1 Introduction

This paper presents a way to combine decision theory, the situation calculus, and conditional plans. It ignores many issues such as concurrent actions, time, multiple agents, and the derivation of causal rules from domain constraints. 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 fundamental concepts.

It is important to make it clear that we are using decision theory and the sitution calculus to reason about agents such as robots. The agents themselves follow conditional plans.

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?

Solving this problem 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 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 (this is called "dead reckoning"). 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 its 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 incur risk of injury (ir even death) on a patient for the sole purpose of finding information on which they may base future actions. Even more mundanely an agent will observe different things depending on whether it turns right or left at an intersection.

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 should 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 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 behind classical planning 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). 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 is taken. This assumes perfect sensors and a deterministic model of the world so that you can prove that your conditional plan will reach the goal. This means that you need to approximate the problem; in most problems, 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. Approximating the problem of finding the best plan to that of finding a good-enough plan is the idea of **satisficing** (Simon 1996).

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

- Not all failures are born equal. A 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 the robot has to get past the stairs. At other times it may not be worth the risk. It is important to consider not only the most likely state of affairs, but also deviations from it.

- 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! However, if we assume we won't have an accident, there is no reason to put up with the inconvenience of wearing a seat belt. By approximating the problem by assuming determinism, we preclude good solutions to the actual problem that isn't determistic. 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. For example, 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 lend 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 as well as 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)

[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`

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 maximises 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 tenets 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 (the basic ingredients of decision-theoretic methods), but that its decisions can be rationalised 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 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, and 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 analogous representations.

Finally it should be noticed that decision-theoretic planning is

[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, but see Snow (1998) for problems with this approach. Smets (1991) gives a nice overview of different models of update.

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, Mackworth & Goebel 1998). These arguments are usually based on reasoning with symbols with an explicit denotation, allowing relations amongst individuals, and permitting 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 and 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 1997a, 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 theory (or game theory when there is more than one agent).

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 disjunction is inadequate

[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.

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) (Bellman 1957, Puterman 1994) and partially observable Markov decision problems (POMDPs) (Monahan 1982, Lovejoy 1991, Zhang & Liu 1997, Kaelbling, Littman & Cassandra 1998) are the appropriate framework for developing decision theoretic planners (e.g., Boutilier, Dearden & Goldszmidt 1995, Geffner & Bonet 1998). MDPs, POMDPs and dynamical system in general (Luenberger 1979), are based on the notion of a **state**. The state is the information about what is true at a time such that if we knew the state at some time, knowing the past at that time won't help predict the future from that time. 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 expicit representation of stochastic actions involves providing, for each action and state, the probability distribution over resulting 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 values propositions at one time affect the values of 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 of the action representation 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 (e.g., 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), which is the problem of how to concisely specify the consequences of an action (and how to effectively use that concise specification computationally). In the classic conceptualisation of 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 at each time (what it has done and what it has observed at that time and in the past) into its action at that time. This is known as an **agent function** (Russell & Subramanian 1995, Rosenschein & Kaelbling 1995) or a **transduction** (Zhang & Mackworth 1995, Poole et al. 1998). A transduction is a specification of what an agent does. An agent cannot directly execute a transduction as it doesn't have access to its history; it only has access to what it can sense and what it has remembered. Two prominent traditions on how to implement transductions in agents are:

- In the first tradition, agents have their own internal states (called belief states). 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 anything more in its belief states. In partially observable MDP (POMDP) models (Section 3.1), we assume (noisy) sensors, where the sensor output is a stochastic function of the action and the state. In these models, there is an optimal agent that represents the belief state as a probability distribution over the actual states of the system. 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 the agents that have perfect sensors but remember nothing and the agents that maintain a perfect model (relative to their limited sensors) are agents that have limited memory or limited reasoning capabilities.

- In the second tradition, we can think of agents implementing **robot programs** or **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). Lin & Levesque (1998) showed that any recursive transduction can be represented in a reasonably simple robot programming language.

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 a policy by having a program counter as part of the state (this is how computers work).

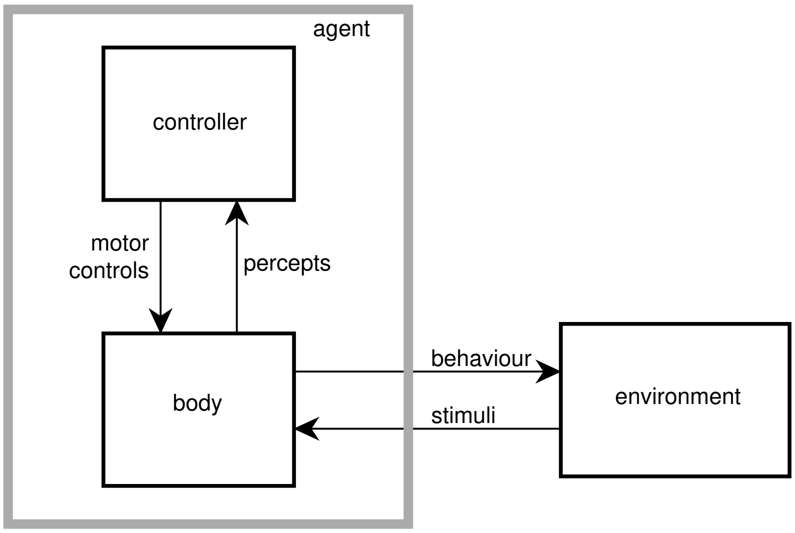
If we are doing exact computation (finding the optimal agent) policies and plans should be essentially the same, as they would implement the same transduction. When we are finding approximately optimal agents, they may be very different as a simple plan may not correspond to a simple policy and vice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

In this paper we consider a simple language for 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 2.11. In other work, we have considered policies within the ICL including multiple agents and noisy sensors (Poole 1997a). We have also investigated continuous time in the same framework (Poole 1995).

1.6 Action Preconditions

It may seem as though the definition of an action is uncontroversial. This is perhaps because most formal theories treat actions as primitive, and assume that the available actions are given. However, when we try to build a robot, we soon find out that there isn't a neat set of actions given to us. To build a satisfactory theory of actions, we need to be very explicit as to exactly what we mean by an action.

[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.



The agent's *body* consists of the sensors and actuators. The *controller* is the part of the agent we are building. The body corresponds to the *vehicle* of Sandewall (1994) and the controller corresponds to the *ego* of Sandewall (1994).

Should *actions* be motor controls (the messages sent from the controller to the actuators), the behaviour (what the agent actually does in the environment), or something else?

Figure 1: A Robotic System

By an action, I mean a particular motor control that is sent by an agent's controller (see Figure 1). Everything else, for example, the agent's behaviour (what the agent actually does or achieves) is part of the effects that we axiomatise. Thus an action is something that is chosen by the controller.

This is sensible as a basis for a theory of action for a number of reasons:

- It is the only thing that a controller can control. For practical reasoning, this is exactly what we want the controller to reason about; namely what it can control. Even if we were not to call these actions, we still want to reason about these commands and their effects. It also seems that if we know the effects of these motor controls under various contingencies, there is nothing else about actions that we need to model for the agent to decide what to do.

- It is something that the agent knows that occurred: that it sent this control command. (The only other thing it really knows is the output messages it receives from its sensors—the percepts of Figure 1.) An agent doesn't know its actual behaviour; only what its motor control was and what its sensors tell it.

- The output of the controller is the one place where the actions can be symbolic. The behaviour of a robot is not symbolic. If we are to give some ontological importance to symbolic actions, then it would seem that the output of a controller is where this can be done.

- Given a controller, this is a well-defined interface. Often the interface between an agent and the environment is arbitrary. Is the hammer held by the robot part of the body? What about the gripper? What if the gripper is detachable, and can be replaced by other components?

- It allows for actions at multiple levels of abstraction.[5] For example, if we have a layered controller (Brooks 1986) (Poole et al. 1998, Section 12.6), we can choose a level of the controller and make the actions at this level.

This view of an action has implications as to what we may mean by the precondition of an action. There are three things that could be meant by a precondition of an action:

- A condition under which the action can be done. For example, Bacchus, Halpern & Levesque (1998) define preconditions as

[5] This point may seem at odds with the previous point: the level of where we decide what the action is is arbitrary. But rather than assuming there is a privileged, objective level of abstraction, we assume there are actions that we can model (and prove relations between) at every level of abstraction.

"necessary and sufficient conditions that characterise when the action is physically possible" (section 2.1).

- A condition under which some effect follows from the action. These are often called *conditional effects*, or *fluent preconditions* (Reiter 1991).
- A condition under which the action should be done. For example, Shanahan (1997) defines a precondition as "the fluents that have to hold when the action is performed for it to be successful" (p. 3). This definition would seem to imply that a precondition is not only a property of the action, but what it means to be successful (i.e., in what the agent is trying to do).

The first notion of precondition seems at odds with treating actions as the motor controls of agents. What happens when the robot sends a motor control when the action isn't physically possible? Presumably whether the action is physically possible is a property of the world (the robot body and the environment), and not a property of the internal state of the agent. But the robot typically doesn't have direct access to the state of the world; it only knows its internal state and the values it receives from its sensors, and these are only linked to the world through noisy sensors and actuators. Thus the agent will have to decide whether to do the action even if it isn't sure whether the action is physically possible. In order to determine if doing the action is the right thing to do, it needs to be able to reason about the effects of the action even when the preconditions don't hold.

It is only the second notion of precondition which we model. This shouldn't really be seen as a feature, but as a necessary evil. It means that we have to axiomatise the effect of an action under all conditions. I do this because, in general, an agent may want to carry out an action even if it doesn't know that the action is "possible" (i.e., even if it isn't sure that the precondition, in the first sense, holds).

The third point, where the action is what should be done, is obtained because we want to build agents that maximise utility (i.e., an agent tries to do as well as it can based on its uncertainty). To do that, we want to be able to evaluate how good arbitrary agents are, not just good ones. In order for an agent to be able to use the third sort of precondition, the preconditions must be knowledge preconditions (i.e., refer to the state of the agent or the outputs of the sensors) and not statements about the world. These are exactly the sorts of policies that are the output of POMDP algorithms, namely what an agent should do depending on what the agent believes.

It would seem as though we need to model the output of the controller (those actions that are *directly executable* (Bacchus et al. 1998) without any preconditions). Parsimony would suggest that if we can do without the other sorts of lower-level deterministic actions (what Bacchus et al. (1998) and (Reiter 1991) seem to call actions), we should. In this paper, we take exactly this approach; we only have directly executable actions, and these have conditional effects. See Section 3.4 for a more detailed comparison with Bacchus et al. (1998).

Thus we equate actions with motor control commands. So what do we do with actions that traditionally have preconditions? For example, in the blocks world, $put\_on(a, b)$, the action of putting block a on block b, classically has preconditions that the robot is holding block a and block b has a clear top. We can see the action $put\_on(a, b)$ as the attempt to put a on b, which has the expected (possibly stochastic) effect of having a on b when a is being held and b has a clear top, but has other effects (such as knocking down the tower containing b) when these preconditions don't hold. In this sense, we can see this as an action attempt; that motor control that would result in the intended effect if the preconditions were to hold. As far as the modeller of the domain is concerned, the preconditions are just one context out of many needed for the conditional effects rules.

1.7 The Situation Calculus and the ICL

The independent choice logic (Poole 1997a) (an extension of probabilistic Horn abduction (Poole 1993) to include multiple agents and negation as failure; see Section 2) is a simple framework consisting of independent choices made by nature (and potentially other agents) and an acyclic logic program to give the consequences of choices.

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 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. 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. The outcomes of random occurrences or exogenous events 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 3.5.

- 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. See Section 1.6.
- 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, the motor control, and these actions only affect the world (the robot body and the environment). We need to specify how the agent's sensors depend on the world, and how the actions affect the world. This does not mean that we cannot model information-producing or sensing 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 1997a, 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/atom/clause is one 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**[6] 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 $\mathbf{M}$.

Definition 2.2 (Apt & Bezem 1991) A logic program F is **acyclic** if there is an assignment of a natural number 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:

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

- There is a unique stable model.

[6]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 (Poole 1998). Note that under this definition $b \leftarrow \sim\sim a$ is the same as $b \leftarrow a$.

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

Facts 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 alternative.

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 choice alternative (i.e., for each total choice). 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.[7] That is, we assume we are given a function

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

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

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).[8] 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 sets of possible worlds defined by finite sets of atomic choices (Poole 1993, Poole 1998).

2.4 The $\mathbf{ICL_{SC}}$

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 associated probability distributions. 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.

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.

[8] 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. See also Section 3.1.2.

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.

Situations do not appear in this definition. This is analogous to defining the first-order predicate calculus without any need to define situations. Situations will provide a standard interpretation for some of the terms. Note that a possible world (see Definition 2.5) 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. It will specify the truth value for every fluent in every situation.

We model all randomness as independent stochastic mechanisms, such that a clairvoyant that knew what was 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 the possible worlds. The ICL lets us model the uncertainty 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 maps to 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 agent 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 **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 the outcomes of the stochastic mechanisms (i.e., what actually happened). 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 we can axiomatise, for each fluent, whether it is true in a situation based on the action that was performed, what was true in the previous situation, and the outcome of the stochastic mechanisms.

2.5 Representational Methodology

In this section we give a representational methodology: how to go about thinking about a domain in order to represent it in the $\mathrm{ICL}_{\mathrm{SC}}$. In the extended example that follows, we will follow this methodology.

Hopefully, it will not be too difficult to transfer the technology to a new example.

When we describe the methodology, we treat frame axioms and ramifications in the same way. We don't need to distinguish them; in fact many of the axioms will be mixes of frame and ramification axioms, referring to both the current situation and to the previous situation. When there are correlated action effects, the distinction between direct effects and ramifications doesn't make much sense. We don't make such a distinction.

The following description is a paraphrase of how we teach students to think about representing knowledge in Bayesian networks. We can see the ICL as a first-order rule-based representation of Bayesian networks (Poole 1993). See Section 3.1 for a more detailed discussion about the relationship to Bayesian networks.

First let's do the propositional (ground) case. We totally order the propositions. We will define each proposition in terms of its predecessors in the total ordering. For each proposition e, a parent context (Poole 1997b) is a conjunction of literals made up of the predecessors of e, such that the other predecessors are contextually independent (Boutilier, Friedman, Goldszmidt & Koller 1996) of e given the context. We find a set of mutually exclusive and covering set $context_1, ..., context_k$ of parent contexts. (The atoms appearing in any of the parent contexts are the parents of e in the corresponding Bayesian network). If e is always true when $context_i$ is true, we write the rule:

$$e \leftarrow context_i$$

If $P(e|context_i)$ isn't 0 or 1, we create a rule:

$$e \leftarrow context_i \wedge ac_i$$

and create an alternative $\{ac_i, nac_i\}$ (where ac_i and nac_i are atoms that don't appear anywhere else), with the probability:

$$P_0(ac_i) = P(e|context_i)$$
$$P_0(nac_i) = 1 - P(e|context_i)$$

If the context is empty (e doesn't depend on any of its predecessors) we don't need to create a rule; we can just make e an atomic choice. When the probability given the context is 0, we don't write any rules.

The case for the ICL with the situation calculus is similar. Intuitively, we make the total ordering of the propositions respect the temporal ordering of situations. We write how a fluent at one situation depends on fluents lower in the ordering at that situation, and on fluents at the previous situation. Note that this rule automatically handles ramifications as well as frame axioms. The total ordering of the fluents guarantees the acyclicity of the rule base and that we don't have circular definitions. The initial situation is handled as any other, but the predecessors in the total ordering are only initial values of fluents (and perhaps atoms that don't depend on the situation).

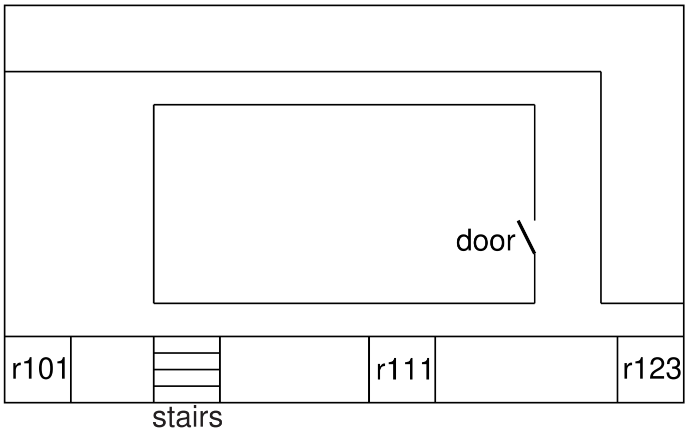


Figure 2: The example robot environment

The only thing peculiar about this is that we often have fluents that depend on values at the current as well as the previous situation. This is important when there are correlated effects of an action; we can't just define each fluent in terms of fluents at the previous situation. But this also means that we treat frame axioms and ramification axioms in exactly the same way.

2.6 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 a key is at the robot's current location. In the domain depicted in Figure 2, we assume the robot wants to enter the lab in the centre, 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 avoid the stairs (which often entails go around the long way between two locations). 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 Let's consider when the robot would be carrying something in a situation. Suppose we have a fluent *carrying* so that $carrying(O, S)$ is true if the robot is carrying O in situation S (for simplicity, we assume the only objects the robot can carry are keys). Suppose carrying doesn't depend on anything at the same time, so we can ask what are the contexts on which $carrying(O, do(A, S))$ depends. It would seem there are three separate contexts where the robot could be carrying O in situation $do(A, S)$:

- $A = pickup(O)$, and the robot and O are at the same position.
- The action isn't $pickup(O)$, the robot was carrying O in situation S, and the action wasn't to put down O.
- The robot was carrying O in situation S, and the action was to put down O. (This is the case where the robot fails to put down the object.)

In no other cases would the robot be carrying O. The following three examples will give rules for each of these three contexts.

Example 2.9 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 a key after it has (successfully) picked it up:

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

Here $pickup\_succeeds(O, S)$ is true if the robot would carry the key O after it picks up the key when the robot and the key are in the same position and is false if the robot would fail to pick up the key in this context. The robot typically does not know the value of $pickup\_succeeds(O, S)$ in situation S, or even the position of the key. We can make each ground instance of $pickup\_succeeds(O, S)$ an atomic choice. That is[9]:

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

$P_0(pickup\_succeeds(O, S))$ reflects how likely it is that the robot 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 that for all terms O and S, $P_0(pickup\_succeeds(O, S)) = 0.88$.

This rule and corresponding alternatives say that the probability that picking up something (in this case a key) actually achieves the robot carrying the object doesn't depend on which particular key is being picked up (nor does it depend on the position, as long as the robot and key are at the same position). Whether the action succeeds (in this case whether the action achieves making the robot carry O) depends on the action, the fluent that the action is to achieve, and

[9]This is a meta-level statement that says that $\mathbf{C}_0$ consists of infinitely many two element sets. The quantification is over terms of the language, not individuals. In our implementation this is written as:

$$random([pickup\_succeeds(O, S) : 0.88, pickup\_fails(O, S) : 0.12]).$$

the context of the rule. Thus, $pickup\_succeeds(O, S)$ should probably be read "pickup succeeds in the robot carrying the key O when the robot and the key are at same position in state S".

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.[10]

Example 2.10 In our ongoing example, the robot is carrying an object as long as the action was not to put down the object or pick up the object,[11] and the robot did not accidentally drop the object while carrying out another action:

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

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

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

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

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

Note that this representation assumes that the probability that the robot keeps carrying an object doesn't depend on the time between actions (i.e., the duration of the situation). This paper does not consider how to combine the situation calculus and time (Reiter 1996) with probabilities.

Example 2.11 The final context in which the robot may be carrying the key is when the action was to put down the key, and the action failed. We can write this as:

$$
\begin{aligned}
& carrying(O, do(putdown(O), S)) \leftarrow \\
& \quad carrying(O, S) \wedge \\
& \quad putdown\_fails(O, S).
\end{aligned}
$$

[10] 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 action which propositions are exceptional. Kowalski's representation could also be used here.

[11] We want the condition $A \neq pickup(O)$ because we have another rule (Example 2.9) to cover the case where the action is to pick up the key, whether or not it was carrying the key at the time. We want that rule to be used rather than this one in the case when it tries to pick up the key and it is carrying the key.

where $putdown\_fails(O, S)$ is an atomic choice, with $P_0(putdown\_fails(O, S))$ the conditional probability that the robot is still carrying O after it has carried out the $putdown(O)$ action.

Note that $putdown(O)$ may succeed in doing other things, it just fails in stopping the robot from carrying O. So $putdown\_fails(O, S)$ should read "putdown fails to stop the robot from carrying O in situation S".

If there were no other clauses for *carrying*, we mean the completion (Clark 1978) of these three rules. Thus the robot is carrying the key if and only if one of the bodies is true.

2.7 Axiomatising Utility

Given the notion of an 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. We need to axiomatise the utility relation, $utility(U, S)$, 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 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 2.12 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:

$$prize(-1000, S) \leftarrow crashed(S).$$

$$prize(1000, S) \leftarrow in\_lab(S) \wedge {\sim}crashed(S).$$

$$prize(0, S) \leftarrow {\sim}in\_lab(S) \wedge {\sim}crashed(S).$$

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:

$$resources(200, s_0).$$

$$resources(R - Cost, do(goto(To, Route), S)) \leftarrow$$

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

Here we have assumed that non-goto actions cost 10, and that paths have costs. 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* (either *yes* or *no*) and costs *Cost*. *Risky* specifies whether the path takes the robot past the stairs. An example of this relation for our domain is:

$$
\begin{aligned}
&path(r101, r111, direct, yes, 10). \\
&path(r101, r111, long, no, 100). \\
&path(r101, r123, direct, yes, 50). \\
&path(r101, r123, long, no, 90). \\
&path(r101, door, direct, yes, 50). \\
&path(r101, door, long, no, 70).
\end{aligned}
$$

2.8 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. 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. 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.

A robot cannot condition its action choice on what is true in the world; it can only condition its action choices on what it senses (the *percepts* of Figure 1) and what it remembers (which we don't consider till Section 2.11). 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 2.13 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 robot 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:

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

The fluent $sensor\_false\_pos(S)$ is true if the *at_key* 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.

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

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

$$
\begin{aligned}
& random([sensor\_true\_pos(S) : 0.92, sensor\_false\_neg(S) : 0.08]). \\
& random([sensor\_false\_pos(S) : 0.03, sensor\_true\_neg(S) : 0.97]).
\end{aligned}
$$

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

2.9 Conditional Plans

The idea behind the $\mathrm{ICL}_{\mathrm{SC}}$ 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 blindly. We assume that agents can sense the world, and choose their actions conditional on what they observe. Agents can have a sequence 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 2.14 An example of a conditional plan is:

$$a; \text{if } c \text{ then } b \text{ else } d; e \text{ endIf}; g$$

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.[12] The agent does not need to be able to determine its value beforehand.

Definition 2.15 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

[12] Recall that c is the output of the sensor, it is the message the agent receives from the sensor. This doesn't imply that the sensor is perfect, just that the agent knows what the sensor output is.

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:

$$trans(P, W, S_1, S_2)$$

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:

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

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 2.16 If our theory is utility complete, the **expected utility** of plan P is:[13]

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

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

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

(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.

[13] 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.

2.10 Details of our Example

Although we only have independent choices, we can use these to model model dependent uncertainties. Suppose we are uncertain about whether, in the initial situation, 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:

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

(from which we conclude $P(at(key, r101, s_0)) = 0.65$.)

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

$$random([locked(door, s_0) : 0.9,$$
$$unlocked(door, s_0) : 0.1]).$$
$$random([at\_key\_lo(r101, s_0) : 0.7,$$
$$at\_key\_lo(r123, s_0) : 0.3]).$$
$$random([at\_key\_unlo(r101, s_0) : 0.2,$$
$$at\_key\_unlo(r123, s_0) : 0.8]).$$
$$at(key, R, s_0) \leftarrow$$
$$at\_key\_lo(R, s_0) \wedge$$
$$locked(door, s_0).$$
$$at(key, R, s_0) \leftarrow$$
$$at\_key\_unlo(R, s_0) \wedge$$
$$unlocked(door, s_0).$$

where $random([a_1 : p_1, \ldots, a_n : p_n])$ 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.

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:

$$random([would\_fall\_down\_stairs(S) : 0.1,$$
$$would\_not\_fall\_down\_stairs(S) : 0.9]).$$

which means

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

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

$$at(robot, To, do(goto(To, Route), S)) \leftarrow$$
$$at(robot, From, S) \wedge$$

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

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

$$
\begin{aligned}
&crashed(do(A, S)) \leftarrow \\
&\quad crashed(S). \\
&crashed(do(A, S)) \leftarrow \\
&\quad risky(A, S) \wedge \\
&\quad would\_fall\_down\_stairs(S). \\
&risky(goto(To, Route), S) \leftarrow \\
&\quad path(From, To, Route, yes, C) \wedge \\
&\quad at(robot, From, S).
\end{aligned}
$$

An example plan is:[14]

$goto(r101, direct)$;
if at_key
  then
    $pickup(key)$;
    $goto(door, long)$
  else
    $goto(r123, direct)$;

[14]Note that the condition at_key is the output of the sensor, as described in Example 2.13.

$$
\begin{array}{l}
\qquad\qquad\qquad pickup(key); \\
\qquad\qquad\qquad goto(door, direct) \\
\qquad\qquad \text{endIf}; \\
\qquad unlock\_door; \\
\qquad enter\_lab
\end{array}
$$

Given the situation calculus axioms, and the choice space, this plan has an expected utility. This is obtained by deriving $utility(U, S)$ 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,

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

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 $would\_fall\_down\_stairs(s_0)$ 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 of the plan (i.e., the sensed value). In particular, we need to explain:

$$
\begin{array}{l}
sense(at\_key, do(goto(r101, direct), s_0)) \wedge \\
\quad utility(V, do(enter\_lab, do(unlock\_door, do(goto(door, long), \\
\qquad do(pickup(key), do(goto(r101, direct), s_0)))))). \\
{\sim}sense(at\_key, do(goto(r101, direct), s_0)) \wedge \\
\quad utility(V, do(enter\_lab, do(unlock\_door, do(goto(door, direct), \\
\qquad do(pickup(key), do(goto(r123, direct), \\
\qquad\quad do(goto(r101, direct), s_0))))))).
\end{array}
$$

2.11 Richer Plan Language

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

- Our programs do not contain loops.
- There are no local variables; all of the internal state of the robot is encoded in the program counter (i.e., where the robot is up to in the plan).

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

while C do P endWhile

as a plan (where C is observable and P is a plan), with the corresponding definition of *trans* being:[15]

$$\begin{aligned}
&trans((\text{while } C \text{ do } P \text{ endWhile}), W, S_1, S_1) \leftarrow \\
&\quad W \not\models sense(C, S_1). \\
&trans((\text{while } C \text{ do } P \text{ endWhile}), W, S_1, S_3) \leftarrow \\
&\quad W \models sense(C, S_1) \wedge \\
&\quad trans(P, W, S_1, S_2) \wedge \\
&\quad trans((\text{while } C \text{ do } P \text{ endWhile}), W, S_2, S_3).
\end{aligned}$$

This would allow for interesting programs including loops such as

while *everything_ok* do *wait* endWhile

(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 is greater than zero, 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 values to local variables, and allow for branching on the values of variables as well as observations. This (and allowing for arithmetic values and operators) will expand the usefulness of the language (Levesque 1996).

The addition of local variables means it is easy to implement belief states explicitly. It 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, the robot can remember whether the door is unlocked. 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.

[15] 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 characterise loop termination.

3 Comparison with Other Representations

3.1 POMDPs and Bayesian nets

Here is a way to motivate this paper that is orthogonal to the presentation of Section 1:

- We start with partially observable Markov decision problems (POMDPs) (Monahan 1982, Lovejoy 1991, Kaelbling et al. 1998) as the underlying model.
- We describe states in terms of propositions or random variables.
- We want to be able to express a locality that some variables only depend on a few other variables; this gives us Bayesian networks (Pearl 1988) or dynamic Bayesian networks (Dean & Kanazawa 1989).
- We not only want to be able to express conditional independence, but contextual independence (Boutilier et al. 1996, Poole 1997b); that some variable is only relevant in certain contexts. Such contextual independencies can be expressed in terms of trees or rules.[16]
- Rules can be extended to first-order (Poole 1993), and even include negation as failure (Poole 1998). This is the basis of the independent choice logic.
- We have a number of choices as to how to represent actions in this framework:
  - An action being carried out at a time can be a proposition. We can then axiomatise what follows from these events (as in the event calculus (Kowalski & Sergot 1986)). An agent can be seen as a function from sensor and action history into actions (or as a set of rules that specify what the agent does under different contingencies). This is the approach taken in (Poole 1997b). It is essentially the approach of influence diagrams (Howard & Matheson 1981).
  - An action can be a term, and we have as propositions that some fluent is true in some situation. This is the approach taken in the current paper.

Note that in the translation of (finite stage) POMDPs into rule form, we haven't made any simplifying assumptions. We can represent any finite stage POMDP. We have however gained the ability to state (and hopefully exploit) conditional and contextual independencies, as well as use a first-order language. It remains an open problem to build

[16] Once we have the rules, we don't even need the network structure (Poole 1993). Often the network structure makes the representation more opaque as it does not explicate in what contexts the different parents are relevant.

algorithms that can exploit those features that we can now represent. Thus the goal is very different to that of Geffner & Bonet (1998), who build a representation that makes many simplifying assumptions in order to test a real algorithm for solving POMDPs.

The following sections consider some of these steps in more detail.

3.1.1 Partially Observable Markov Decision Processes

A Markov decision process (Bellman 1957, Puterman 1994) is defined in terms of

- time points[17] (usually discrete; sometimes finite and sometimes infinite),
- states,
- actions,
- a state transition function that, given a state and an action, returns a probability distribution over resulting states, and
- a reward function that specifies the reward given a state, the action taken, and the resulting state.

A Markov decision problem is a Markov decision process together with an optimality criteria, such as maximizing the accumulated discounted reward or the average reward per time period. A partially observable MDP, or a POMDP (Monahan 1982, Lovejoy 1991, Kaelbling et al. 1998), is an MDP together with a set of observables and an observation function (sensor model) that specifies the probability distribution for the observables for each state and action.

Note that having the agents maintain a belief state (a probability distribution over states of the system) is *not* part of the definition of a POMDP. Rather it is a theorem that says that an optimal agent (ignoring computation time and space) can be obtained by an agent that maintains such a belief state (Astrom 1965). A POMDP model lets us determine the expected utility of any agent (including purely reactive agents that ignore their past or agents that ignore the observations). There is also a theorem that says that (at least for finite horizon, problems) the optimal agent can be described in terms of the conditional plan it adopts. This is the basis of Sondik's (1971) exact algorithm for solving POMDPs and those that have followed it (Kaelbling et al. 1998, Zhang & Liu 1997, Cassandra, Littman & Zhang 1997). The intuition is that an agent is completely determined by what it does, and the only time an agent can change what it does is when it observes something (see Lin & Levesque (1998) for a related discussion).

In the framework of this paper, the agent doesn't (have to) do probabilistic reasoning, it only has to do the right thing (to borrow the title from Russell & Wefald (1991)). The role of the utility is

[17] The different time points are often called *stages*. Thus we talk about infinite stage problems and finite stage problems. We sometimes also talk about the horizon, which is the number of stages, so we have infinite horizon problems and finite horizon problems.

to compare agents. This is important when we consider bounded rationality (Russell 1997, Horvitz 1989) or type-2 rationality (Good 1983), where we must take into account the time of the computation done by the agent in order to compute utility. I wouldn't expect optimal agents to do (exact) probabilistic reasoning at all, because it's too hard! As an extreme example, consider an agent with only a few bits of memory that it can maintain as it is acting. You wouldn't expect it to use these bits to approximate a number, but instead it would probably be better for it to encode (remember) a few salient facts about its past. In order to be able to be extended to model bounded rational agents, the ICL_{SC} is designed to model the effects of the agents actions rather than model the reasoning of the agent.[18]

Even if an agent uses quite a different architecture than maintaining a probability distribution over states, for finite horizon problems, the agent can be described in terms of what it will do based on what information it receives; that is, in terms of a conditional plan. If we have an infinite horizon problem, we may need a more complicated plan representation to represent the optimal policy (Kaelbling et al. 1998, Section 6.7).

Finding good algorithms for POMDPs is an active area of research, for both exact algorithms (Kaelbling et al. 1998, Cassandra et al. 1997, Boutilier & Poole 1996) and approximation algorithms (Zhang & Liu 1997, Geffner & Bonet 1998).

3.1.2 Bayesian Networks

As discussed in Section 1.4, we can describe the states in terms of random variables or propositions, and describe the state transition function, the reward function and the observation function in terms of Bayesian networks.

Not only do we want the variable independence of a Bayesian network, but we also want to exploit contextual independence (Boutilier et al. 1996, Poole 1997b). A convenient way to express contextual independence is to give a set of rules to specify a conditional probability table. Once we have the rules, we can lift the representation to a first-order by the introduction of logical variables. This is the basis for probabilistic Horn abduction[19] (Poole 1993) and its successor, the independent choice logic (Poole 1997a).

[18]This would involve adding explicit time to the model to take into account the computation time of the agent (the thinking time as well as the acting time). This would make the framework much more complicated. I wanted to get the foundations debugged first. There isn't much point in building a monolithic formalism on shaky foundations.

[19]Poole (1993) proved that the same framework could be reached from Bayesian networks plus rule-based conditional probabilities, the possible-worlds definition of Section 2.2, and from an abductive view where the atomic choices are assumable and the probability of any proposition could be obtained from the explanations of the proposition. In the independent choice logic (Poole 1997a) the language is extended to include arbitrary acyclic logic programs, including negation as failure (Poole 1998) as well as choices by various agents.

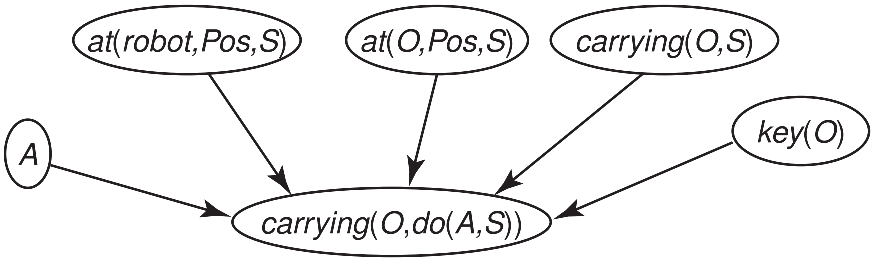


Figure 3: A Bayesian network fragment for the *carrying* fluent

Given this mapping, the rules of Examples 2.9, 2.10 and 2.11, can be represented as the Bayesian network fragment of Figure 3. Note that this is a parametrized Bayesian network fragment; we mean that each grounding of this forms a different part a Bayesian network (Bayesian networks being propositional representations).

Note that, in the grounding, the A variable, that can take the values of all possible actions, will be deterministic; one value will have probability one and the other values will have probability zero. Note that this isn't the probability that the agent does A, but is the probability that the action in its child node is that action; the action in the child node is completely determined in the grounding. Note also that the nodes containing *Pos* need to be combined as an "*or*" for the various values for *Pos*.

This network representation is quite different to the standard ways to represent actions in Bayesian networks. There are two traditional ways to handle actions in Bayesian networks:

- The first is to represent actions in terms of decision nodes as in influence diagrams (Howard & Matheson 1981). These decision nodes represent the proposition that the agent does something at some time. When we are designing an agent (or their policies) we get to choose values for these depending on the information when the decision is make. If we were using this as a representation of another agent, we could have a probability distribution over these actions. The parents of these decision nodes are the information available when the action was performed.

- The second tradition is to treat actions as interventions (Pearl 1995). An action changes the value of a variable externally (effectively cutting the parents from the variable, and giving it a new value).

The actions in the Bayesian network built from fragments like that of Figure 3 are more like hypotheticals. For example, the variable $carrying(key, S)$, for some particular value of S, represents the probability that the agent would be carrying the key if it were to carry out the actions specified by the situation term S. What the agent does isn't specified in the network; the network represents all possible contingencies (in terms of what the agent does) at once.

3.1.3 Abduction

The independent choice logic is essentially abductive: the probability of any proposition (or expected value of any term) can be computed from the explanations of that proposition (Poole 1993, Poole 1998). We can compute the expected utility for a conditional plan by explaining the utility variable and explaining the observations. We can thus think of reasoning in the ICL similarly to Shanahan:

> The key idea of this paper is to consider the process of assimilating a stream of sensor data as abduction. Given such a stream, the abductive task is to hypothesise the existence, shapes, and locations of objects which, given the output the robot has supplied to its motors, would explain that sensor data (Charniak & McDermott 1985, page 455). This is, in essence, the map building task for a mobile robot.
>
> (Shanahan 1998, p. 2)

This is exactly what happens in the framework of this paper. We find the explanations for the observations, as well as the explanations of the final utility. We happen to have probabilities on the assumables, but the whole framework can be considered in terms of abduction. On our case, it is not the explanations themselves that are of interest, but only what they tell us about the expected outcomes.

3.2 Probabilistic STRIPS

One of the popular action representations for stochastic actions is probabilistic STRIPS (Kushmerick et al. 1995, Draper, Hanks & Weld 1994, Haddawy, Doan & Goodwin 1995). In this section we show that the proposed representation is more concise in the sense that the $\mathrm{ICL_{SC}}$ 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 $\mathrm{ICL_{SC}}$ representation.[20]

It is easy to translate probabilistic STRIPS into $\mathrm{ICL_{SC}}$: using the notation of (Kushmerick et al. 1995, p. 247), each action a is

[20] We are assuming that we keep the same actions, and don't invent new actions. It is known that if we can invent new micro-actions representing the individual independent aspects, probabilistic STRIPS can be modelled with only a polynomial increase in representation size (Littman 1997). This is similar to the way actions are represented in Example 3.2.

represented as a set $\{\langle t_i, p_i, e_i\rangle\}$, where each t_i is an expression called the trigger, $0 \leq p_i \leq 1$, and e_i is a set of literals called the effects. The intuition is that when t_i is true, with probability p_i, e_i specifies what primitive literals are changed.

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 S is added to every atomic formula in formula f), where b_i is a unique predicate symbol, and r_i is an introduced atomic choice. 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 each positive element p of e_i, we have a rule:

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

For each negative element $\overline{p}$ 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 $\mathrm{ICL_{SC}}$ action representation is much more modular for some problems than probabilistic STRIPS, where, as in STRIPS, the actions have to be represented in one step for each context. Probabilistic STRIPS is worse than the $\mathrm{ICL_{SC}}$ representation when actions affect 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 2.10. 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 the action representation 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.

3.3 MDP 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 using trees what is represented here using 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.[21] One problem with the action networks is that the 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 those plans that are stochastically dominated (Kaelbling et al. 1996).[22]

In contrast to (Haddawy & Hanks 1993), we allow a general language to specify utility. The aim of this work is not to identify useful utility functions, but rather to give a language to specify utilities.

3.4 Bacchus, Halpern and Levesque

The closest work to this paper is the combination of probability and the situation calculus of Bacchus, Halpern and Levesque (1995, 1998),[23] which we will refer to as BHL. At the top level we can be seen as trying to do the same thing. When we consider the detailed design choices, we find that we have made the opposite choice in virtually every decision.

We can contrast this work with that of BHL by the design choices:

[21] 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 1994). This isn't a big distinction when comparing representations, although it is when comparing algorithms.

[22] 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.

[23] Historical note: I have only seen the February 1998 draft of Bacchus et al. (1998).

- In their introduction, they explicitly contrast their work with the work starting from Bayesian networks. In particular, they claim "...[Bayesian networks] have difficulties in dealing with features like disjunction ..." (Bacchus, Halpern & Levesque 1995). I take a Bayesian perspective that disjunction is not a feature we want. As outlined in Section 1.2, what defines a Bayesian is that probability is a measure of belief, that any proposition can have a probability (both of which I would expect that BHL would agree with) and that all uncertainty should be measured by probability (which they don't agree with).

- We differ as to what are primitive actions. In this paper, the primitive actions are the motor control commands that are the output of the controller. These have no preconditions. BHL have analogous actions that are "directly executable by the agent". These directly executable actions in their framework are non-deterministic programs, built from more primitive deterministic actions that do have preconditions. This is discussed in more detail below.

- In order to get the completion that is implicit in many of the solutions to the frame problem to work, we need to be able to say that some effect is true after an action if and only if some conditions are true before the action. In order to be able to say this, we have to have already resolved the nondeterminism. BHL do this completion in terms of their primitive actions, which have already resolved all of the nondeterminism. They then have a probability distribution over the primitive actions, which lets them handle uncertainty. I do this completion in each possible world, where a possible world has resolved all of nature's choices. I then have a probability distribution over possible worlds.

- We also make different design decisions with respect to whether the situation should match the state (see Section 1.7). BHL keep the semantic conceptualization of a situation as a state. In their framework a situation specifies what is true, but an agent doesn't know what situation it is in. In my framework, an agent knows what situation it is in, but the situation doesn't fully specify what is true.

- We also differ as to whether the probabilistic reasoning is internal or external to the agent. In BHL, the probabilistic reasoning is internal to the agent. They axiomatise how the robots beliefs change as it carries out actions and senses the world. The agents in the framework presented here do not (have to) do probabilistic reasoning. As in POMDPs, 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. (See Section 3.1).

- BHL explicitly have sensing actions, where there is no such distinction between sensing and other actions in this paper. (See Section 1.7.)
- Part of the motivation for the independent choice logic (and the Bayesian network community) is to find useful independence assumptions that are both natural and can be exploited for computational gain. The ICL can represent the independence of Bayesian networks, as well as contextual independence and noisy-ors.[24] BHL use a general language for which one can state whatever independence assumptions one likes.

BHL need at least two different sorts of actions: the primitive actions and the directly executable actions. They also distinguish action preconditions, and the preconditions of the effect axioms (what Reiter (1991) calls fluent preconditions). However these distinctions seem arbitrary, as the following example shows:

Example 3.1 Suppose there are two people who want to axiomatise a domain using the framework of BHL. Suppose they make different modelling decisions:

- The first person decides there is a primitive action a, with precondition p and conditional effects: when c is true, the effect is e_1, and when $\neg c$ is true, the effect is e_2.
- The second person decides to model the domain with two primitive actions a_1 and a_2. Action a_1 has precondition $p \wedge c$ and the (unconditional) effect is e_1. Action a_1 has precondition $p \wedge {\sim} c$ and the effect is e_2.

Whenever the first person uses action a, the second person uses the nondeterministic choice between a_1 and a_2.

Which representation would be better? It turns out that it makes absolutely no difference which way the domain is axiomatised. All of the agents choices (as long as $p \not\equiv true$, the agent cannot choose any of these primitive actions as they are not direct executable) will result in exactly the same effects. In some very strong sense these two representations should be seen as equivalent.

I would argue that, (at least in the context of probabilistic actions), rather than being a fundamental property of actions, the

[24]There is nothing in the ICL that forces the bodies of rules to be disjoint. The methodology of Section 2.5 was presented with disjoint rule bodies just to keep the framework simple, and because such rules are sufficient for many applications. When the bodies are not disjoint, they are combined using an "or", in exactly the same was as Clark's (1978) completion. This is also the basis for the noisy-or (Pearl 1988, p. 184) of Bayesian networks.

distinction between action preconditions and fluent preconditions is largely illusory. Rather than trying to maintain an arbitrary distinction between action preconditions and fluent preconditions, I propose that we get rid of action preconditions, and only have conditional effects (fluent preconditions).

We have a straightforward notion of what an action is, and one simple mechanism for handling effects and ramifications. What is an action in BHL isn't so well defined; it seems a though they need to embed the effects as part of the actions, as the following example shows.

Example 3.2 Consider an action that has a number of independent[25] stochastic effects. For example, administering a drug may have independent effects of curing the disease, causing an allergic reaction, and raising blood pressure. Exploiting the independence is important to compactly represent the knowledge. Because we have probabilities associated with effects, we can easily represent the independent effects of the single action *administer-the-drug*, with three alternatives: one whether the drug cures the disease, one whether the drug causes an allergic reaction, and one whether the drug raises the blood pressure. However, BHL associate probabilities with actions. While BHL don't have built-in independence assumptions, they do have the mechanisms to state them. To make explicit the probabilities that need to be stated to be independent, BHL would use three concurrent actions: the action of the drug curing the disease, the action of the drug causing the allergic reaction, and the action of the drug raising the blood pressure. Rather than this formalism having a clean notion of fluent preconditions, it would seems that the notion of what is an action is intertwined with what are the fluents.

Note that the modelling of actions of this example is similar to the construction of Littman (1997, Theorem 3), who shows how probabilistic STRIPS can represent independently evolving actions by creating such micro-actions.

One major advantage of the ICL_{SC} is that exactly the same mechanism can be used for direct effects and ramifications. The general case is somewhere between these two extremes; where the truth of a fluent depends on some fluents at the current situation and some at the previous situation. This occurs, for example, when there are correlated action affects. BHL needs a different mechanism to handle ramifications than they have for action effects.

3.5 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 (1997a)

[25] If the effects aren't independent, we can't do better than considering the combinatorial effects, as the number of independent parameters that needs to be specified is exponential in the number of state variables.

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.

For the discrete time case, we represent time in terms of the integers. The fact that the agent did 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). Note that many actions can be done at any time step.

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 3.3 The axiom for carrying of Example 2.9 can be stated as:

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

The frame axiom for *carrying* in Example 2.10 would look like:

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

The rule covering the case where the putdown fails (Example 2.11) becomes:

$$\begin{aligned}
&carrying(O, T+1) \leftarrow \\
&\quad do(putdown(O), T) \wedge \\
&\quad carrying(O, T) \wedge \\
&\quad putdown_fails(O, T).
\end{aligned}$$

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 2.13 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 1997a).

Example 3.4 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 mechanisms to allow multiple agents to choose which of the actions that they can control occur, and to allow a probability distribution over events that nature controls (Poole 1997a).

Extending the situation calculus version presented here to multiple agents isn't as straightforward. The way we have treated the situation calculus gives an agent-oriented view of time — the situations in some sense mark particular time points that correspond to the agent attempting its actions. Everything else (e.g., actions by nature or other agents) has to meld with this division of time. This is even trickier when we realise that when agents have sloppy actuators and noisy sensors, the actions defining the situations correspond to action attempts. While the agent knows what it did (i.e., its motor controls), and what its sensors tell it, other agents don't know what this agent has done; they only know what their sensors tell them. 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, complex actions (Reiter 1996, Lin & Shoham 1995), or perhaps even a form of concurrent asynchronous situations (where each agent has its own division of time defined by its situations). These seem to 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 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 its behaviour only its motor control.

4 Conclusion

This paper has presented a formalism that combines situation calculus axioms, conditional plans and Bayesian decision theory in a coherent framework. The paper is proposing a new model; not an alternative to POMDPs, Bayesian networks, logic programs, and the standard situation calculus, but a way to combine them. The representation 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 alternate representations for actions that are much closer to the event calculus (Poole 1995, Poole 1997a). 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 Bayesian decision theory will be eventually seen as the appropriate formal basis for acting under uncertainty (as it is in many disciplines). 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 to exploit the representations.

Acknowledgments

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, Craig Boutilier, Holger Hoos, Michael Horsch and Alan Mackworth for comments on earlier drafts. Thanks to Uffe Kjærulff, Ray Reiter, Hector Geffner, and Chitta Baral for thoughtful comments in the online commentary on this paper. Particular thanks to Ray Reiter for pointing out the recent work of Bacchus et al. (1998) and explaining the relationship to this paper.

References

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

Astrom, K. (1965). Optimal control of Markov decision problems with incomplete state estimation, *J. Math. Alan. Applic.* **10**: 174–205.

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.

Bacchus, F., Halpern, J. Y. & Levesque, H. J. (1998). Reasoning about noisy sensors and effectors in the situation calculus, *Artificial Intelligence* . To appear.
*http://www.lpaig.uwaterloo.ca/˜fbacchus/on-line.html

Bellman, R. (1957). *Dynamic Programming*, Princeton University Press, Princeton, NJ.

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., Friedman, N., Goldszmidt, M. & Koller, D. (1996). Context-specific independence in Bayesian networks, *in* E. Horvitz & F. Jensen (eds), *Proc. Twelfth Conf. on Uncertainty in Artificial Intelligence (UAI-96)*, Portland, OR, pp. 115–123.

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.

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.

Cassandra, A., Littman, M. & Zhang, N. (1997). Incermental pruning: A simple, fast, exact mehtod for partially observable markov decision processes, *in* D. Geiger & P. Shenoy (eds), *Proc. Thirteenth Conf. on Uncertainty in Artificial Intelligence (UAI-97)*, pp. ??–??

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

Charniak, E. & McDermott, D. (1985). *Introduction to Artificial Intelligence*, Addison-Wesley, Reading, MA.

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.

Geffner, H. & Bonet, B. (1998). High-level planning and control with incomplete information using POMDPs, *Proceedings AIPS-98 Workshop on Integrating Planning, Scheduling and Execution in Dynamic and Uncertain Environments.*
*http://www.ldc.usb.ve/ hector/

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.

Good, I. (1983). *Good Thinking: The foundations of Probability and Its Applications*, University of Minnesota Press, Minneapolis, MI.

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.

Horvitz, E. J. (1989). Reasoning about beliefs and actions under computational resource constraints, *in* L. Kanal, T. Levitt & J. Lemmer (eds), *Uncertainty in Artificial Intelligence 3*, Elsevier, New York, pp. 301–324.

Howard, R. A. & Matheson, J. E. (1981). Influence diagrams, *in* R. A. Howard & J. Matheson (eds), *The Principles and Applications of Decision Analysis*, Strategic Decisions Group, CA, pp. 720–762.

Kaelbling, L., Littman, M. & Cassandra, A. (1998). Planning and acting in partially observable stochastic domains, *Artificial Intelligence* **101**: 99–134.

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.

Kowalski, R. & Sergot, M. (1986). A logic-based calculus of events, *New Generation Computing* **4**(1): 67–95.

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. & Levesque, H. (1998). What robots can do: robot programs and effective achievability, *Artificial Intelligence* **101**: 201–226.

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

Littman, M. (1997). Probabilistic propositional planning: Representations and complexity, *Proc. 14th National Conference on Artificial Intelligence*, pp. 748–754.

Lovejoy, W. (1991). A survey of algorithmic methods for partially observable Markov decision processes, *Annals of Operations Research* **28**: 47–66.

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.

Monahan, G. E. (1982). A survey of partially observable Markov decision processes: Theory, models and algorithms, *Management Science* **28**: 1–16.

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.
*http://www.cs.ubc.ca/spider/poole/abstracts/lprc.html

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. (1997a). 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. (1997b). Probabilistic partial evaluation: Exploiting rule structure in probabilistic inference, *Proc. 15th International Joint Conf. on Artificial Intelligence (IJCAI-97)*, Nagoya, Japan, pp. 1284–1291.
*http://www.cs.ubc.ca/spider/poole/abstracts/pro-pa.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. (1994). *Markov Decision Processes: Discrete Stochastic Dynamic Programming*, John Wiley and Sons, New York.

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

Rosenschein, S. J. & Kaelbling, L. P. (1995). A situated view of representation and control, *Artificial Intelligence* **73**: 149–173.

Russell, S. (1997). Rationality and intelligence, *Artificial Intelligence* **94**: 57–77.

Russell, S. J. & Subramanian, D. (1995). Provably bounded-optimal agents, *Journal of Artificial Intelligence Research* **2**: 575–609.

Russell, S. & Wefald, E. (1991). *Do the Right Thing: Studies in Limited Rationality*, MIT Press, Cambridge, MA.

Sandewall, E. (1994). *Features and Fluents: The Representation of Knowedge about Dynamical Systems*, Vol. 1, Clarendon Press, Oxford.

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.

Shanahan, M. (1998). A logical account of the common sense informatic situation for a mobile robot, *Electronic Transactions on Artificial Intelligence* **Received**.
*http://www.ida.liu.se/ext/etai/ra/rac/010/

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.

Snow, P. (1998). The vulnerability of the transferable belief model to Dutch books, *Artificial Intelligence* **105**: 345–354.

Sondik, E. (1971). *The optimal Control of Partially Observable Markov processes*, PhD thesis, Department of Electrical Engineering, Stanford University, Stanford, CA.

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, N. & Liu, W. (1997). A model approximation scheme for planning in partially observable stochastic domains, *Journal of Artificial Intelligence Research* **7**: 199–230.

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