Linköping Electronic Articles in
Computer and Information Science
Vol. 3(1998): nr 8

Decision Theory, the Situation Calculus and Conditional Plans

David Poole
Department of Computer Science
University of British Columbia
2366 Main Mall
Vancouver, B.C., Canada V6T 1Z4
poole@cs.ubc.ca
http://www.cs.ubc.ca/spider/poole

Linköping University Electronic Press
Linköping, Sweden

http:/www.ep.liu.se/ea/cis/1998/008/

Published on June 15, 1998 by
Linköping University Electronic Press
581 83 Linköping, Sweden

Linköping Electronic Articles in
Computer and Information Science
ISSN 1401-9841
Series editor: Erik Sandewall

©*1998 David Poole*
Typeset by the author using LaTeX
Formatted using étendu *style*

Recommended citation:
<Author>. <Title>. Linköping Electronic Articles in
Computer and Information Science, Vol. 3(1998): nr 8.
http://www.ep.liu.se/ea/cis/1998/008/. *June 15, 1998.*

This URL will also contain a link to the author's home page.

The publishers will keep this article on-line on the Internet
(or its possible replacement network in the future)
for a period of 25 years from the date of publication,
barring exceptional circumstances as described separately.

The on-line availability of the article implies
a permanent permission for anyone to read the article on-line,
to print out single copies of it, and to use it unchanged
for any non-commercial research and educational purpose,
including making copies for classroom use.
This permission can not be revoked by subsequent
transfers of copyright. All other uses of the article are
conditional on the consent of the copyright owner.

The publication of the article on the date stated above
included also the production of a limited number of copies
on paper, which were archived in Swedish university libraries
like all other written works published in Sweden.
The publisher has taken technical and administrative measures
to assure that the on-line version of the article will be
permanently accessible using the URL stated above,
unchanged, and permanently equal to the archived printed copies
at least until the expiration of the publication period.

For additional information about the Linköping University
Electronic Press and its procedures for publication and for
assurance of document integrity, please refer to
its WWW home page: http://www.ep.liu.se/
or by conventional mail to the address stated above.

Abstract

This paper shows how to combine decision theory and logical representations of actions in a manner that seems natural for both. In particular, we assume an axiomatization of the domain in terms of situation calculus, using what is essentially Reiter's solution to the frame problem, in terms of the completion of the axioms defining the state change. Uncertainty is handled in terms of the independent choice logic, which allows for independent choices and a logic program that gives the consequences of the choices. As part of the consequences are a specification of the utility of (final) states, and how (possibly noisy) sensors depend on the state. The robot adopts conditional plans, similar to the GOLOG programming language. Within this logic, we can define the expected utility of a conditional plan, based on the axiomatization of the actions, the sensors and the utility. Sensors can be noisy and actions can be stochastic. The planning problem is to find the plan with the highest expected utility. This representation is related to recent structured representations for partially observable Markov decision processes (POMDPs); here we use stochastic situation calculus rules to specify the state transition function and the reward/value function. Finally we show that with stochastic frame axioms, action representations in probabilistic STRIPS are exponentially larger than using the representation proposed here.

1 Introduction

This paper presents a way to combine decision theory, the situation calculus, and conditional plans. It ignores many other issues such as concurrent actions, time, multiple agents, and the derivation of causal rules from domain constraints. This follows from the idea that we want to separately study the orthogonal issues, and try to devise solutions to individual subproblems that are not incompatible with the solutions to other subproblems. The goal is simplicity; the resulting system is simple, as one would hope when trying to combine two fundamental concepts.

The rest of this introduction gives philosophical starting points for this paper. Some of these arguments are standard and are given here to make them open to scrutiny.

1.1 Reasoning about actions

In this paper, we consider reasoning about actions to be about one simple problem: given a model of itself and the world, and some goals (or preferences), what should an agent do?

This is complicated because:

- What an agent should do now depends on what it will do in the future. The only reason I am typing these words now is because I plan to submit this to a journal in the future. The only reason a robot may be going in a particular direction is because it is going to get a key to open a door.

- What an agent will do in the future depends on what it will observe in the future. Only a stupid agent, or one in a very uninteresting environment, would look at the world, decide what to do, and then act without consulting its sensors. If I observe someone has written a related paper to this one, I will change what I write to reflect this. If my robot notices that the door it is getting the key for is already open, or notices that it's path is blocked, it should change what it does to reflect this new information.

- What an agent will observe in the future depends on what it does now. The classic example of this is in medical tests; it's not uncommon for doctors to inflict pain and risk on a patient for the sole purpose of finding information from which they can condition future actions. Even more mundanely an agent will observe different things depending on whether it turns right or left.

Work on reactive robots (Brooks 1986, Brooks 1991) had proposed ignoring the first point; the agent reacts to the environment without considering what it will do in the future. There are many domains for which simple reaction to the environment, without thinking, will not lead the agent to a desirable state. What we can learn from the work on reactive robots is that agents must be able to react (quickly) to the environment. The representation in this paper is not at odds with reactive agents (Poole, Mackworth &

Goebel 1998, Chapter 12), but rather emphasises how to reason about current actions based on thinking about the future.

Classic planning work in AI (Fikes & Nilsson 1971, Yang 1997) has ignored the second point. The idea is to make a linear plan based on assuming what the world is like, and to patch this plan or replan if execution monitoring says that the plan has not worked. However, for virtually every interesting domain there are no actions whose consequences can be predicted based on information known at planning time. When the information needed to predict the consequences of actions will become known at execution time, you can use conditional planning (Manna & Waldinger 1980, Peot & Smith 1992). It would seem that conditional planning lets us solve the complications of planning as set out above; an agent can consider adopting a conditional plan that lets it condition its actions on what it observes.

The traditional view of conditional planning (Manna & Waldinger 1980, Peot & Smith 1992) assumes that the agent can achieve the goal no matter which path through the conditional plan was taken. This assumes perfect sensors and a perfect model of the world (apart from the conditions that will be directly observable during execution) so that you can prove that your conditional plan will reach the goal.

This means that you need to approximate the problem; the effects of actions are *not* completely predictable in the real world. Unexpected things do happen, and you can't always observe all of the conditions that affect the outcome of an action. This is the idea of satisficing (Simon 1996); you need to approximate the problem of finding the best plan to that of finding a good-enough plan.

However both sequential (unconditional) and traditional conditional planning are problematic for a number of reasons:

- Not all failures are born equal. The robot failing to pick up a key is very different from it falling down the stairs. In the first case it can just try again, in the second case you may need to repair or replace the robot (and anything else it fell on). Sometimes it may be worth the risk of falling down the stairs if it has to get past the stairs. At other times it may not. It is important to consider not only the most likely state of affairs, but also deviations from these.

- Ignoring the possible effects that are not the most likely can lead to bad plans. For example, it is usually a good idea to wear a seat belt when driving in a car. However, when we only consider solving a goal, we *never* come up with a plan to wear a seat belt. This is because we don't want it to be a goal state to have an accident while wearing a seat belt (it is usually very easy to achieve having an accident). In fact, we want to avoid having an accident! By approximating the problem we preclude good solutions to the actual problem. This becomes even more ridiculous, when we worry about finding exact solutions to these approximate problems.

- There may not be any normal state of affairs. There are many actions where the outcomes are not completely predictable at all. For exam-

> ple, the effect of picking up a cup often is that everything in the cup remains in it, and that nothing else gets disturbed, but anyone with kids knows that this isn't the *normal* outcome (and robots are not, and won't be for a long time, as adept as kids in picking up cups).

The problem is that any model of a domain is an approximation of the domain. The idea of satisficing is good; to simplify the problem to make it computationally easy to solve. It isn't of much use when the simplified problem isn't easy to solve or when the simplified problem does not lead itself to approximate solutions. It is dangerous when we forget the formalization is only an approximation, and treat it as the real thing.

There is an alternative. To quote Rich Sutton[1]:

> *Approximate the solution, not the problem.*

It may be better to more accurately model the problem and our knowledge and ignorance of the problem (including modelling the approximation caused by the modelling activity itself). This doesn't mean we have to model at the lowest level of detail or that there cannot be a more accurate model of the world, but rather that the model contains a true reflection of the knowledge and ignorance contained in the model. We would also like a model that allows for the existence of good-enough plans (or approximately optimal plans). The specification of a "good enough" plan shouldn't be embedded in the model, but should be usable during inference. Providing a modelling language that lets us model our knowledge of a domain and lets us find approximately optimal plans is the promise of decision-theoretic planning.

1.2 Decision Theory

Bayesian decision theory is one of the simplest, most universally applicable, yet most misunderstood theory about reasoning and acting. Bayesian decision theory specifies what an agent should (decide to) do, given its preferences and partial information about its environment.

The appeal of Bayesian decision theory is based on theorems (Von Neumann & Morgenstern 1953, Savage 1972) that say that under certain reasonable assumptions about preferences, an agent will choose an action that maximizes its expected utility (see Myerson (1991) and Ordeshook (1986) for good introductions). It is normative in the sense that if an agent isn't acting according to the tenants of decision theory, it must be violating one of the assumptions. This result does not mean that an agent has to explicitly manipulate probabilities and utilities, but that its decisions can be rationalized in these terms. For AI researchers building intelligent systems, we can argue that if we want to build a rational agent that acts according to the tenants of decision theory, we should reason directly in terms of probabilities and utilities: if the agent is going to act according to some probabilities and

[1] From *Reinforcement Learning: Lessons for Artificial Intelligence*, A talk presented by Rich Sutton at the 1997 International Joint Conference on Artificial Intelligence Nagoya, Japan, August 28, 1997. `http://www-anw.cs.umass.edu/ rich/IJCAI97/IJCAI97.html`

utilities, we should let it act according to the most reasonable set of probabilities and utilities.

Bayesian decision theory is radical in that it suggests that *all* uncertainty be summarised in terms of probabilities. This includes genuinely stochastic phenomenon, ignorance, partial observability, or simplifications due to modelling assumptions. In all of these cases, probability is a measure of the agent's beliefs. Bayesian decision theory goes against the permissive trend that suggests that we try to integrate many different ways to handle uncertainty[2].

It is important to note that decision theory has nothing to say about representations. Adopting decision theory doesn't mean adopting any particular representation. While there are some representations that can be directly extracted from the theory, such as the explicit reasoning over the state space or the use of decision trees, these become intractable as the problem domains become large; it is like theorem proving by enumerating the interpretations. Adopting logic doesn't mean you have to enumerate interpretations or generate the semantic tree (Chang & Lee 1973), nor does adopting decision theory mean you have to use such representations.

Finally it should be noticed that decision-theoretic planning is very different from probabilistic planning (Kushmerick, Hanks & Weld 1995), where the aim is to find a plan that reaches the goal with probability greater than some threshold. Rather than having a goal, we specify the value of each outcome. It is quite possible that the optimal plan *never* achieves the best-possible goal; the risk in trying to get to that goal may not be worthwhile when compared to another plan that gets to a less-valuable state (e.g., it may not be worth trying to achieve world peace if that entails a risk of killing everyone on Earth).

1.3 Logic and Uncertainty

There are many normative arguments for the use of logic in AI (Nilsson 1991, Poole et al. 1998). These arguments are usually based on reasoning with symbols with an explicit denotation, allowing relations amongst individuals, and quantification over individuals. This is often translated as needing (at least) the first-order predicate calculus. Unfortunately, the first-order predicate calculus has very primitive mechanisms for handling uncertainty, namely the use of disjunction and existential quantification.

If we accept the normative arguments of Bayesian decision theory with

[2]One such theory that has been advocated is Dempster-Shafer theory (Shafer 1976) which could be described as allowing disjunctive assertions about probabilities. This may be useful for theoretical (as opposed to practical) reasoning about other agents, where you can be uncertain about their probability. It doesn't make sense to be uncertain about your own beliefs when your beliefs are exactly a measure of your uncertainty. In practical reasoning where you have to act, you will act according to some probabilities, and these are your beliefs. For an alternative to the view expressed here, the transferable belief model (Smets & Kennes 1994) suggests using belief functions to represent beliefs and then converting them to probabilities for decision making. This is more an argument about representing all updating in terms of Bayesian conditioning. Smets (1991) gives a nice overview of different models of update.

those for logic (and they don't seem to be contradictory), then we have to consider how to handle uncertainty. Bayesian decision theory specifies that all uncertainty be handled by probability.

The independent choice logic (ICL) (Poole 1997, Poole 1998) reconciles Bayesian decision theory with logic. It is designed to include the advantages of logic, but to handle *all* uncertainty using Bayesian decision or game theory.

The idea is, rather than using disjunction to handle uncertainty, to allow agents, including nature, to make choices from a choice space, and use a restricted underlying logic to specify the consequences of the choices. We can adopt acyclic logic programs (Apt & Bezem 1991) under the stable model semantics (Gelfond & Lifschitz 1988) as the underlying logical formalism. This logic includes no uncertainty in the sense that every acyclic logic program has a unique stable model[3]. All uncertainty is handled by independent stochastic mechanisms. A deterministic logic program gives the consequences of the agent's choices and the random outcomes.

What is interesting is that simple logic programming solutions to the frame problem (see Shanahan 1997, Chapter 12) seem to be directly transferable to the ICL which has more sophisticated mechanisms for handling uncertainty than the predicate calculus. I would even dare to venture that the main problems with formalizing action within the predicate calculus arise because of the inadequacies of disjunction to represent the sort of uncertainty we need.

When mixing logic and probability, one can extend a rich logic with probability, and have two kinds of uncertainty: that uncertainty from the probabilities and that from disjunction in the logic (Bacchus 1990, Halpern & Tuttle 1993). An alternative that is pursued in the independent choice logic is to have all of the uncertainty in terms of probabilities.

1.4 Representations of Actions and Uncertainty

The combination of decision theory and planning (Feldman & Sproull 1975) is very appealing. The general idea of planning is to construct a sequence of steps, perhaps conditional on observations that solves a goal. In decision-theoretic planning, this is generalised to the case where there is uncertainty about the environment and we are interested in, not only solving a goal, but what happens under any of the contingencies. Goal solving is extended to the problem of maximizing the agent's expected utility, where the utility is an arbitrary function of the final state (or the accumulation of rewards received earlier).

Recently there have been claims made that Markov decision processes (MDPs) (Puterman 1990) are the appropriate framework for developing decision theoretic planners (e.g., Boutilier, Dearden & Goldszmidt 1995). MDPs, and dynamical system in general (Luenberger 1979) are based on the notion of a **state**: what is true at a time such that the past at that time can only affect

[3]We can conclude either a or $\sim a$ for every closed formula a. This cannot use disjunction to encode uncertainty because $a \vee b$ is only a consequence if one of a or b is. Note that this is a property of the underlying logic, not a property of the ICL.

the future from that time by affecting the state. In terms of probability, the future is independent of the past given the state. This is called the Markov property. In the discrete-time Markovian view, the notion of an action is straightforward: an **action** is a stochastic function from states into states. That is, an action and a state leads to a probability distribution over resulting states. Again, this is the semantics of actions, it doesn't lead to efficient representations.

The naive representation is to represent actions explicitly; for each action and state, give the probability distribution over states. An action can then be represented as a $s \times s$ matrix, where s is the number of states (Luenberger 1979). As you can imagine, this soon explodes for all but the smallest state-spaces.

Artificial intelligence researchers are very interested in finding good representations. We usually think of the world, not in terms of states, but in terms of propositions (or random variables). We would then like to specify actions in terms of how the propositions at one time affect the propositions at the next time. This is the idea behind two slice temporal Bayesian networks (Dean & Kanazawa 1989): we divide the state into random variables and, for each action, write how the random variables at one time affect the random variables at the next time. When the value of a random variable is only affected by a few (a bounded number of) random variables at the previous stage for each action, the complexity is the number of variables times the number of actions. This is a significant improvement over the explicit state-space representation as the state space is exponentially larger than the number of variables (if there are n binary variables, there are $s = 2^n$ states).

This problem is similar to the frame problem (McCarthy & Hayes 1969, Shanahan 1997): how to concisely specify the consequences of an action (and how to effectively use that concise specification computationally). In the frame problem, the assumption is that an action only affects a few propositions. There have been many suggestions as to how to get compact representations of actions under these assumptions (Shanahan 1997). This paper shows how one such representation, the situation calculus (McCarthy & Hayes 1969) can be combined with decision theory.

1.5 Modelling Agents

Another dimension for considering actions is in the capabilities of agents; what sensing they can do, and how they choose which actions to do next. Essentially an agent should be seen as a function of its history (what it has done and what it has observed now and in the past) into its next action. This is known as a transduction (Zhang & Mackworth 1995, Poole et al. 1998). The problem with this as a specification of an agent is that an agent doesn't have access to its history; it only has access to what it can sense and has remembered. There are two traditions on how to implement transductions in agents:

- In the first tradition, agents have internal states (called belief states) and we build agents by constructing a **state transition function** that specifies how the agent's belief state is updated from its previous belief state and its observations, and a **command function** (policy) that specifies what the agent should do based on its observations and belief state (Poole et al. 1998, Chapter 12). In fully observable MDPs, the agent can observe the actual state and so doesn't need belief states. In partially observable MDPs (POMDPs), we assume (noisy) sensors, where the sensor output is a stochastic function of the action and the state. In these models, the belief state is a probability distribution over the actual states of the system, and the state transition function is given by the model of the action and the observation (the value received by the sensor) and Bayes' rule. In between these are agents that have limited memory or limited reasoning capabilities.

- In the second tradition, we can think of agents implementing **robot plans** as in GOLOG (Levesque, Reiter, Lespérance, Lin & Scherl 1997). These plans consider sequences of steps, with conditions, loops, assignments of values to local variables, and other features we expect to find in programming languages. In order to react to the world, we would expect the conditions in the branching to be observations about the world (the values received by potentially noisy sensors) as well as the values of internal variables (Levesque 1996).

Policies (functions from belief state and observations) and plans (composed of primitive actions and built from sequential composition, conditionals and iteration) are different although each can be simulated by the other. A policy can be simulated by an iterative structure over a conditional[4]. A plan can be simulated by having a program counter as part of the state (this is how computers work).

If we are doing exact computation (finding the optimal agent) they should be essentially the same, as they would implement the same transduction. When we are finding approximately optimal agents, they may be very different as a simple plan may not correspond to a simple policy and vice versa.

In this paper we consider simple plans made up of sequential composition and conditionals (conditioning on the output of potentially noisy sensors). Iteration and local variables are explored briefly in Section 2.10. In other work, we have considered the policies within the ICL including multiple agents and noisy sensors (Poole 1997). We have also investigated continuous time in the same framework (Poole 1995).

1.6 The Situation Calculus and the ICL

The independent choice logic (Poole 1997) (an extension of probabilistic Horn abduction (Poole 1993) to include multiple agents and negation as failure) is a simple framework consisting of independent choices make by nature (and potentially other agents) and an acyclic logic program to give the consequences of choices.

[4]In the traditional view of policies, the conditional would be a case statement over all possible states. A while loop over an arbitrary condition would be like the tree-structured policies of (Boutilier et al. 1995); these trees are representations of conditional statements.

In this section we sketch how the situation calculus can be embedded in the ICL. We only need to axiomatise the deterministic aspects in the logic programs; the uncertainty is handled separately. What gives us confidence that we can use simple solutions to the frame problem, for example, is that every statement that is a consequence of the facts that doesn't depend on the atomic choices is true in every possible world. Thus, if we have a property that depends only on the facts and is robust to the addition of atomic choices, then it will follow in the ICL; we would hope than any logic programming solution to the frame problem would have this property. One such property is Clark's completion (Clark 1978), which is true for every predicate defined by the logic program and isn't part of a choice (Poole 1998).

Before we show how to add the situation calculus to the ICL, there are some design choices that need to be made.

- In the deterministic case, the trajectory of actions by the agent up to some time point determines what is true at that point. Thus, the trajectory of actions, as encapsulated by the situation term of the situation calculus (McCarthy & Hayes 1969, Reiter 1991) can be used to denote the state, as is done in the traditional situation calculus. However, when dealing with uncertainty, the trajectory of an agent's actions up to a point, does not uniquely determine what is true at that point. What random occurrences or exogenous events occurred also determines what is true. We have a choice: we can keep the semantic conception of a situation (as a state) and make the syntactic characterization more complicated by perhaps interleaving exogenous actions, or we can keep the simple syntactic form of the situation calculus, and use a different notion that prescribes truth values. We have chosen the latter, and distinguish the *situation* denoted by the trajectory of actions, from the *state* that specifies what is true in the situation. In general there will be a probability distribution over states resulting from a set of actions by the agent. It is this distribution over states, and their corresponding utility, that we seek to model.

  This division means that agent's actions are treated very differently from exogenous actions. The situation terms define only the agent's actions in reaching that point in time. The situation calculus terms indicate only the trajectory, in terms of steps, of the agent and essentially just serve to delimit time points at which we want to be able to say what holds. This is discussed further in Section 3.3.

- None of our representations assume that actions have preconditions; all actions can be attempted at any time. The effect of the actions can depend on what else is true in the world. This is important because the agent may not know whether the preconditions of an action hold, but, for example, may be sure enough to want to try the action.

- When building conditional plans, we have to consider what we can condition these plans on. We assume that the agent has passive sensors, and that it can condition its actions on the output of these sensors. We only have one sort of action, and these actions only affect the world (which includes both the robot and the environment). All we need to do is to specify how the agent's sensors depend on the world. This does not mean that we cannot model information-producing actions (e.g., looking in a particular place) — these information producing actions produce effects that make the sensor values correlate with what is true in the world. The sensors can be noisy; the value they return does not necessarily correspond with what is true in the world (of course if there was no correlation with what is true in the world, they would not be very useful sensors).

2 The Independent Choice Logic

In this section we present the independent choice logic (ICL). The semantic base is the same as that in (Poole 1997, Poole 1998), but the agents are modelled differently. In particular, all of the choices here are controlled by nature.

2.1 Background: Acyclic Logic Programs

We use the Prolog conventions with **variables** starting an upper case letter and **constants**, **function symbols**, and **predicate symbols** starting with lower case letters. A **term** is either a variable, a constant, or is of the form $f(t_1, \ldots, t_m)$ where f is a function symbol and $t_1, \ldots, t_m$ are terms. An **atomic formula** (atom) is either a predicate symbol or is of the form $p(t_1, \ldots, t_m)$ where p is a predicate symbol and $t_1, \ldots, t_m$ are terms. A **formula** is either an atom or is of the form $\sim f$, $f \wedge g$ or $f \vee g$ where f and g are formulae. A **clause** is either an atom or is a **rule** of the form $a \leftarrow f$ where a is an atom and f is a formula (the **body** of the clause). Free variables are assumed to be universally quantified at the level of a clause. A **logic program** is a set of clauses.

A **ground** term is a term that does not contain any variables. A ground instance of a term/atom/clause c is a term/atom/clause obtained by uniformly replacing ground terms for the variables in c. The **Herbrand base** is the set of ground instances of the atoms in the language (inventing a new constant if the language does not contain any constants). A **Herbrand interpretation** is an assignment of **true** or **false** to each element of the Herbrand base. If P is a program, let $gr(P)$ be the set of ground instances of elements of P.

Definition 2.1 (Gelfond & Lifschitz 1988) Interpretation $\mathbf{M}$ is a **stable model**[5] of logic program $\mathbf{F}$ if for every ground atom h, h is true in $\mathbf{M}$ if and only if either $h \in gr(\mathbf{F})$ or there is a rule $h \leftarrow b$ in $gr(\mathbf{F})$ such that b is true in $\mathbf{M}$. Conjunction $f \wedge g$ is true in $\mathbf{M}$ if both f and g are true in $\mathbf{M}$. Disjunction $f \vee g$ is true in $\mathbf{M}$ if either f or g (or both) are true in $\mathbf{M}$. Negation $\sim f$ is true in $\mathbf{M}$ if and only if f is not true in $\mathbf{M}$.

[5]This is a slight generalization of the normal definition of a stable model to include more general bodies in clauses. This is done here because it is easier to describe the abductive operations in terms of the standard logical operators. Note that under this definition $b \leftarrow \sim\sim a$ is the same as $b \leftarrow a$.

Definition 2.2 (Apt & Bezem 1991) A logic program F is **acyclic** if there is an assignment of a natural number (non-negative integer) to each element of the Herbrand base of F such that, for every rule in $gr(F)$ the number assigned to the atom in the head of the rule is greater than the number assigned to each atom that appears in the body.

Acyclic programs are surprisingly general. Note that acyclicity does not preclude recursive definitions. It just means that all such definitions have to be well founded. They have very nice semantic properties, including the following that are used in this paper:

Theorem 2.3 (Apt & Bezem 1991) Acyclic logic programs have the following properties:

1. There is a unique stable model.
2. Clark's completion (Clark 1978) characterises what is true in this model.

Apt & Bezem (1991) give many examples to show that acyclic logic programs are a good representation for models of deterministic state change under complete knowledge.

2.2 Choice Space, Facts and the Semantics

An independent choice space theory is made of two principal components:

Choice space $\mathbf{C}$: a set of sets of ground atomic formulae, such that if χ_1, and χ_2 are in the choice space, and $\chi_1 \neq \chi_2$ then $\chi_1 \cap \chi_2 = \{\}$. An element of a choice space is called a **choice alternative** (or sometimes just an alternative). An element of a choice alternative is called an **atomic choice**.

Facts $\mathbf{F}$: an acyclic logic program such that no atomic choice unifies with the head of a clause.

Definition 2.4 Given choice space $\mathbf{C}$, a **selector function** is a mapping $\tau : \mathbf{C} \to \cup\mathbf{C}$ such that $\tau(\chi) \in \chi$ for all $\chi \in \mathbf{C}$. The **range** of selector function τ, written $\mathbf{R}(\tau)$ is the set $\{\tau(\chi) : \chi \in \mathbf{C}\}$. The range of a selector function is called a **total choice**. In other words, a total choice is a selection of one member from each element of $\mathbf{C}$.

The semantics of an ICL is defined in terms of possible worlds. There is a possible world for each selection of one element from each alternative. The atoms which follow from these atoms together with $\mathbf{F}$ are true in this possible world.

Definition 2.5 Suppose we are given an ICL theory $\langle \mathbf{C}, \mathbf{F} \rangle$. For each selector function τ there is a **possible world** w_τ. We write $w_\tau \models_{\langle \mathbf{C}, \mathbf{F} \rangle} f$, read "$f$ is true in world w_τ based on $\langle \mathbf{C}, \mathbf{F} \rangle$", iff f is true in the (unique) stable model of $\mathbf{F} \cup \mathbf{R}(\tau)$. When understood from context, the $\langle \mathbf{C}, \mathbf{F} \rangle$ is omitted as a subscript of $\models$.

The fact that every proposition is either true or false in a possible world follows from the fact that acyclic logic programs have exactly one stable model.

Note that, for each alternative $\chi \in \mathbf{C}$ and for each world w_τ, there is exactly one element of χ that's true in w_τ. In particular, $w_\tau \models \tau(\chi)$, and $w_\tau \not\models \alpha$ for all $\alpha \in \chi - \{\tau(\chi)\}$.

2.3 Probabilities

The next part of the formalism is a probability distribution over the alternatives[6]. That is, we assume we are given a function

$$P_0 : \cup\mathbf{C} \rightarrow [0,1]$$

such that

$$\forall \chi \in \mathbf{C},\ \sum_{\alpha \in \chi} P_0(\alpha) = 1.$$

The **probability** of a proposition is defined in the standard way. For a finite choice space, the probability of any proposition is the sum of the probabilities of the worlds in which it is true. The probability of a possible world is the product of the probabilities of the atomic choices that are true in the world. That is, the atomic choices are (unconditionally) probabilistically independent. Poole (1993) proves that such independent choices together with an acyclic logic program can represent any finite probability distribution. Moreover the structure of the rule-base mirrors the structure of Bayesian networks (Pearl 1988)[7]. Similarly we can define the **expectation** of a function that has a value in each world, as the value averaged over all possible worlds, weighted by their probability.

When the choice space isn't finite we can define probabilities over measurable sets of worlds. In particular, it suffices to give a measure over finite sets of finite atomic choices (Poole 1993, Poole 1998).

2.4 The ICL$_{SC}$

Within the ICL we can use the situation calculus as a representation for change. Within the logic, there is only one agent, nature, who controls all of the alternatives. These alternatives thus have probability distributions over them. The probabilities are used to represent our ignorance of the initial state and the outcomes of actions. We can then use the situations to reflect the time at which some fluents are true or not.

The following defines what needs to be specified as part of an independent choice logic (for the situation calculus) theory. Note that a possible world defines a complete history. It will specify the truth value for every fluent in every situation. Notice that situations do not appear in this definition. This is analogous to defining the first-order predicate calculus without

[6] In terms of (Poole 1997), all of the alternatives are controlled by nature.

[7] This mapping also lets us see the relationship between the causation that is inherent in Bayesian networks (Pearl 1995) and that of the logical formalisms. See Poole (1993) for a discussion on the relationship, including the Bayesian network solution to the Yale shooting problem and stochastic variants.

any need to define situations. Situations will provide a standard interpretation for some of the terms.

Definition 2.6 An $\mathbf{ICL_{SC}}$ **theory** is a tuple $\langle \mathbf{C}_0, A, \mathbf{O}, P_0, \mathbf{F} \rangle$ where

$\mathbf{C}_0$ called **nature's choice space**, is a choice space.

A called the **action space**, is a set of primitive actions that the agent can perform.

$\mathbf{O}$ the **observables**, is a set of terms.

P_0 is a function $\cup \mathbf{C}_0 \rightarrow [0, 1]$ such that $\forall \chi \in \mathbf{C}_0, \sum_{\alpha \in \chi} P_0(\alpha) = 1$. I.e., P_0 is a probability measure over the alternatives controlled by nature.

$\mathbf{F}$ called the **facts**, is an acyclic logic program such that no atomic choice (in an element of $\mathbf{C}_0$) unifies with the head of any clause.

We model all randomness as independent stochastic mechanisms, such that an external viewer that knew the initial state (i.e., what is true in the situation s_0), and knew how the stochastic mechanisms resolved themselves would be able to predict what was true in any situation. This external viewer, would thus know which possible world was the actual one, and would thus know what is true in every situation. As we don't know the actual world, we have a probability distribution over them. The ICL lets us model this in terms of independent stochastic mechanisms (these are the alternatives with associated probability distributions) and a logic program to give the consequences.

Before we introduce the probabilistic framework we present the situation calculus (McCarthy & Hayes 1969). The general idea is that robot actions take the world from one situation to another situation. We assume there is a situation s_0 that is the initial situation, and a function $do(A, S)$ that given action A and a situation S returns the resulting situation. An agent that knows what it has done, knows what situation it is in. It however does not necessarily know what is true in that situation. The robot may be uncertain about what is true in the initial situation, what the effects of its actions are and what exogenous events occurred.

We use logic (i.e., the facts $\mathbf{F}$) to specify the transitions specified by actions and thus what is true in a situation. What is true in a situation depends on the action attempted, what was true before and what stochastic mechanism occurred. A fluent is a predicate (or function) whose value in a world depends on the situation; we use the situation as the last argument to the predicate (function). We assume that for each fluent we can axiomatise in what situations it is true based on the action that was performed, what was true in the previous state and the outcome of the stochastic mechanisms.

Note that a possible world in this framework corresponds to a complete history. A possible world specifies what is true in each situation. In other words, given a possible world and a situation, we can determine what is true in that situation.

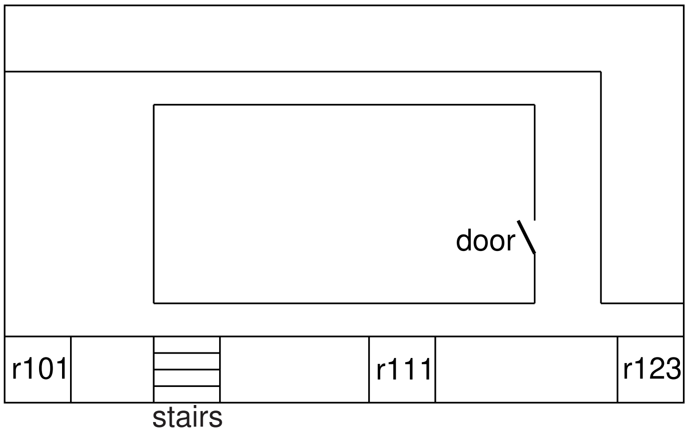

Figure 1: The example robot environment

2.5 An Example Domain

The following ongoing example is used to show the power of the formalism. It is not intended to be realistic.

Example 2.7 Suppose we have a robot that can travel around an office building, pick up keys, unlock doors, and sense whether the key is at the location it is currently at. In the domain depicted in Figure 1, we assume we want to enter the lab, and there is uncertainty about whether the door is locked or not, and uncertainty about where the key is (and moreover the probabilities are not independent). There are also stairs that the robot can fall down, but it can choose to go around the long way and avoid the stairs. The utility of a plan depends on whether it gets into the lab, whether it falls down the stairs and the resources used. The robot starts at $r111$.

Example 2.8 We can write standard situation calculus rules; the only difference is that some of the elements of the body of a rule may be atomic choices. The following rule says that the robot is carrying the key after it has (successfully) picked it up:

$$\begin{aligned}
&carrying(key, do(pickup(key), S)) \leftarrow \\
&\quad at(robot, Pos, S) \wedge \\
&\quad at(key, Pos, S) \wedge \\
&\quad pickup_succeeds(S).
\end{aligned}$$

Here $pickup\_succeeds(S)$ is true if the agent would succeed if it picks up the key and is false if the agent would fail to pick up the key. The agent typically does not know the value of $pickup\_succeeds(S)$ in situation S, or even the position of the key. We would expect that each ground instance of $pickup\_succeeds(S)$ would be an atomic choice. That is

$$\forall S \; \{pickup\_succeeds(S), pickup\_fails(S)\} \in \mathbf{C}_0$$

$P_0(pickup\_succeeds(S))$ reflects how likely it is that the agent succeeds in carrying the *key* given that it was at the same position as the key and attempted to pick it up. For the example below we assume $P_0(pickup\_succeeds(S)) = 0.88$

The general form of a frame axiom specifies that a fluent is true after a situation if it were true before, and the action were not one that undid the fluent, and there was no mechanism that undid the fluent.[8]

Example 2.9 For example, an agent is carrying the key as long as the action was not to put down the key or pick up the key[9], and the agent did not accidentally drop the key while carrying out another action:

$$\begin{aligned}&carrying(key, do(A, S)) \leftarrow\\ &\quad carrying(key, S) \wedge\\ &\quad A \neq putdown(key) \wedge\\ &\quad A \neq pickup(key) \wedge\\ &\quad keeps\_carrying(key, S).\end{aligned}$$

If there were no other clauses for *carrying*, we mean the completion of these two rules (Clark 1978). Thus the agent is carrying the key if and only if one of the bodies is true. Note that this implies that putting down the key always succeeds.

$keeps\_carrying(key, S)$ may be something that the agent does not know whether it is true — there may be a probability that the agent will drop the key. If dropping the key is independent at each situation, we can model this as:

$$\forall S\ \{keeps\_carrying(key, S), drops(key, S)\} \in \mathbf{C}_0$$

The above clause thus forms a stochastic frame axiom. For the example below we assume

$$P_0(keeps\_carrying(key, S)) = 0.95$$

2.6 Axiomatising Utility

Given the notion of an $\mathrm{ICL_{SC}}$ theory, we can write rules for utility. Assume the utility depends on the situation that the robot ends up in and the possible world. In particular we allow for rules that imply $utility(U, S)$, which is true in a possible world if the utility is U for situation S in that world. That is, $utility(U, S)$ means that if the robot stops in situation S it will get utility U. The utility depends on what is true in the state defined by the situation and the world — thus we write rules that imply *utility*. In order to make sure that

[8]This is now a reasonably standard logic programming solution to the frame problem (Shanahan 1997, Chapter 12), (Apt & Bezem 1991). It is essentially the same as Reiter's (1991) solution to the frame problem. It is closely related to Kowalski's (1979) axiomatization of action, but for each proposition, we specify which actions are exceptional, whereas Kowalski specifies for every every action which propositions are exceptional. Kowalski's representation could also be used here.

[9]We want the condition $A \neq pickup(key)$ to cover the case where the agent is carrying the key and tries to pick it up. In this case only the first rule is applicable, and this situation is like the case where the agent is picking up the key. If we didn't have this condition, then the rules would say that the agent is only not carrying the key if both the pickup failed and the robot dropped the key.

we can interpret these rules as utilities we need to have utility being functional: for each situation S, and for each possible world w_τ, there exists a unique U such that $utility(U, S)$ true in w_τ. If this is the case we say the theory is **utility complete**. Ensuring utility completeness can be done locally; we have to make sure that the rules for utility cover all of the cases and there aren't two rules that imply different utilities whose bodies are compatible.

Example 2.10 Suppose the utility is the sum of a prize plus the remaining resources:

$$
\begin{aligned}
& utility(R+P, S) \leftarrow \\
& \quad prize(P, S) \wedge \\
& \quad resources(R, S).
\end{aligned}
$$

The prize depends on whether the robot reached its destination or it crashed. No matter what the definition of any other predicates is, the following definition of *prize* will ensure there is a unique prize for each world and situation:

$$
\begin{aligned}
& prize(-1000, S) \leftarrow crashed(S). \\
& prize(1000, S) \leftarrow in\_lab(S) \wedge \sim crashed(S). \\
& prize(0, S) \leftarrow \sim in\_lab(S) \wedge \sim crashed(S).
\end{aligned}
$$

The resources used depends not only on the final state but on the route taken. To model this we make *resources* a fluent, and like any other fluent we axiomatise it:

$$
\begin{aligned}
& resources(200, s_0). \\
& resources(R - Cost, do(goto(To, Route), S)) \leftarrow \\
& \quad at(robot, From, S) \wedge \\
& \quad path(From, To, Route, Risky, Cost) \wedge \\
& \quad resources(R, S). \\
& resources(R, do(A, S)) \leftarrow \\
& \quad crashed(S) \wedge \\
& \quad resources(R, S). \\
& resources(R - 10, do(A, S)) \leftarrow \\
& \quad \sim gotoaction(A) \wedge \\
& \quad \sim crashed(S) \wedge \\
& \quad resources(R, S). \\
& gotoaction(goto(To, Route)).
\end{aligned}
$$

Here we have assumed that non-goto actions cost 10, and that paths have costs. Note that we are assuming that if the robot has crashed it isn't at any location. Once it has crashed, attempting to do an action doesn't incur any cost (but doesn't achieve anything either).

Paths and their risks and costs are axiomatised using

$$path(From, To, Route, Risky, Cost)$$

that is true if the path from *From* to *To* via *Route* has risk given by *Risky* and costs *Cost*. An example of this relation for our domain is:

$path(r101, r111, direct, yes, 10).$
$path(r101, r111, long, no, 100).$
$path(r101, r123, direct, yes, 50).$
$path(r101, r123, long, no, 90).$
$path(r101, door, direct, yes, 50).$
$path(r101, door, long, no, 70).$

2.7 Axiomatising Sensors

We also need to axiomatise how sensors work. We assume that sensors are passive; this means that they receive information from the environment, rather than *doing* anything; there are no sensing actions. This seems to be a better model of actual sensors, such as eyes, ears, cameras or sonar and makes modelling simpler than when sensing is an action. So called "information producing actions" (such as opening the eyes, moving a camera, performing a biopsy on a patient, or exploding a parcel to see if it is (was) a bomb) are normal actions that are designed to change the world so that the sensors correlate with the value of interest. Note that under this view, there are no information producing actions, or even informational effects of actions; rather various conditions in the world, some of which are under the robot's control and some of which are not, work together to give varying values for the output of sensors.

A robot cannot condition its action on what is true in the world; it can only condition its actions on what it senses and what it remembers (which we don't consider till Section 2.10). The only use for sensors is that the output of a sensor depends, perhaps stochastically, on what is true in the world, and thus can be used as evidence for what is true in the world.

Within our situation calculus framework, we write axioms to specify how sensed values depend on what is true in the world. What is sensed depends on the situation and the possible world. We assume that there is a predicate $sense(C, S)$ that is true if C is sensed in situation S. Here C is a term in our language, that represents one value for the output of a sensor. C is **observable** (that is, $C \in \mathbf{O}$ in Definition 2.6).

Example 2.11 A sensor may be able to detect whether the robot is at the same position as the key. It is not reliable; sometimes it says the robot is at the same position as the key when it is not (a false positive), and sometimes it says that the robot is not at the same position when it is (a false negative). Suppose that noisy sensor *at_key* detects whether the agent is at the same position as the key. Fluent $sense(at\_key, s)$ is true (in a world) if the robot senses that it is at the key in situation s. It can be axiomatised as:

$$\begin{aligned}
& sense(at_key, S) \leftarrow \\
& \quad at(robot, P, S) \land \\
& \quad at(key, P, S) \land \\
& \quad sensor_true_pos(S).
\end{aligned}$$

$$\begin{aligned}&sense(at\_key, S) \leftarrow\\&\quad at(robot, P_1, S) \wedge\\&\quad at(key, P_2, S) \wedge\\&\quad P_1 \neq P_2 \wedge\\&\quad sensor\_false\_pos(S).\end{aligned}$$

The fluent $sensor\_false\_pos(S)$ is true if the sensor is giving a false-positive value in situation S, and $sensor\_true\_pos(S)$ is true if the sensor is not giving a false negative in situation S. Each of these could be part of an atomic choice, which would let us model sensors whose errors at different times are independent.

$$\forall S\ \{sensor\_true\_pos(S), sensor\_false\_neg(S)\} \in \mathbf{C}_0$$
$$\forall S\ \{sensor\_false\_pos(S), sensor\_true\_neg(S)\} \in \mathbf{C}_0$$

Suppose the sensor has a 3% false positive rate and an 8% false negative rate. In the syntax of our implementation, this can be written as

$$random([sensor\_true\_pos(S) : 0.92, sensor\_false\_neg(S) : 0.08]).$$
$$random([sensor\_false\_pos(S) : 0.03, sensor\_true\_neg(S) : 0.97]).$$

where $P_0(sensor\_true\_pos(S)) = 0.92$, and $P_0(sensor\_false\_pos(S)) = 0.03$.

Alternatively, if we had a theory about how sensors break, we could write rules that imply these fluents.

2.8 Conditional Plans

The idea behind the $\mathrm{ICL_{SC}}$ is that agents get to choose situations (they get to choose what they do, and when they stop), and nature gets to choose worlds (there is a probability distribution over the worlds that specifies the distribution of effects of the actions).

Agents get to choose situations, but they do not have to choose situations blind. We assume that agents can sense the world, and choose their actions conditional on what they observe. Moreover agents can have sequences of acting and observing.

Agents do not directly adopt situations, they adopt *plans* or *programs*. In general these programs can involve atomic actions, conditioning on observations, loops, nondeterministic choice and procedural abstraction (Levesque et al. 1997). In this paper we only consider simple conditional plans which are programs consisting only of sequential composition and conditioning on observations (Levesque 1996, Poole 1996)).

Example 2.12 An example of a conditional plan is:

a; if c then b else d; e endIf; g

An agent executing this plan will start in situation s_0, then do action a, then it will sense whether c is true in the resulting situation. If c is true, it will do b then g, and if c is false it will do d then e then g. Thus this plan either selects the situation $do(g, do(b, do(a, s_0)))$ or the situation $do(g, do(e, do(d, do(a, s_0))))$. It selects the former in all worlds where $sense(c, do(a, s_0))$ is true, and selects the latter in all worlds where $sense(c, do(a, s_0))$ is false. Note that each world is definitive on each fluent for each situation. The expected utility of this plan is the weighted average of the utility for each of the worlds and the situation chosen for that world. The only property we need of c is that its value in situation $do(a, s_0)$ will be able to be observed. The agent does not need to be able to determine its value beforehand.

Definition 2.13 A **conditional plan**, or just a **plan**, is of the form

skip
A where A is a primitive action
$P; Q$ where P and Q are plans
if C then P else Q endIf
where C is observable; P and Q are plans

Note that "*skip*" is not an action; the *skip* plan means that the agent does not do anything — time does not pass. This is introduced so that the agent can stop without doing anything (this may be a reasonable plan), and so we do not need an "if C then P endIf" form as well; this would be an abbreviation for "if C then P else *skip* endIf".

Plans select situations in worlds. We can define a relation:

$$trans(P, W, S_1, S2)$$

that is true if doing plan P in world W from situation S_1 results in situation S_2. This is similar to the *DO* macro of Levesque et al. (1997) and the *Rdo* of Levesque (1996), but here what the agent does depends on what it observes, and what the agent observes depends on which world it happens to be in.

We can define the *trans* relation in pseudo Prolog as:

$trans(skip, W, S, S).$
$trans(A, W, S, do(A, S)) \leftarrow$
$\quad primitive(A).$
$trans((P; Q), W, S_1, S_3) \leftarrow$
$\quad trans(P, W, S_1, S_2) \wedge$
$\quad trans(Q, W, S_2, S_3).$
$trans((\text{if } C \text{ then } P \text{ else } Q \text{ endIf}), W, S_1, S_2) \leftarrow$
$\quad W \models sense(C, S_1) \wedge$
$\quad trans(P, W, S_1, S_2).$
$trans((\text{if } C \text{ then } P \text{ else } Q \text{ endIf}), W, S_1, S_2) \leftarrow$
$\quad W \not\models sense(C, S_1) \wedge$
$\quad trans(Q, W, S_1, S_2).$

Now we are at the stage where we can define the expected utility of a plan. The expected utility of a plan is the weighted average, over the set of possible worlds, of the utility the agent receives in the situation it ends up in for that possible world:

Definition 2.14 If our theory is utility complete, the **expected utility** of plan P is[10]:

$$\varepsilon(P) = \sum_{\tau} p(w_\tau) \times u(w_\tau, P)$$

(summing over all selector functions τ on $\mathbf{C}_0$) where

$$u(W, P) = U \text{ if } W \models \mathit{utility}(U, S)$$
$$\quad \text{where } \mathit{trans}(P, W, s_0, S)$$

(this is well defined as the theory is utility complete), and

$$p(w_\tau) = \prod_{\chi_0 \in \mathbf{R}(\tau)} P_0(\chi_0)$$

$u(W, P)$ is the utility of plan P in world W. $p(w_\tau)$ is the probability of world w_τ. The probability is the product of the independent choices of nature.

2.9 Details of our Example

We can model dependent uncertainties. Suppose we are uncertain about whether the door is locked, and where the key is (it could be in room $r101$ or room $r123$), and suppose that these are not independent, with the following probabilities:

$$P(\mathit{locked}(\mathit{door}, s_0)) = 0.9$$
$$P(\mathit{at}(\mathit{key}, r101, s_0) | \mathit{locked}(\mathit{door}, s_0)) = 0.7$$
$$P(\mathit{at}(\mathit{key}, r101, s_0) | \mathit{unlocked}(\mathit{door}, s_0)) = 0.2$$

(from which we conclude $P(\mathit{at\_key}(r101, s_0)) = 0.65$.)

Following the methodology outlined in (Poole 1993) this can be modelled as:

$$\begin{aligned}
&\mathit{random}([\mathit{locked}(\mathit{door}, s_0) : 0.9,\\
&\quad \mathit{unlocked}(\mathit{door}, s_0) : 0.1]).\\
&\mathit{random}([\mathit{at_key_lo}(r101, s_0) : 0.7,\\
&\quad \mathit{at_key_lo}(r123, s_0) : 0.3]).\\
&\mathit{random}([\mathit{at_key_unlo}(r101, s_0) : 0.2,\\
&\quad \mathit{at_key_unlo}(r123, s_0) : 0.8]).\\
&\mathit{at}(\mathit{key}, R, s_0) \leftarrow\\
&\quad \mathit{at_key_lo}(R, s_0) \wedge\\
&\quad \mathit{locked}(\mathit{door}, s_0).\\
&\mathit{at}(\mathit{key}, R, s_0) \leftarrow\\
&\quad \mathit{at_key_unlo}(R, s_0) \wedge\\
&\quad \mathit{unlocked}(\mathit{door}, s_0).
\end{aligned}$$

where $\mathit{random}([a_1 : p_1, \ldots, a_n : p_n])$ means $\{a_1, \ldots, a_n\} \in \mathbf{C}_0$ and $P_0(a_i) = p_i$. This is the syntax used by our implementation.

[10]We need a slightly more complicated construction when we have infinitely many worlds. We need to define probability over measurable subsets of the worlds (Poole 1993, Poole 1998), but that would only complicate this presentation.

We can model complex stochastic actions using the same mechanism. The action *goto* is risky; whenever the robot goes past the stairs there is a 10% chance that it will fall down the stairs.

This is modelled with the choice alternatives:

$$\begin{aligned}&random([would\_fall\_down\_stairs(S) : 0.1,\\&\quad would\_not\_fall\_down\_stairs(S) : 0.9]).\end{aligned}$$

which means

$$\begin{aligned}&\forall S\ \{would\_fall\_down\_stairs(S),\\&\quad would\_not\_fall\_down\_stairs(S)\} \in \mathbf{C}_0\\&\forall S\ P_0(would\_fall\_down\_stairs(S)) = 0.1\end{aligned}$$

These atomic choices are used in the bodies of rules. We can define the propositional fluent *at*:

$$\begin{aligned}&at(robot, To, do(goto(To, Route), S)) \leftarrow\\&\quad at(robot, From, S) \land\\&\quad path(From, To, Route, no, Cost) \land\\&\quad resources(R, S) \land\\&\quad R \geq Cost.\\&at(robot, To, do(goto(To, Route), S)) \leftarrow\\&\quad at(robot, From, S) \land\\&\quad path(From, To, Route, yes, Cost) \land\\&\quad would\_not\_fall\_down\_stairs(S) \land\\&\quad resources(R, S) \land\\&\quad R \geq Cost.\\&at(robot, Pos, do(A, S)) \leftarrow\\&\quad \sim gotoaction(A) \land\\&\quad at(robot, Pos, S).\\&at(X, P, S) \leftarrow\\&\quad X \neq robot \land\\&\quad carrying(robot, X, S) \land\\&\quad at(robot, P, S).\\&at(X, Pos, do(A, S)) \leftarrow\\&\quad X \neq robot \land\\&\quad \sim carrying(robot, X, S) \land\\&\quad at(X, Pos, S).\end{aligned}$$

In those worlds where the path is risky and the agent would fall down the stairs, then it crashes:

$$\begin{aligned}&crashed(do(A, S)) \leftarrow\\&\quad crashed(S).\\&crashed(do(A, S)) \leftarrow\\&\quad risky(A, S) \land\end{aligned}$$

$$\quad \textit{would\_fall\_down\_stairs}(S).$$

$$\textit{risky}(\textit{goto}(\textit{To}, \textit{Route}), S) \leftarrow$$

$$\quad \textit{path}(\textit{From}, \textit{To}, \textit{Route}, \textit{yes}, \_) \wedge$$

$$\quad \textit{at}(\textit{robot}, \textit{From}, S).$$

An example plan is:

> *goto*$(r101, \textit{direct})$;
> if *at_key*
>   then
>   *pickup*$(\textit{key})$;
>   *goto*$(\textit{door}, \textit{long})$
>   else
>   *goto*$(r123, \textit{direct})$;
>   *pickup*$(\textit{key})$;
>   *goto*$(\textit{door}, \textit{direct})$
>   endIf;
> *unlock_door*;
> *enter_lab*

Given the situation calculus axioms, and the choice space, this plan has an expected utility. This is obtained by deriving $\textit{utility}(U, S)$ for each world that is selected by the plan, and using a weighted average over the utilities derived. The possible worlds correspond to choices of elements from alternatives. We do not need to generate the possible worlds — only the explanations (Poole 1998) of the utility and the conditions used in the plans. For example, in all of the worlds where the following are true,

$$\{\textit{locked}(\textit{door}, s_0), \textit{at\_key\_lo}(r101, s_0),$$
$$\quad \textit{would\_not\_fall\_down\_stairs}(s_0),$$
$$\quad \textit{sensor\_true\_pos}(\textit{do}(\textit{goto}(r101, \textit{direct}), s_0)),$$
$$\quad \textit{pickup\_succeeds}(\textit{do}(\textit{goto}(r101, \textit{direct}), s_0))$$
$$\quad \textit{keeps\_carrying}(\textit{key}, \textit{do}(\textit{pickup}(\textit{key}), \textit{do}(\textit{goto}(r101, \textit{direct}), s0)))\}$$

the sensing succeeds (and so the "then" part of the condition is chosen), the prize is 1000, and the resources left are the initial 200, minus the 10 going from $r111$ to $r101$, minus the 70 going to the door, minus the 30 for the other three actions. Thus the resulting utility is 1090. The sum of the probabilities for all of these worlds is the product of the probabilities of the choices made, which is $0.9 \times 0.7 \times 0.9 \times 0.92 \times 0.88 \times 0.95 \approx 0.436$.

Similarly all of the the possible worlds with $\textit{would\_fall\_down\_stairs}(s_0)$ true have prize -1000, and resources 190, and thus have utility -810. The probability of all of these worlds sums to 0.1.

The expected utility of this plan can be computed by enumerating the other cases. We don't have to enumerate the worlds, just the explanations (Poole 1998) of the different values for the utility and the conditional. In particular, we need to explain:

$$\textit{sense}(\textit{at\_key}, \textit{do}(\textit{goto}(r101, \textit{direct}), s0)) \wedge$$
$$\quad \textit{utility}(V, \textit{do}(\textit{enter\_lab}, \textit{do}(\textit{unlock\_door}, \textit{do}(\textit{goto}(\textit{door}, \textit{long}),$$

$$do(pickup(key), do(goto(r101, direct), s0)))))).$$
$$\sim sense(at\_key, do(goto(r101, direct), s0)) \wedge$$
$$utility(V, do(enter\_lab, do(unlock\_door, do(goto(door, direct),$$
$$do(pickup(key), do(goto(r123, direct),$$
$$do(goto(r101, direct), s0))))))).$$

2.10 Richer Plan Language

There are two notable deficiencies in our definition of a plan; these were omitted in order to make the presentation simpler.

1. Our programs do not contain loops.
2. There are no local variables; all of the internal state of the robot is encoded in the program counter.

One way to extend the language to include iteration in plans, is by adding a construction such as

while C do P endDo

as a plan (where C is observable and P is a plan), with the corresponding definition of *trans* being[11]:

$$trans((\text{while } C \text{ do } P \text{ endDo}), W, S_1, S_1) \leftarrow$$
$$W \not\models sense(C, S_1).$$
$$trans((\text{while } C \text{ do } P \text{ endDo}), W, S_1, S_3) \leftarrow$$
$$W \models sense(C, S_1) \wedge$$
$$trans(P, W, S_1, S_2) \wedge$$
$$trans((\text{while } C \text{ do } P \text{ endDo}), W, S_2, S_3).$$

This would allow for interesting programs including loops such as

while *everything_ok* do *wait* endDo

(where *wait* has no effects) which is very silly for deterministic programs, but is perfectly sensible in stochastic domains, where the agent loops until an exogenous event occurs that stops everything being OK. This is not part of the current theory as it violates utility completeness, however, for many domains, the worlds where this program does not halt have measure zero — as long as the probability of failure > 0, given enough time something will always break

Local variables can easily be added to the definition of a plan. For example, we can add an assignment statement to assign local variables values, and allow for branching on the values of variables as well as observations. This (and allowing for arithmetic values and operators) will expand the representational power of the language (Levesque 1996).

[11] Note that we really need a second-order definition, as in (Levesque 1996), to properly define the *trans* relation rather than the recursive definition here. This will let us characterize loop termination.

The addition of local variables will make some programs simpler, such as those programs where the agent is to condition on previous values for a sensor. For example, suppose the robot's sensor can tell whether a door is unlocked a long time before it is needed. With local variables, whether the door is unlocked can be remembered. Without local variables, that information needs to be encoded in the program counter; this can be done by branching on the sense value when it is sensed, and having different branches depending on whether the door was open or not.

3 Comparison with Other Representations

3.1 Probabilistic STRIPS

One of the popular action representations for stochastic actions is probabilistic STRIPS (Kushmerick et al. 1995, Draper, Hanks & Weld 1994, Boutilier & Dearden 1994, Haddawy, Doan & Goodwin 1995). In this section we show that the proposed representation is more concise in the sense that the ICL_{SC} representation will not be (more than a constant factor) larger than the corresponding probabilistic STRIPS representation plus a rule for each predicate, but that sometimes probabilistic STRIPS representation will be exponentially larger than the corresponding ICL_{SC} representation.

It is easy to translate probabilistic STRIPS into ICL_{SC}: using the notation of (Kushmerick et al. 1995), each action a is represented as a set $\{\langle t_i, p_i, e_i\rangle\}$. Each tuple can be translated into the rule of form:

$$b_i(a,S) \leftarrow t_i[S] \wedge r_i[S]$$

($f[S]$ means the state term is added to every atomic formula in formula f), where b_i is a unique predicate symbol, the different r_i for the same trigger are collected into an alternative set, such that $P_0(r_i(S)) = p_i$ for all S. For those positive elements p of e_i, we have a rule:

$$p[do(a,S)] \leftarrow b_i(a,S)$$

For those negative elements $\overline{p}$ of e_i we have the rule,

$$undoes(p,a,S) \leftarrow b_i(a,S)$$

and the frame rule for each predicate:

$$p[do(A,S)] \leftarrow p[S] \wedge \sim undoes(p,A,S).$$

The ICL_{SC} action representation is much more modular for some problems than probabilistic STRIPS, where, as in STRIPS, the actions have to be represented all at once. Probabilistic STRIPS is worse than the ICL_{SC} representation when actions effect fluents independently. At one extreme (where the effect does not depend on the action), consider stochastic frame axioms such as the axiom for *carrying* presented in Example 2.9. In probabilistic STRIPS the conditional effects have to be added to every tuple representing an action — in terms of (Kushmerick et al. 1995), for every trigger that is compatible with carrying the key, we have to split into the cases where the agent drops the key and where the agent doesn't. Thus the probabilistic STRIPS representation grows exponentially with the number of independent stochastic frame axioms: consider n fluents which persist stochastically and independently and the *wait* action, with no effects. The $\mathrm{ICL_{SC}}$ representation is linear in the number of fluents, whereas the probabilistic STRIPS representation is exponential in n. Note that if the persistence of the fluents are not independent, then the $\mathrm{ICL_{SC}}$ representation will also be the exponential in n — we cannot get better than this; the number of probabilities that have to be specified is also exponential in n. In some sense we are exploiting the conciseness of Bayesian networks — together with structured probability tables (Poole 1993) — to specify the dependencies amongst the outcomes.

3.2 MPD and POMDP Representations

The $\mathrm{ICL_{SC}}$ representation is closely related to two slice temporal Bayesian networks (Dean & Kanazawa 1989) or the action networks of (Boutilier et al. 1995, Boutilier & Poole 1996) that are used for Markov decision processes (MDPs). The latter represent in trees what is represented here in rules — see (Poole 1993) for a comparison between the rule language presented here and Bayesian networks. The situation calculus rules can be seen as structured representations of the state transition function, and the rules for utility can be seen as a structured representation of the reward or value function[12]. One problem with the action networks is that the problem representations grow with the product of the number of actions and the number of state variables — this is exactly the frame problem (McCarthy & Hayes 1969) that is solved here using Reiter's solution (Reiter 1991); if the number of actions that affect a fluent is bounded, the size of the representation is proportional the number of fluents (state variables).

In partially observable Markov decision processes (POMDPs), the state of the world isn't observable by the agent. As in this paper, the agent can only observe the values of its sensors. The representation in this paper can be seen as a representation for POMDPs. POMDP researchers (Kaelbling, Littman & Cassandra 1996) have proposed *policy trees*, which correspond to the plans developed here. Boutilier & Poole (1996) exploit the action network representation for finding optimal policies in partially observable MDPs. The general idea behind their structured POMDP algorithm is to use what is essentially regression (Waldinger 1977) on the situation calculus rules to build plans of future actions contingent on observations — policy trees. The difficult part for exact computation is to not build plans that are stochastically dominated[13] (Kaelbling et al. 1996).

[12] At least for finite stage MDPs. Infinite stage MDPs usually use a reward for each time step and the value of a policy is the cumulative reward. Often rewards at future times are discounted compared to immediate rewards (Puterman 1990). This isn't a big distinction when comparing representations, although it is when comparing algorithms.

[13] Intuitively, conditional plan π can be stochastically dominated by a set of conditional plans, if whatever the agent believes (i.e., whatever its probability distribution over states), the expected utility of one of the plans in the set of plans will be greater than or equal to the expected utility of π.

In contrast to (Haddawy & Hanks 1993), we allow a general language to specify utility. Utility can be an arbitrary function of the final state, and because any information about the past can be incorporated into the state, we allow the utility to be an arbitrary function of the history. The aim of this work is not to identify useful utility functions, but rather to give a language to specify utilities.

The use of probability in this paper should be contrasted to that in (Bacchus, Halpern & Levesque 1995). The agents in the framework presented here do not (have to) do probabilistic reasoning. As in MDPs, the probabilistic reasoning is about the agent and the environment. An optimal agent (or an optimal program for an agent) may maintain a belief state that is updated by Bayes' rule or some other mechanism, but it does not have to. It only has to do the right thing. Moreover we let the agent condition its actions based on its observations, and not just update its belief state. We can also incorporate non-deterministic actions.

3.3 Independent Choice Logic and Reactive Policies

There is a conceptually different way to use the ICL to model time and action. Here we can only sketch the idea; see Poole (1997) for details. We only consider discrete time here. See Poole (1995) for a way to handle continuous time (allowing for integration and differentiation with respect to time) using a method similar to the event calculus.

The idea is to represent agents and nature in the same way. For the situation calculus axiomatization above, the single agent was treated quite differently to nature. Symmetry is important when we consider multiple agents.

We represent time in terms of the integers. The fact that the agent attempted an action is represented by a proposition indexed by time. We use a predicate $do(A, T)$ that is true if the agent attempted action A at time T. What is true at a time depends on what was true at the previous times and what actions have occurred, and the outcome of stochastic mechanisms. This places actions by the agent at the same level as actions by nature (or actions by other agents).

There are two parts to axiomatise. The first is to axiomatise the effect of actions, and the second is to specify what an agent will do based on what it observes (i.e., its policy).

To axiomatise the effect of actions, for the discrete time case we write how what is true at one time depends on what was true at the previous time (including what actions occurred). We would write similar axioms to the situation calculus, but indexed by time, and using *do* as a predicate.

Example 3.1 The axiom for carrying of Example 2.8 can be stated as:

$$
\begin{aligned}
& carrying(key, T+1) \leftarrow \\
& \quad do(pickup(key), T) \wedge \\
& \quad at(robot, Pos, T) \wedge \\
& \quad at(key, Pos, T) \wedge \\
& \quad pickup\_succeeds(T).
\end{aligned}
$$

The frame axiom for *carrying* in Example 2.9 would look like:

$$\begin{aligned}
&carrying(key, T+1) \leftarrow \\
&\quad carrying(key, T) \wedge \\
&\quad \sim do(putdown(key), T) \wedge \\
&\quad \sim do(pickup(key), T) \wedge \\
&\quad keeps_carrying(key, S).
\end{aligned}$$

These don't look very different to the situation calculus axioms!

Similarly axioms for sensing that only refer to a single situation/state, such as those of Example 2.11 would remain the same, but the variables are quantified over times, not situations.

This slight change to the representation of the facts has profound effects on the plans. There are no situations. What an agent does is a set of propositions for different times. Within this framework, it is natural to think in terms of agents adopting policies.

What an agent does depends on what it observes and what it remembers. A **policy** is a logic program that specifies what an agent will do based on what it senses and what it has remembered (Poole 1997).

Example 3.2 The following rule could be one part of a policy for the robot:

$$\begin{aligned}
&do(pickup(key), T) \leftarrow \\
&\quad sense(at_key, T) \wedge \\
&\quad recall(want_key, T).
\end{aligned}$$

Here *recall* could be a predicate that represents the internal state of the agent. It can be axiomatised like any other relation.

This rule is very different to a situation calculus program, because it says that whenever the robot senses it is at a key, and wants it, it should pick it up (as well as doing any other actions that are implied by other rules). In order to implement a situation-calculus type plan using such rules, the robot needs to maintain something like a program counter or continuations. In order for a situation calculus program to implement such rules, it has to loop over a conditional statement that checks the conditions of the rules, and does the appropriate concurrent actions.

With axioms about utility, a policy has a utility in a possible world, and so, by averaging over possible worlds it has an expected utility. The goal is to choose the policy with the highest expected utility.

Within the policy-based framework, concurrent actions and multiple agents are easy to represent. The proposed framework here is, like the event calculus, narrative-based (Shanahan 1997) in that it is reasoning about a particular course of events. This is true for each possible world, but we can have a probability distribution over possible worlds. We have a mechanism for allowing multiple agents to choose which events that they can control occur for each context, and to allow a probability distribution over events that nature controls (Poole 1997).

Extending the situation calculus version to multiple agents isn't as straightforward. The way we have treated the situation calculus (and we have tried hard to keep it as close to the original as possible) really gives an agent-oriented view of time — the situations in some sense mark particular time points that correspond to the agent completing its actions. Everything else (e.g., actions by nature or other agents) then has to meld with this division of time. This is even trickier when we realize that when agents have sloppy actuators and noisy sensors, the actions defining the situations correspond to action attempts; the agent doesn't really know what it did, it only knows what it attempted and what its sensors now tell it. When there are multiple agents, either there has to be a common clock, some master agent with which the other agents define their state transition, or complex actions (Reiter 1996, Lin & Shoham 1995). These all mean that the actions need to be carried out lock-step, removing the intuitive appeal of the situation calculus, and making it much closer to the event calculus. The work of Reiter (1996) and Lin & Shoham (1995) assumes a very deterministic world. Not only must the world unfold deterministically, but you must know how it unfolds. This is very different to the assumptions that hold here, where an agent doesn't even know what it has done, only what it has attempted. The work here may show how to reconcile such an omnipotent view with stochastic actions and limited sensing.

4 Conclusion

This paper has presented a formalism that lets us combine situation calculus axioms, conditional plans and Bayesian decision theory in a coherent framework. It is closely related to structured representations of POMDP problems. The hope is that we can form a bridge between work in AI planning and in POMDPs, and use the best features of both. This is the basis for ongoing research.

We are also investigating (Poole 1995, Poole 1997) alternate representations for actions that are much closer to the event calculus. Which will turn out to be a more useful representation is a matter for debate, further research and, eventually, history to determine.

We are betting that decision theory will be eventually seen as the appropriate formal basis for acting under uncertainty (as it is in many disciplines). You can ignore it at the peril of your work becoming irrelevant. Workers in knowledge representation should take heart that the need for knowledge representation won't go away; we will still need good representations and good algorithms.

Acknowledgements

This work was supported by Institute for Robotics and Intelligent Systems, Phase III (IRIS-III), project "Dealing with Actions", and Natural Sciences and Engineering Research Council of Canada Operating Grant OGPOO44121. Thanks to Valerie McRae for proofreading.

References

Apt, K. R. & Bezem, M. (1991). Acyclic programs, *New Generation Computing* **9**(3-4): 335–363.

Bacchus, F. (1990). *Representing and Reasoning with Uncertain Knowledge*, MIT Press, Cambridge, Massachusetts.

Bacchus, F., Halpern, J. Y. & Levesque, H. J. (1995). Reasoning about noisy sensors in the situation calculus, *Proc. 14th International Joint Conf. on Artificial Intelligence (IJCAI-95)*, Montréal, Québec, pp. 1933–1940.

Boutilier, C., Dearden, R. & Goldszmidt, M. (1995). Exploiting structure in policy construction, *Proc. 14th International Joint Conf. on Artificial Intelligence (IJCAI-95)*, Montréal, Québec, pp. 1104–1111.

Boutilier, C. & Poole, D. (1996). Computing optimal policies for partially observable decision processes using compact representations, *Proc. 13th National Conference on Artificial Intelligence*, Portland, OR, pp. 1168–1174.

Boutilier, R. & Dearden, R. (1994). Using abstractions for decision-theoretic planning with time constraints, *Proc. 12th National Conference on Artificial Intelligence*, Seattle, WA, pp. 1016–1022.

Brooks, R. (1986). A robust layered control system for a mobile robot, *IEEE Journal of Robotics and Automation* **2**(1): 14–23.

Brooks, R. A. (1991). Intelligence without reason, *Proc. 12th International Joint Conf. on Artificial Intelligence (IJCAI-91)*, Sydney, Australia, pp. 569–595.

Chang, C. L. & Lee, R. C. T. (1973). *Symbolic Logical and Mechanical Theorem Proving*, Academic Press, New York.

Clark, K. L. (1978). Negation as failure, *in* H. Gallaire & J. Minker (eds), *Logic and Databases*, Plenum Press, New York, pp. 293–322.

Dean, T. & Kanazawa, K. (1989). A model for reasoning about persistence and causation, *Computational Intelligence* **5**(3): 142–150.

Draper, D., Hanks, S. & Weld, D. (1994). Probabilistic planning with information gathering and contingent execution, *Proceedings of the Second International Conference on AI Planning Systems*, Menlo Park, CA, pp. 31–36.

Feldman, J. R. & Sproull, R. F. (1975). Decision theory and artificial intelligence II: The hungry monkey, *Cognitive Science* **1**: 158–192.

Fikes, R. E. & Nilsson, N. J. (1971). STRIPS: A new approach to the application of theorem proving to problem solving, *Artificial Intelligence* **2**(3-4): 189–208.

Gelfond, M. & Lifschitz, V. (1988). The stable model semantics for logic programming, *in* R. Kowalski & K. Bowen (eds), *Proceedings of the Fifth Logic Programming Symposium*, Cambridge, MA, pp. 1070–1080.

Haddawy, P., Doan, A. & Goodwin, R. (1995). Efficient decision-theoretic planning: Techniques and empirical analysis, *in* P. Besnard & S. Hanks (eds), *Proc. Eleventh Conf. on Uncertainty in Artificial Intelligence (UAI-95)*, Montréal, Québec, pp. 229–236.

Haddawy, P. & Hanks, S. (1993). Utility models for goal-directed decision-theoretic planners, *Technical Report 93-06-04*, University of Washington, Department of Computer Science and Engineering.
*ftp://ftp.cs.washington.edu/pub/ai/haddawy-hanks-aij-submission.ps.Z

Halpern, J. & Tuttle, M. (1993). Knowledge, probability, and adversaries, *Journal of the ACM* **40**(4): 917–962.

Kaelbling, L. P., Littman, M. L. & Cassandra, A. R. (1996). Planning and acting in partially observable stochastic domains, *Technical Report CS-96-08*, Department of Computer Science, Brown University.
*http://www.cs.brown.edu/publications/techreports/reports/CS-96-08.html

Kowalski, R. (1979). *Logic for Problem Solving*, Artificial Intelligence Series, North-Holland, New York.

Kushmerick, N., Hanks, S. & Weld, D. S. (1995). An algorithm for probabilistic planning, *Artificial Intelligence* **76**: 239–286. Special issue on planning and scheduling.

Levesque, H. J. (1996). What is planning in the presence of sensing?, *Proc. 13th National Conference on Artificial Intelligence*, Portland, OR, pp. 1139–1146.

Levesque, H. J., Reiter, R., Lespérance, Y., Lin, F. & Scherl, R. B. (1997). GOLOG: A logic programming language for dynamic domains, *Journal of Logic Programming, Special issue on Reasoning about Action and Change* **31**: 59–83.
*ftp://ftp.cs.toronto.edu/pub/cogrob/README.html

Lin, F. & Shoham, Y. (1995). Provably correct theories of action, *Journal of ACM* **42**(2): 283–320.
*ftp://ftp.cs.toronto.edu/pub/cogrob/README.html

Luenberger, D. G. (1979). *Introduction to Dynamic Systems: Theory, Models and Applications*, Wiley, New York.

Manna, Z. & Waldinger, M. (1980). A deductive approach to program synthesis, *ACM Transactions on Programming Languages and Systems* **2**(1): 90–121.

McCarthy, J. & Hayes, P. J. (1969). Some philosophical problems from the standpoint of artificial intelligence, *in* M. Meltzer & D. Michie (eds), *Machine Intelligence 4*, Edinburgh University Press, pp. 463–502.

Myerson, R. B. (1991). *Game Theory: Analysis of Conflict*, Harvard University Press, Cambridge, MA.

Nilsson, N. J. (1991). Logic and artificial intelligence, *Artificial Intelligence* **47**: 31–56.

Ordeshook, P. C. (1986). *Game theory and political theory: An introduction*, Cambridge University Press, New York.

Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*, Morgan Kaufmann, San Mateo, CA.

Pearl, J. (1995). Causal diagrams for empirical research, *Biometrika* **82**(4): 669–710.

Peot, M. A. & Smith, D. E. (1992). Conditional nonlinear planning, *in* J. Hendler (ed.), *Proc. First International Conference on AI Planning Systems (AIPS-92)*, College Park, Maryland, pp. 189–197.

Poole, D. (1993). Probabilistic Horn abduction and Bayesian networks, *Artificial Intelligence* **64**(1): 81–129.

Poole, D. (1995). Logic programming for robot control, *Proc. 14th International Joint Conf. on Artificial Intelligence (IJCAI-95)*, Montréal, Québec, pp. 150–157.
*ftp://ftp.cs.ubc.ca/ftp/local/poole/papers/lprc.ps.gz

Poole, D. (1996). A framework for decision-theoretic planning I: Combining the situation calculus, conditional plans, probability and utility, *in* E. Horvitz & F. Jensen (eds), *Proc. Twelfth Conf. on Uncertainty in Artificial Intelligence (UAI-96)*, Portland, OR, pp. 436–445.
*http://www.cs.ubc.ca/spider/poole/abstracts/iclsc.html

Poole, D. (1997). The independent choice logic for modelling multiple agents under uncertainty, *Artificial Intelligence* **94**: 7–56. special issue on economic principles of multi-agent systems.
*http://www.cs.ubc.ca/spider/poole/abstracts/icl.html

Poole, D. (1998). Abducing through negation as failure: stable models in the Independent Choice Logic, *Journal of Logic Programming* **to appear**.
*http://www.cs.ubc.ca/spider/poole/abstracts/abnaf.html

Poole, D., Mackworth, A. & Goebel, R. (1998). *Computational Intelligence: A Logical Approach*, Oxford University Press, New York.

Puterman, M. L. (1990). Markov decision processes, *in* D. P. Heyman & M. J. Sobel (eds), *Handbooks in OR and MS, Vol. 2*, Elsevier Science Publishers B. V., chapter 8, pp. 331–434.

Reiter, R. (1991). The frame problem in the situation calculus: A simple solution (sometimes) and a completeness result for goal regression, *in* V. Lifschitz (ed.), *Artificial Intelligence and the Mathematical Theory of Computation: Papers in Honor of John McCarthy*, Academic Press, San Diego, CA, pp. 359–380.

Reiter, R. (1996). Natural actions, concurrency and continuous time in the situation calculus, *Proc. Fifth International Conf. on Principles of Knowledge Representation and Reasoning*, Cambridge, MA. *ftp://ftp.cs.toronto.edu/pub/cogrob/README.html

Savage, L. J. (1972). *The Foundation of Statistics*, 2nd edn, Dover, New York.

Shafer, G. (1976). *A Mathematical Theory of Evidence*, Princeton University Press, Princeton, NJ.

Shanahan, M. (1997). *Solving the Frame Problem: A Mathematical Investigation of the Common Sense Law of Inertia*, MIT Press, Cambridge, MA.

Simon, H. (1996). *The Sciences of the Artificial*, third edn, MIT Press, Cambridge, MA.

Smets, P. (1991). About updating, *in* B. D'Ambrosio, P. Smets & P. Bonissone (eds), *Proc. Seventh Conf. on Uncertainty in Artificial Intelligence (UAI-91)*, UCLA, pp. 378–385.

Smets, P. & Kennes, R. (1994). The transferable belief model, *Artificial Intelligence* **66**: 191–234.

Von Neumann, J. & Morgenstern, O. (1953). *Theory of Games and Economic Behavior*, third edn, Princeton University Press, Princeton, NJ.

Waldinger, R. (1977). Achieving several goals simultaneously, *in* E. Elcock & D. Michie (eds), *Machine Intelligence 8: Machine Representations of Knowledge*, Ellis Horwood, Chichester, England, pp. 94–136.

Yang, Q. (1997). *Intelligent Planning: A Decomposition and Abstraction-Based Approach*, Springer–Verlag, New York.

Zhang, Y. & Mackworth, A. (1995). Constraint nets: A semantic model for hybrid dynamic systems, *Theoretical Computer Science* **138**: 211–239.