => all(r2,d)((Adjacent3(r,r2,d), Isa(r2,Room)) &=> {
(Visited(r2) => OldRoom(r,r2,d)),
(¬Visited(r2)) v=> {NewNextRoom(r,r2,d), Safe(r2) => SafeNewRoom(r,r2,d)}}))

Figure 6: CassieW’s rule for categorizing rooms.

¬Have(gold) => (all(r1)(In(r1) => ActPlan(explore(cave),
snsequence(withsome({?r2,?d1}, SafeNewRoom(r1,?r2,?d1),
snsequence3(turn(?d1), move(forward), believe(Bored(0))),
snif({if(Bored(4), withsome({?r3,?d2}, NewNextRoom(r1,?r3,?d2),
snsequence3(turn(?d2), move(forward), believe(Bored(0))),
do(random))),
else(withsome({?r3,?d2}, OldRoom(r1,?r3,?d2),
snsequence3(turn(?d2), go(forward), do(raiseBoredom)),
do(random))))),
explore(cave)))))

15 Getting Home

As soon as CassieW believes she has the gold, she changes the explore(cave) plan to one of finding home: Have(gold) => ActPlan(explore(cave), find(home)). Her plan for finding home, shown in Fig. 9 is to go back the way she came, using her VisitedFrom beliefs as “crumbs” (see §11), picking them up as she goes, and stopping when she reaches cell(0,0). She doesn’t need to do(perceive) on her way home.

16 Results

We ran a series of trials with max x = max y = 6, and the width and height of the cave each independently 4±1. We ran enough trials so that there were 10 cases of each of 3 types: P — it is possible for CassieW to get to the gold moving only into safe rooms; I — it is impossible for CassieW to get to the gold; PS — the gold is perceptually screened from CassieW, i.e., she can get the gold only by taking a risky move. The average scores are shown in the table below.

<table>
<thead>
<tr>
<th>Type</th>
<th>Nwon</th>
<th>AvgWon</th>
<th>Nlost</th>
<th>AvgLost</th>
<th>Avg</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>9</td>
<td>985.0</td>
<td>1</td>
<td>-2023.0</td>
<td>684.2</td>
</tr>
<tr>
<td>PS</td>
<td>2</td>
<td>982.0</td>
<td>8</td>
<td>-2006.5</td>
<td>-1408.8</td>
</tr>
<tr>
<td>I</td>
<td>0</td>
<td>10</td>
<td>10</td>
<td>-2008.5</td>
<td>-2008.5</td>
</tr>
</tbody>
</table>

17 Conclusions

The SNePS/SNeRE knowledge representation, reasoning and acting system provides an expressive language for building agents that perform integrated first-person, on-line reasoning and acting. GLAIR is an effective architecture for building such agents. CassieW, a wumpus world agent, is an excellent example of the use of GLAIR and SNePS/SNeRE.

References


all(d)(Isa(d, Direction) => ActPlan(turn(d),
snif(if("Facing(d), withsome/3(?f, Facing(?f),
snif((if(Clockwise(d,?f), go(left)),
if(Clockwise(?f,d), go(right)),
else(turn(around))))))))

ActPlan(turn(around), snsequence(go(right), go(right)))

Figure 8: CassieW’s plans for turning.

tall(r)(In(r) => ActPlan(find(home), snif({if(In(cell(0,0)), do(stop)),
else(snsequence(withsome/3(?r2,VisitedFrom(r, ?r2),
withsome/3(?d,Adjacent3(r, ?r2, ?d),
snsequence3(turn(?d), go(forward), disbelieve(VisitedFrom(r, ?r2))))),
find(home))))))

Figure 9: CassieW’s plan for finding home.


A FLUX Agent for the Wumpus World

Michael Thielscher
Dresden University of Technology
mit@inf.tu-dresden.de

Abstract

FLUX is a programming method for the design of agents that reason logically about their actions and sensor information in the presence of incomplete knowledge. We show how FLUX can be used to program an agent for the Wumpus World. Experimental results show how the agent performs in terms of the reward function and how well the FLUX program scales up.

1 Introduction

The paradigm of Cognitive Robotics [1] is to endow agents with the high-level cognitive capability of reasoning. Exploring their environment, agents need to reason when they interpret sensor information, memorize it, and draw inferences from combined sensor data. Acting under incomplete information, agents employ their reasoning facilities for selecting the right actions. To this end, intelligent agents form a mental model of their environment, which they constantly update to reflect the changes they have effected and the sensor information they have acquired. The Wumpus World [2] is a good example of an environment where an agent needs to choose its actions not only on the basis of the current status of its sensors but also on the basis of what it has previously observed or done. Moreover, some properties of the environment can only be observed indirectly and require the agent to combine observations made at different stages.

FLUX [4; 5] is a high-level programming method for the design of intelligent agents that reason about their actions on the basis of the fluent calculus [3]. A constraint logic program, FLUX comprises a method for encoding incomplete states along with a technique of updating these states according to a declarative specification of the elementary actions and sensing capabilities of an agent. Incomplete states are represented by lists (of fluents) with variable tail, and negative and disjunctive state knowledge is encoded by constraints.

FLUX programs consist of three parts: A kernel provides the general reasoning facilities by encoding the foundational axioms of the fluent calculus. The domain-specific background theory contains the formal specification of the underlying environment, including effect axioms for the actions of the agent. Finally, the strategy specifies the intended behavior of the agent. Space limitations do not permit us to fully recapitulate syntax and semantics of FLUX; we refer to [4; 5] for details. In the following section, we present a FLUX background theory for the Wumpus World, and in Section 3 we give a FLUX program that implements a particular strategy for an agent that systematically explores an unknown grid with the goal to find and bring home the gold. We conclude in Section 4 by reporting on some experiments and outlining possible ways of improving the basic strategy.

2 The Background Theory

A background theory describes the general properties of the environment and the actions of the agent. Following the specification laid out in [2], the Wumpus World agent moves in a rectangular grid of cells. An example scenario is depicted in Figure 1: There is a heap of gold somewhere in the grid, some of the cells contain bottomless pits, and one them houses the hostile Wumpus. The agent perceives a breeze (a stench, respectively) if it is adjacent to a cell containing a pit (the Wumpus, respectively), and the agent notices a glitter in any cell containing gold and it hears a scream if the Wumpus gets killed (through the arrow shot by the agent). Figure 2 shows...
where the agent will sense a breeze, stench, and glitter, respectively, with respect to the scenario of Figure 1.

To axiomatize the Wumpus World, we use the following nine fluents. \( A_i(x, y) \) and \( \text{Facing}(d) \) represent, respectively, that the agent is in cell \((x, y)\) and faces direction \(d\) \(\in\{1, 2, 3, 4\}\) (north, east, south, or west); \( \text{Gold}(x, y) \), \( \text{Pit}(x, y) \), and \( \text{Wumpus}(x, y) \) represent that square \((x, y)\) houses, respectively, gold, a pit, or the Wumpus; \( \text{Dead} \) represents that the Wumpus is dead; \( \text{Has} \) represents that the agent has \( \text{Gold} \) Arrow; and \( Y_{\text{dim}} \) and \( X_{\text{dim}} \) represent the (initially unknown) extent of the grid. The agent does not need a fluent to represent its own status of being alive or not, because we intend to write a cautious strategy by which the agent never takes the risk to fall into a pit or to enter the square with the Wumpus (unless the latter is known to be dead).

FLUX allows to combine physical and sensing effects in single action specifications, which we have exploited here: Update axioms are encoded in FLUX as definitions of the predicate \( \text{StateUpdate}(Z_1, A(x, y), z_2, y) \) describing the update of state \( z_1 \) to \( z_2 \) according to the physical effects of action \( A(x, y) \) and the sensing result \( y \). We assume that when executing any of its physical action, the agent perceives a vector with five truth-values:

\[
\begin{bmatrix}
\text{stench}, \text{breeze}, \text{glitter}, \text{bump}, \text{scream}
\end{bmatrix}
\]

The actions of the Wumpus World agent are then encoded as follows:

\[
\text{state_update}(Z_1, \text{go}, Z_2, [S, B, G, \text{Bump}, \_]) :=
\begin{align*}
& \text{holds(at}(X,Y),Z_1), \text{holds(facing}(D),Z_1), \\
& \text{adjacent}(X,Y,D,X1,Y1), \\
& (\text{Bump}=\text{false} -> \\
& \text{update}(Z1,[\text{at}(X1,Y1)],\text{at}(X,Y),Z2), \\
& \text{stench_perception}(X1,Y1,S,Z2), \\
& \text{breeze_perception}(X1,Y1,B,Z2), \\
& \text{glitter_perception}(X1,Y1,G,Z2) \\
& ; \text{Bump}=\text{true}, \\
& \text{stench_perception}(X,Y,G,Z2) \\
& ) .
\end{align*}
\]

\[
\text{state_update}(Z1, \text{turn_left}, Z2, [S, B, G, \_]) :=
\begin{align*}
& \text{holds(facing}(D),Z1), \\
& (D\neq4 \#\slash D1\#\neq\text{D-1}) \#\slash (D\neq1 \#\slash D1\#\neq\text{D-1}), \\
& \text{update}(Z1, [\text{facing}(D1)], \text{facing}(D),Z2), \\
& \text{holds(at}(X,Y),Z2), \\
& \text{stench_perception}(X,Y,S,Z2), \\
& \text{breeze_perception}(X,Y,B,Z2), \\
& \text{glitter_perception}(X,Y,G,Z2).
\end{align*}
\]

In the effect axiom for \( \text{Go} \), for instance, the first four components of the sensory input are evaluated: If the agent does not perceive a \( \text{bump} \), then the physical effect is to reach the adjacent location, and the \( \text{stench} \), \( \text{breeze} \), and \( \text{glitter} \) percepts are then evaluated against the updated (incomplete) state. The auxiliary predicates used in this and the other update axioms employ the two constraints \( \text{NotHolds} \) and \( \text{OrHolds} \), for which the FLUX kernel contains a constraint solver:

\[
\text{stench_perception}(X,Y,\text{Percept},Z) :=
\begin{align*}
& \text{XE}=X+1, \text{XW}=X-1, \text{YN}=Y+1, \text{YS}=Y-1, \\
& (\text{Percept}=\text{false} -> \\
& \text{not_holds(wumpus}(X,E),Z), \\
& \text{not_holds(wumpus}(X,W),Z), \\
& \text{not_holds(wumpus}(X,N),Z), \\
& \text{not_holds(wumpus}(X,S),Z) \\
& ; \text{Percept}=\text{true}, \\
& \text{or_holds([wumpus}(X,E),Z), wumpus}(X,W), wumpus}(X,N), wumpus}(X,S)\} ,Z) .
\end{align*}
\]

\[
\text{breeze_perception}(X,Y,\text{Percept},Z) :=
\begin{align*}
& \text{XE}=X+1, \text{XW}=X-1, \text{YN}=Y+1, \text{YS}=Y-1, \\
& (\text{Percept}=\text{false} -> \\
& \text{not_holds(pit}(X,E),Z), \\
& \text{not_holds(pit}(X,W),Z), \\
& \text{not_holds(pit}(X,N),Z), \\
& \text{not_holds(pit}(X,S),Z) \\
& ; \text{Percept}=\text{true}, \\
& \text{or_holds([pit}(X,E),Z), pit}(X,W), pit}(X,N), pit}(X,S)\} ,Z) .
\end{align*}
\]

\[
\text{glitter_perception}(X,Y,\text{Percept},Z) :=
\begin{align*}
& \text{Percept}=\text{false} \rightarrow \text{not_holds(gold}(X,Y),Z), \\
& \text{Percept}=\text{true}, \text{holds(gold}(X,Y),Z).
\end{align*}
\]

The update axioms for the two \( \text{Turn} \) actions use standard predicates for FD-constraints (finite domains), preceded by
the symbol “#”. The update clauses are direct encodings of
the corresponding knowledge update axioms for the Wumpus
World [4].

For the sake of simplicity, our FLUX program for the
Wumpus World agent does not include precondition axioms,
since going and turning is always possible while the precon-
ditions for Grab, Shoot, and Exit are implicitly verified as
part of the strategy (see Section 3). In addition to the specifica-
tions of the actions, a FLUX background theory consists of
domain constraints and an initial state specification. Initially,
the agent is at (1,1), faces west (that is, 2) and possesses an
arrow. Moreover, the agent knows that the Wumpus is still
alive and that the home square is safe. The agent does not
know the extension of the grid, nor the locations of the gold,
the Wumpus, or any of the pits. The domain constraints are
summarized in a clause defining consistency of states, which
adds range and other constraints to the initial knowledge of
the agent:

\[
\text{init}(Z0) :-
Z0 = [at(1,1), facing(2), has(arrow) | Z],
not_holds(dead,Z),
not_holds(wumpus(1,1),Z0),
not_holds(pit(1,1),Z),
\]
consistent(Z).

consistent(Z) :-
% uniqueness constraints
holds(xdim(X),Z,Z1),
not_holds_all(xdim(_),Z1),
holds(ydim(Y),Z,Z2),
not_holds_all(ydim(_),Z2),
holds(at(AX,AY),Z,Z3),
not_holds_all(at(_,_),Z3),
holds(facing(D),Z,Z4),
not_holds_all(facing(_),Z4),
holds(gold(GX,GY),Z,Z5),
not_holds_all(gold(_,_),Z5),
holds(wumpus(WX,WY),Z,Z6),
not_holds_all(wumpus(_,_),Z6),

% range constraints
[X,Y] :: [1..100], D :: [1..4],
AY #>= 1, AX #<= X, AY #>= 1, AY #<= Y,
GX #>= 1, GX #<= X, GX #>= 1, GX #<= Y,
WX #>= 1, WX #<= X, WX #>= 1, WX #<= Y,

% constraints for pits (boundary etc.)
not_holds_all(pit(_0,0),Z),
not_holds_all(pit(0,0),Z),
Y1 #= Y+1, not_holds_all(pit(_,Y1),Z),
X1 #= X+1, not_holds_all(pit(X1,0),Z),
not_holds(pit(GX,GY),Z),
not_holds(pit(WX,WY),Z),
duplicate_free(Z).

Here, the FLUX kernel predicate \( \text{Holds}(f,z,z_1) \) means that
fluent \( f \) holds in state \( z \), and \( z_1 \) is \( z \) without \( f \). The kernel
constraint \( \text{DuplicateFree}(z) \) is used to ensure that fluents do
not occur twice in state list \( z \).

3 The Strategy

Strategy programs are based on the following pre-defined
FLUX predicates:

- **Knows(f,z)** (respectively, KnowsNot(f,z)), meaning that fluent \( f \) is known to hold (respectively, known not to hold)
in incomplete state \( z \).
- **KnowsVal(\vec{x},f,z)**, meaning that fluent \( f \) is known to hold for arguments \( \vec{x} \) in incomplete state \( z \).
- **Execute(a,z_1,z_2)**, meaning that the actual execution of
  action \( a \) in state \( z_1 \) leads to state \( z_2 \).

These predicates are used in the followin sample control pro-
gram for a cautious Wumpus World agent:

**main :-**

- init(Z0), execute(turn_left,Z0,Z1),
- Cpts=[1,1,[1,2]], Vis=[[1,1]], Btr=[],
- main_loop(Cpts,Vis,Btr,Z).

**main_loop([X,Y,Choices|Cpts],Vis,Btr,Z) :-**

- Choices=[Dir|Dirs) ->
  (explore(X,Y,Dir,Vis,Z1) ->
   main_loop([X,Y,Dirs|Cpts], Vis,Btr,Z1)
  ;
   knows_val([X1,Y1],at(X1,Y1),Z1) ->
   execute(grab,Z1,Z2),
   go_home(Z2)
  ;
   hunt_wumpus(X1,Y1,Z1,Z2, Vis,Vis2,Cpts,Cpts1),
   main_loop(Cpts1,Vis1,Btr1,Z2)
  )
  
  ;
  main_loop([X,Y,Dirs|Cpts],Vis,Btr,Z)
  ;
  backtrack(Cpts,Vis,Btr,Z).

**explore(X,Y,D,V,Z1,Z2) :-**

- adjacent(X,Y,D,X1,Y1),
- \(+\) member([X1,Y1],V),
- (D#=1 -> \(-\) knows(ydim(Y),Z1),
  D#=2 -> \(-\) knows(xdim(X),Z1),
  true),
  knows_not(pit(X1,Y1),Z1),
  ( knows_not(wumpus(X1,Y1),Z1),
    knows_not(pit(X1,Y1),Z1),
    knows_not(wumpus(X1,Y1),Z1),
    knows_not(pit(X1,Y1),Z1),
    turns_to(D,Z1,Z2),
    execute(go,Z2,Z2).

**backtrack(_,_,[,] :-**

**backtrack(Cpts,Vis,D,Btr,Z) :-**

- R is (D+1) mod 4 + 1,
- turns_to(R,Z1),
- execute(go,Z1,Z2),
- main_loop(Cpts1,Vis1,Cpts1),
- main_loop(Cpts1,Vis1,Btr1,Z3).

**turn_to(D,Z1,Z2) :-**

- knows(facing(D),Z1) -> Z2=Z1
After the initialization of the world model and the execution of a TurnLeft action at the home square (to acquire the first sensory input), the main loop is entered by which the agent systematically explores the environment. To this end, the program employs three parameters containing, respectively, from a location once all choices have been considered. A choice points yet to be explored, the squares that have been visited, and the current path. The latter is used to backtrack from a location once all choices have been considered. A choice point is a list of directions, which are encoded by a predicate Explore checks whether it has found the gold. If so, it takes the quick-

\[
\text{path}(X,Y,WX,WY,\text{Vis}) \\
\text{go_home}(Z) := \\
\text{a_star_plan}(Z,S), \\
\text{execute}(S,Z,Z1), \text{execute}(\text{exit},Z1,_) .
\]

The agent hunts the Wumpus only in case the location is known and, for the sake of simplicity, only when the agent happens to be in the same row or column (predicate InDirection). When the Wumpus has been killed, the agent can explore areas which it may have rejected earlier. Therefore, all cells that lie on the path \(\theta\) between the agent and the Wumpus (predicate Path\((x,y,wx,wy,\theta)\)) may be re-visited. To this end, it is ensured that the current list of choice points includes the direction in which the agent has shot the arrow.

The following table illustrates what happens in the first nine calls to the main loop when running the program with the scenario depicted in Figure 1.

The letters \(G\), \(L\) are abbreviations for the actions Go and TurnLeft, respectively. After going north to (1, 2), the agent cannot continue in direction 1 or 2 because both (1, 3) and (2, 2) may be occupied by a pit according to the agent's current knowledge. Direction 3 is not explored since location (1, 1) has already been visited, and direction 4 is ruled out as (0, 2) is outside of the boundaries. Hence, the agent backtracks to (1, 1) and continues with the next choice there, direction 2, which brings it to location (2, 1). From there it goes north, and so on. Eventually, the agent arrives at (5, 5), where it senses a glitter and grabs the gold. The backtracking path at this stage is depicted in Figure 3. Along its way, the agent has determined the square with the Wumpus and shot its arrow in order to safely pass through this square.

4 Experimental Results

In order to see how the agent performs wrt. the reward function as specified in [2] and how the program scales to environments of increasing size, we ran series of experiments.
Figure 3: Exploring the cave depicted in Figure 1, our agent eventually reaches the cell with the gold, having shot the Wumpus along its way and inferred the locations of three pits. Parts of the grid are still unknown territory.

with square grids of different size. The scenarios were randomly chosen by adding a pit to each cell with probability 0.2 and then randomly placing the Wumpus and the gold in one of the free cells. The following table shows the cumulated reward over 100 runs each for various grid sizes. The third row shows the number of successful runs, that is, where the agent found the gold:

<table>
<thead>
<tr>
<th>Size</th>
<th>Total Reward</th>
<th>Successful Runs</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 x 5</td>
<td>43,028</td>
<td>45</td>
</tr>
<tr>
<td>6 x 6</td>
<td>32,596</td>
<td>35</td>
</tr>
<tr>
<td>7 x 7</td>
<td>20,276</td>
<td>23</td>
</tr>
<tr>
<td>8 x 8</td>
<td>15,471</td>
<td>18</td>
</tr>
<tr>
<td>9 x 9</td>
<td>12,616</td>
<td>15</td>
</tr>
<tr>
<td>10 x 10</td>
<td>10,012</td>
<td>13</td>
</tr>
</tbody>
</table>

As can be seen from the table, with increasing size it gets more and more difficult for our cautious agent to find the gold.

The following table shows the average time (seconds CPU time of a 1733 MHz processor) it takes for the agent to select an action and to infer the update. The times were obtained by averaging over 100 runs with different scenarios and by dividing the total time for solving a problem by the number of physical actions executed by the agent:

<table>
<thead>
<tr>
<th>Size</th>
<th>Time per Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 x 5</td>
<td>0.0282 sec</td>
</tr>
<tr>
<td>6 x 6</td>
<td>0.0287 sec</td>
</tr>
<tr>
<td>7 x 7</td>
<td>0.0342 sec</td>
</tr>
<tr>
<td>8 x 8</td>
<td>0.0463 sec</td>
</tr>
<tr>
<td>9 x 9</td>
<td>0.0503 sec</td>
</tr>
<tr>
<td>10 x 10</td>
<td>0.0579 sec</td>
</tr>
</tbody>
</table>

The figures show that the program scales well. The slight increase is due to the increasing size of the state when the agent has acquired knowledge of large portions of the (increasingly large) environment. Indeed, the program scales gracefully to environments of much larger size; we also ran experiments with a 50 x 50-grid, placing the gold in the upper right corner and with a sparser distribution of pits (using a probability 0.05 for each square, so that in most cases the agent had to explore large portions of the grid). These scenarios were completed, on the average over 20 runs, within 340 seconds, and the average time for each action was 0.7695 sec.

5 Improvements

The FLUX program presented in this paper encodes a particular, quite simple strategy for a Wumpus World agent. Due to its declarative nature, it should be easy to improve the program in various ways:

1. Our agent tends to make more turns than necessary, because it systematically goes through the possible choices of directions for the next exploration step. The agent should rather check whether the direction it currently faces is among the possible choices, and then simply continue on its way.

2. Since the use of the arrow gives a negative reward, the agent should shoot the Wumpus only if this allows to enter areas that are otherwise inaccessible.

3. With increasing size, it gets increasingly more difficult for a cautious agent to find the gold; the average reward may be increased if the agent ventures to take the occasional step into a square that is not known to be safe.

4. The computational behavior may be tuned by a one-dimensional representation of the grid.

References


