Modeling action, knowledge and control

Hector Geffner[1] and **Jacques Wainer**[2]

Abstract. We formulate a model for reasoning about actions, knowledge and control in the presence of incomplete information and sensing operations. In the model, actions can be non-deterministic, sensors can report the value of either symbols, terms or formulas, and knowledge is constructively defined in terms of observations. The model is related to possible worlds accounts of actions and knowledge but its syntax and semantics are simpler and may be found more useful for the design of actual planners. The model is also related to probabilistic accounts of actions and belief (POMDPs) and can be seen as their logical counterpart. We have used variations of this model for the analysis and synthesis of controllers for various tasks.

1 INTRODUCTION

In AI there have been two main approaches to the problem of planning with incomplete information: a procedural approach, focused on extensions of the Strips language and algorithms (e.g., [5, 3]), and a logical approach, focused on extensions of logic for handling actions, knowledge and control (e.g., [11, 18, 9]). In general the two approaches haven't come together yet, and thus planners with incomplete information that both work and have a solid foundation are not common. Part of the problem, we believe, is that models of action and knowedge have been too complex, and as a result they haven't served of much guidance in the design of actual planners. In this paper we aim to contribute to bridge this gap by formulating a different model of action and knowledge that is simpler but no less rigorous or meaningful. We have used variations of this framework in the analysis and automatic synthesis of controllers (see [14] and [6, 1]). Here we present the main elements of the approach and its connections to other approaches.

2 ACTIONS

The model of actions comprises a *language* for describing actions and initial conditions in the form of a logical theory, and a *semantics* that maps such theories into sets of possible state trajectories s_0, s_1, s_2, ... where each s_i represents the state of the world at time i. A key element in the model is the interpretation of action rules as constraints over state transitions, an element that is common to most recent approaches to the frame problem (e.g., [7, 15, 16]).

[1] Dpto. de Computación, Universidad Simón Bolívar, Caracas 1080-A, Venezuela.
[2] Instituto de Computação, Universidade Estadual de Campinas, Caixa Postal 6176, 13083-970 Campinas SP, Brazil.

ECAI 98. *13th European Conference on Artificial Intelligence*
Edited by Henri Prade
Published in 1998 by John Wiley & Sons, Ltd.

The language is a simplified first-order typed language that involves constant, function and predicate symbols but does not involve variables and quantification. We assume that all symbols have *types* and the way symbols get combined into *terms*, *atoms* and *formulas* is standard except that as in any typed language, the types of the symbols are taken into account. That is, if $f_{\alpha\beta}$ is a function symbol with type $\alpha\beta$, meaning that $f_{\alpha\beta}$ denotes a function that takes objects from a *domain* D_α into the *domain* D_β, then $f_{\alpha\beta}(t)$ is a legal term when t is a term of type α and the type of $f_{\alpha\beta}(t)$ is β. For simplicity, we assume in the presentation that the arity of function and predicate symbols is one, but of course all definitions apply to the general case by interpreting t, α and D_α as *tuples* of terms, types and domains respectively. We say that a domain D_α is *named* when it is specified by a set of constant symbols that we call *names*. In that case, it's assumed that the symbols are the names of the different objects in D_α. Thus the domain D_{Blocks} for $Blocks = \{block_1, block_2, \ldots, block_n\}$ is assumed to have n objects of type $Block$ each with a name $block_i$.

Symbols are divided according to certain semantical properties into four classes:

1. *fixed symbols* refer to symbols whose denotation is assumed fixed and known across time
2. *fluent symbols* refer to symbols whose denotation tends to persist but is not necessarily known
3. *action symbols* are used to denote actions
4. *random symbols* have denotations that change randomly

Examples of fixed symbols are symbols like '3', '+', '=', ..., and all the *names*.[3] A symbol *temperature* will normally be a fluent symbol, *move* an action symbol, and *noise_level* a random symbol. Random symbols are the only novelty in this ontology and they will provide the 'noise' that will make the dynamics of action theories *non-deterministic*.

All action symbols are *predicate symbols,* and for the sake of simplicity, we assume that all fluent symbols are *function symbols.* Constant symbols like *temperature* are expressed as fluent symbols of arity 0, while relational fluents like *near* are expressed by means of fluent symbols of type $\alpha\beta$ where β is the boolean type.

Terms and formulas are defined in the standard way provided the compatibility among types. By *state formulas* we mean formulas composed of fixed and fluent symbols only, and by *action formulas*, formulas possibly composed of action symbols as well. Finally an *action* is an atom of the form $p(t)$ where p is an action symbol and t is a tuple of terms not involving random symbols.

[3] That is, names are assumed to be *rigid designators* [8].

2.1 Action Theories

The language is used to define *action theories* representing the effects of actions. An action theory is a triplet $T = \langle D, A, C\rangle$, where D is a domain theory composed of a set of *action rules* and *preconditions*, A is a set of *actions occurrences*, and C is a set of *state formulas* characterizing the initial conditions.

Action rules are rules of the form $B \rightarrow f(t) := t_1$ where B is an action formula, $f(t)$ and t_1 are terms of the same type, and f is a fluent symbol. Rules of this form express how the denotation of f changes after situations in which B is true. Note that actions symbols can appear in any form in the antecedent of action rules. Indeed, such antecedents may contain negated actions, conjunctions of actions, or no actions at all. In the latter case, the rules will apply at all times.

We often abbreviate collections of rules by means of *schemas.* For example, if *move* is an action that accepts arguments of type *Block* and D_{Block} is a named domain specified by the names $\{block\_1, block\_2, \ldots, block\_n\}$, then the effect of *move* in the blocks world can be expressed by the schemas:

$$\begin{array}{lcl} move(x,y) & \rightarrow & \mathsf{clear}(y) := \mathbf{false} \\ move(x,y) & \rightarrow & \mathsf{loc}(x) := y \\ move(x,y) & \rightarrow & \mathsf{clear}(\mathsf{loc}(x)) := \mathbf{true} \end{array}$$

Each schema is an abbreviation of all the rules that can be obtained by replacing the meta-variables x, y, ... by each of the block names. The schemas can be read as saying that after moving block x on top of y, y will no longer be clear, the *new* location of x will be y, and the block that was the *old* location of x becomes clear. This behavior will follow from understanding the rules as constraints on state transitions and interpreting all symbols, except the outer occurrence of f in $f(\cdot) := \ldots$, as referring to their 'old' values. The semantics will make this explicit.

Action preconditions express conditions that need to be true for an action to be *executable.* For example, to buy something, you need to have enough money; to grab something, you need to have an empty hand; etc. Action preconditions are expressed as *action formulas.* For example, if $buy \supset \mathsf{have\_money}$ is an action precondition formula, then the action *buy* will be executable only in situations in which have_money is true. The action preconditions for *move* in the blocks world can be expressed by

$$move(x,y) \supset \mathsf{clear}(x) = \mathbf{true} \wedge \mathsf{clear}(y) = \mathbf{true} \wedge x \neq y$$

Action occurrences. An action occurrence is an expression of the form $p(t)[i]$ where $p(t)$ is an action and i is a time index $i \in [0, 1, \ldots]$. More generally we define a state temporal language with atoms given by expressions of the form $F[i]$ for state formulas F closed under the boolean connectives. We use such language only for queries.

Initial Conditions. The initial conditions C consist of a set of state formulas. Such formulas can express general constraints (e.g., that no block can be sitting on top of itself; $loc(block) \neq block$), or specific observations about the initial state (e.g., $loc(block_1) = block_2$). In the first case, the semantics guarantees that the constraints are satisfied in the initial states. Yet it's up to the modeler to avoid writing rules that can lead to the violation of such constraints in the future. This is like in Strips, where states with atoms like $on(A, A)$ are ruled out only by the action rules and the initial conditions.

2.2 Semantics

The meaning of an action theory is a set of state trajectories $s_0, s_1, s_2, \ldots$, where each s_i represents the state of the system at time i.

Formally, a *state* s is a mapping that assigns to each fixed and fluent symbol x of type α a denotation $x^s \in D_\alpha$. The denotation of fixed symbols is the same in all states and is assumed to be known.[4] Likewise, *noise* refers to a mapping n that assigns each random symbol r of type α a denotation $r^n \in D_\alpha$. The mappings s and n can be composed into a mapping $s + n$ that assigns a denotation to all fixed, fluent and random symbols. Similarly, for *a set of actions* a, $s + a$ represents a mapping that extends the mapping s to all *action symbols.* For an action symbol p, $p^{s+a}(o)$ is defined as **true** if there is an action $p(t)$ in a such that $o = t^s$, and **false** otherwise.

By the standard compositional rules, s implicitly assigns a truth value to all *state formulas,* $s + a$ assigns a truth value to all *action formulas,* and $s + n$ assigns a denotation to all *terms.*

2.2.1 Models

For a set of actions a, let us define the *set* $\text{NEXT}(s_i, a)$ of *possible successors* of a state s_i in an action theory T as the collection of states s_{i+1} such that for all fluent symbols f and some noise assignment n, $f^{s_{i+1}}(v) = t_1^{s_i+n}$ if there is a rule $B \rightarrow f(t) := t_1$ such that $B^{s+a} = \mathbf{true}$ and $v = t^s$, and $f^{s_{i+1}}(v) = f^{s_i}(v)$ otherwise (persistence).

The *models* of $T = \langle D, A, C\rangle$ are the state trajectories s_0, s_1, ..., such that s_0 satisfies the initial conditions C and $s_{i+1} \in \text{NEXT}(s_i, A(i))$, where $A(i)$ is the set of actions $p(t)$ such that $p(t)[i] \in A$.

We say that a trajectory $s_0, s_1, \ldots$ *satisfies* a temporal formula $F[i]$, when F is true in s_i, and that $F[i]$ *follows* from T if F is true in the states s_i of all models $s_0, s_1, \ldots$ of T.

2.2.2 Executability

We are interested in action theories that describe actions that can be safely executed without violating preconditions or introducing inconsistencies. Let us say that a set of actions a is *executable in a state* s, if $s + a$ satisfies all action precondition formulas and $\text{NEXT}(s, a)$ is non-empty. Let us also say that a set of actions *a is executable at time i in T* iff a is executable in all states s_i of all models s_0, s_1, ... of T. Finally, let us say that *an action theory* $T = \langle D, A, C\rangle$ *is executable* if every i the set of actions $A(i)$ is executable at time i in T.

One reason to focus on action theories that are executable is to comply with the 'causality principle' in system theory [12] by which *exogenous actions should not affect beliefs about the past*[5]

[4] In implementations such as [6, 1] the denotation of fixed (function) symbols is enforced by tying such symbols with actual functions in the underlying language.

[5] See also [17].

Theorem 1 (Causality Principle) *Let T be an executable action theory and let T' be the result of adding to T a set of actions $p(i)$ executable in T. Then a formula $F[k]$, $k < i$, follows from T* if and only if *$F[k]$ follows from T_i.*

Paradoxes as the one arising in the Yale Shooting Problem in which an action at some time (e.g., *shoot*[3]) has an influence over the value of a fluent at an earlier time (e.g., *loaded*[2]) are automatically ruled out by this property. The same holds true for most recent models of action [7, 15, 16]. The proposed model is closest to [7] but deals explicitly with time, accommodates a richer language for action, and makes room for compound terms and formulas, and non-determinism.

3 POLICIES

In the action theories considered above, the actions and their times of occurrences must be explicitly given. For many scenarios this is reasonable but in others it's not. In many cases, all we have is a description of a plan or a policy to follow and the actions that they determine may depend on the context.

Let us thus define a (control) *policy* as a function that maps states into sets of actions. That is, given a state s, a policy π_i will produce a set of actions $\pi_i(s)$. Moreover, let us define a *generalized* policy π as a sequence of policies π_0, π_1, π_2, ..., one for each time point $i = 0, 1, \ldots$. We say that a generalized policy π is *stationary* if the functions π_i are all identical, and in such case, we use π to denote them. Otherwise we say that the policy π is non-stationary.[6]

The definitions and results of the previous section can be easily generalized from action theories $T_A = \langle D, A, C\rangle$ to *control* theories $T_\pi = \langle D, \pi, C\rangle$: the set $\text{NEXT}(s_i, a(i))$ of possible successors of a state s_i given the actions $A(i)$ becomes the set $\text{NEXT}(s_i, \pi_i(s_i))$, while the condition for the executability of π in T_π becomes that in all models s_0, s_1, ... of T_π, for all i, the actions $\pi_i(s_i)$ are executable in s_i.

Control policies provide a very general framework for analyzing the effect of *sequential plans*, *condition-action rules*, and even *contingent plans.* Indeed, in order to determine the effects of a sequential plan $[a_0, a_1, a_2, \ldots, a_n]$ we just need to consider the non-stationary policy π such that $\pi_i(s) = \{a_i\}$, for each $i = 1, \ldots, n$. Similarly, a given set of condition-action pairs $c_k \mapsto a_k$, $k = 1, \ldots, n$, where c_k's are state formulas and a_k's are actions, can be represented by the stationary policy π, where $\pi(s)$ is the set of all actions a_k such that c_k is true in s. Finally, contingent plans encoded as programs (e.g., [9]) can be expressed as non-stationary policies, although the characterization of such policies in terms of their program representation is not as direct and will not concern us here.

Example 1 Consider a control policy for inserting a nail into a wall expressed by the condition-action pair $\mathsf{h} > 0 \mapsto hit$ where h is the distance to go, and the effects of *hit* are modeled by the action rules:

$$\begin{aligned} hit \ \wedge \ \mathsf{h} \geq 1 \quad &\rightarrow \quad \mathsf{h} := e \cdot \mathsf{h} \\ hit \ \wedge \ \mathsf{h} > 0 \ \wedge \ \mathsf{h} < 1 \quad &\rightarrow \quad \mathsf{h} := 0 \end{aligned}$$

where e is a random symbol with domain $[0, 0.9]$. Let us assume that h is initially positive. The reader can verify that in all models s_0, s_1, ... of the resulting control theory, $\mathsf{h} = 0$ becomes true in some state s_i for finite i and remains so for all successor states. In other words, the control policy achieves the goal $\mathsf{h} = 0$.

4 KNOWLEDGE

The conditions above for determining the executability of a policy presume that states are *fully observable.* We deal now with the *partially observable* case. For that we need to model what the agent believes at different times.

We model *belief states* in a standard way as *sets of states* that the agent deems possible [8, 11]. We say that a *state expression* x, that can be either a symbol, a term or a formula, is *known to have a certain value* v in a belief state Bel if $x^s = v$ for all states $s \in Bel$.

We also say that x is *known* in Bel if for every pair of states s and s' in Bel, $x^s = x^{s'}$. Note that 'F known to be true' implies that 'F is known', but not the other way around.

We want to address the question of when an agent with beliefs Bel will *know* how to execute a policy π. We say in that case that π is *executable* in Bel.

A tentative answer is that π is executable in Bel if for all states s and s' in Bel, $\pi(s) = \pi(s')$. This is not entirely adequate however because actions $p(t)$ and $p(t')$ that look different syntactically may be referring to the same action, as when $t = t'$ is known to be true in Bel. At the same time, the equality $t = t'$ may be known to be true but neither t or t' may be known individually. For these reasons, let us introduce the *primitive* actions as the actions $p(t)$ where t is a tuple of *names.* Let us also say that two actions $p(t)$ and $p(t')$ are equivalent in Bel when $t = t'$ is known to be true in Bel, and that two sets of actions A_1 and A_2 are equivalent in Bel when for each action in A_1 there is an equivalent action in A_2, and vice versa, for each action in A_2 there is an equivalent action in A_1. Then the executability of a policy can be defined as:

Definition 1 *A policy π is executable in a belief state Bel iff $\pi(s)$ for all $s \in Bel$ is equivalent to a unique set P of primitive actions that are executable in s.*

A result of this definition is that when π is executable in Bel, π does not depend on the individual states in Bel but on Bel itself.[7] For this reason we refer to the set of actions P above as $\pi(Bel)$.

The introduction of actions that are primitive is common to other proposals in the literature (e.g., [11, 9]). Actually, in [9] only primitive actions are allowed in plans. Non-primitive actions however may be very convenient and are allowed here.

5 OBSERVATIONS

We focus now on the conditions under which a generalized policy $\pi = \pi_0, \pi_1, \ldots$ can be executed in the context of a control theory $T = \langle D, \pi, I\rangle$. For this we need to determine the possible belief states $Bel(i)$ for each time $i = 0, 1, \ldots$, and that requires knowing *what will be observable* and *when.* For example, we may want to say that Mary's phone number will be observable after looking up her name in the phone book.

[6] The terminology is from Markov Decision Processes [13].

[7] This is explicit in the theory of Partially Observable Markov Decision Processes or POMDPs[19, 2].

We will represent that a state expression x is *observable* after doing certain actions when certain conditions hold, by means of *observation* or *knowledge gathering rules* [18] of the form:

$$B \rightarrow \mathbf{obs}(x) \quad (1)$$

where B is an action formula, x is either a symbol, a term, or a formula, and **obs** is a special symbol with a special interpretation. For example the scenario above can be encoded by the schema:

$$lookup(pers) \wedge listed(pers) \rightarrow \mathbf{obs}(phone\#(persn))$$

Given a control theory $T = \langle D, \pi, I, K\rangle$ extended with a set K of observation rules, we let $O(s, a)$ stand for the set of expressions x that are *observable* in the state s after doing action a:

$$O(s,a) \stackrel{\text{def}}{=} \{x | B \rightarrow \mathbf{obs}(x) \ \in O \ \text{ and } B^{s+a} = \mathbf{true}\} \quad (2)$$

The *observations* o in the states s after doing a are the mappings that assign to each expression $x \in O(s,a)$ the denotation $x^o = x^s$. Clearly, when observations provide only *partial* information, many states can give rise to the same observation. That is, an agent that ends up in the state s after doing an action a may get an observation o that won't allow him to distinguish s from s' if $x^s = x^{s'}$ for all observables x, i.e., $x \in \mathsf{O}(s,a)$. Indeed, if Bel and Bel_a^o denote the belief states before and after doing an action a and observing o, the latter can be expressed in terms of the former as follows:

$$Bel_a^o = \{s' | s' \in \text{NEXT}(s,a) \text{ for } s \text{ in } Bel \ \ \& \ x^{s'} = x^o\} \quad (3)$$

That is, the possible states after doing the action a in Bel are the states that can follow from a and some state $s \in Bel$ that are compatible with the observations. Moreover, the set of possible observations o are given by the mappings $x^o = x^{s'}$ over all $x \in \mathsf{O}(s', a)$, for each of the states $s' \in \text{NEXT}(s,a)$ that can follow a state $s \in Bel$ after the action a. For each such o, there is a next belief state Bel_a^o and the collection of all such belief states is referred to as $\text{NEXT}(Bel, a)$. The possible ways in which beliefs may evolve can then be described as follows.

Definition 2 *The sequences of belief states* Bel_0, Bel_1, Bel_2, ... *that are possible given a control theory* $T = \langle D, \pi, C, K\rangle$ *are the ones in which* Bel_0 *is the set of states that satisfy the initial conditions* C, *and* $Bel_{i+i} \in \text{NEXT}(Bel_i, \pi_i(Bel_i))$ *for* $i = 0, 1, \ldots$.

The notation $\pi_i(Bel_i)$ presumes that the policy π_i is executable in Bel_i. Non-executable theories are ill-formed and do not need to be considered; they either violate action preconditions, introduce inconsistencies or assume more information than what's provided by the sensors.

We say that a control theory T makes a state expression x known at time i when x is known in the belief states Bel_i of all possible belief state sequences Bel_0, Bel_1, ...; that T makes a formula F true at i when F is known to be true in all such belief states Bel_i, and finally that T achieves F when T makes F true at all times $i > j$, for some time $j \geq 0$.

Example 2 Consider the following plan for a robot in an assembly line [4]: inspect whether the coming widget is blemished, and discard it if flawed but dispatch it otherwise. It is assumed that as long as the widget is not painted, flaws translate into observable blemishes. The rules in the theory are assumed to be the following:[8]

$$\begin{aligned} \neg painted \wedge flawed \ &\rightarrow \ blemished \\ \neg painted \wedge \neg flawed \ &\rightarrow \ \neg blemished \\ inspect \ &\rightarrow \ \mathbf{obs}(blemished) \end{aligned}$$

We are interested in determining whether the policy that corresponds to the plan 'inspect and if flawed reject else dispatch' is executable when the initial condition is $\neg painted$; that means whether the action *inspect* makes *blemished* known in that context. The corresponding non-stationary policy $\pi = \{\pi_0, \pi_1, \ldots\}$ is such that $\pi_0(s) = \{inspect\}$ for all s, and $\pi_1(s) = \{reject\}$ for s such that $flawed^s = \mathbf{true}$ and $\pi_1(s) = \{dispatch\}$ for s such that $flawed^s = \mathbf{false}$. Then $\pi_i(s) = \{\}$ for all $i > 1$ and s.

Let's first analyze the possible transitions from s_0 to s_1. Given that *painted* is initially false, from the first two rules we can conclude that if *flawed* is true in s_0, *blemished* will be true in s_1, and similarly, if *flawed* is false in s_0, *blemished* will be false in s_1. In addition, the fluent *flawed* does not change and hence *flawed and blemished will be either both true at* s_1 *or both false.*

Let's now consider the transition of beliefs. The initial belief state $Bel = Bel_0$ is given by all the states that are compatible with the initial condition $\neg painted$. The action *inspect*, which does not have influence on any fluent, makes the value of *blemished* at time 1 observable. Since this is a binary fluent, two observations o and o' are possible: one that reveals *blemished* true, the other that reveals *blemished* false. As a result, there will be two belief states $Bel_1 = Bel_a^o$ and $Bel_1' = Bel_a^{o'}$ with $a = inspect$ and $Bel = Bel_0$; the former containing all states s_1 in which blemished is true, the second containing all states s_1' in which blemished is false. Yet since *flawed* and *blemished* will be either both true at s_1 or both false, *flawed* will be known at time 1. As a result the policy π_1 'reject if flawed and dispatch otherwise' is executable, and so is the generalized policy $\pi = \{\pi_0, \pi_1, \ldots\}$.

6 RELATED APPROACHES

6.1 Probabilistic Approaches

The model of action and knowledge above is closely related to probabilistic accounts.[9] The equation for Bel_a^o above (3) is indeed the qualitative counterpart of the equation used for belief update in POMDPs, a class of models of sequential decision problems involving uncertain actions and noisy sensors [19, 2]. It's worth noting that while probabilistic frameworks such as POMDPs accommodate explicit representations of belief, they never have to check what belief state sequences are possible. This is because policies are *defined* as functions of the states of belief, and therefore are always executable. In addition, in probabilistic frameworks one does not usually *predict* the possible observations, but gets the *actual* observations by interacting with the real or simulated world. As a result there

[8] The first two rules do not involve any actions and play the role of 'ramification' constraints. For a more faithful encoding of this problem, see [1].

[9] The same can be said about [10] which is an epistemic extension of the language $\mathcal{A}$ [7].

is always a *single* belief sequence that is determined by the actual observations, and which is computed incrementally at *running time*.

6.2 Possible World Approaches

Recent possible world models of action and knowledge such as [18, 9] represent beliefs by means of a binary accessibility relation K on situations where $K(sit', sit)$ means that sit' is deemed a possible situation from sit. Since situations in such versions of the situation calculus are state-action sequences, we can roughly translate such accessibility relations as $K(\mathsf{w'}_i, \mathsf{w}_i, \mathsf{a}_i)$, meaning that $\mathsf{w'}_i = s'_0, s'_1, \ldots, s'_i$ is a possible state trajectory up to time i when the world has evolved according to the trajectory $\mathsf{w}_i = s_0, s_1, \ldots, s_i$ after having done the actions $\mathsf{a}_i = a_0, a_1, \ldots, a_{i-1}$.[10] From this accessibility relation K, the *belief state* that results at time i when the trajectory w_i and the actions a_i are assumed as the 'actual' ones, is simply the set of states s'_i in all trajectories $\mathsf{w'}_i = s'_0, s'_1, \ldots, s'_i$ such that $K(\mathsf{w'}_i, \mathsf{w}_i, \mathsf{a}_i)$ holds.

While our approach is based on the recursion (3) that establishes how beliefs Bel change into beliefs Bel_a^o after doing an action a and observing o, the approach in [18, 9] is based on a corresponding recursion on the accessibility relation K on trajectories, which in our language could be expressed as:

$$K(\mathsf{w'}_{i+1}, \mathsf{w}_{i+1}, \mathsf{a}_{i+1}) \textbf{ iff } K(\mathsf{w'}_i, \mathsf{w}_i, \mathsf{a}_i),\ \ s'_{i+1} \in \text{NEXT}(s_i, a_i)$$
$$\textbf{and } \ x^{s'_{i+1}} = x^{s_{i+1}} \text{ for all } x \in O(s_{i+1}, a_i)$$

meaning that a trajectory $\mathsf{w}'_{i+1} = s'_0, s'_1, \ldots, s'_{i+1}$ will be deemed possible after the 'world' has evolved according to the trajectory $\mathsf{w}_{i+1} = s_0, s_1, \ldots, s_{i+1}$ after $\mathsf{a}_{i+1} = a_0, \ldots, a_i$, if the trajectory $\mathsf{w}'_i = s'_0, s'_1, \ldots, s'_i$ was deemed possible from $\mathsf{w}_i = s_0, s_1, \ldots, s_i$ and the new state s'_{i+1} is compatible with both the action a_i and the observations $x^{s_{i+1}}$ for $x \in O(s_{i+1}, a_i)$.

In [18], a similar condition is expressed directly in the *object* language by encoding the binary relation K as a relational fluent in the situation calculus. It's not difficult to show that the belief states that result from such accessibility relation are in correspondence with the belief states that follow from the belief update equation (3).

A distinction between our model and [18] is that the latter is defined syntactically and is focused on trajectories as opposed to states. For these reasons we think it may be less convenient for modeling and computation. There are other differences as well such as our ability to sense arbitrary expressions and handle non-determinism, and the ability in [18] to accommodate variables, quantification and an epistemic operator in the language.

7 SUMMARY

We have developed a model for reasoning about actions, knowledge and control in the presence of incomplete information and sensing operations. In this model, actions can be non-deterministic, sensors can report the value of either symbols, terms or formulas, and knowledge is constructively defined in terms of observations. The model is closely related to possible worlds and probabilistic accounts of knowledge and belief, and serves a bridge between them. Its simple syntax and semantics can make it more useful for the design of actual planners. We have actually used versions of this framework for the analysis of simple robot navigation programs [14] and the synthesis of POMDP controllers [6, 1].

[10] The sequence a_0 is the empty sequence.

ACKNOWLEDGEMENTS

We thank Blai Bonet for valuable feedback on a number of topics relevant to this paper. The first author was partially supported by Grant S1-96001365 from Conicit. The second author was partially supported by the CNPq/PROTEM III, Project LOGIA.

REFERENCES

[1] B. Bonet and H. Geffner, 'Planning and control with incomplete information using POMDPs: Experimental results'. Available at `http://www.ldc.usb.ve/~hector`.

[2] A. Cassandra, L. Kaebling, and M. Littman, 'Acting optimally in partially observable stochastic domains', in *Proceedings AAAI94*, pp. 1023–1028, (1994).

[3] G. Collins and L. Pryor, 'Planning under uncertainty: Some key issues', in *Proceedings IJCAI95*, (1995).

[4] D. Draper, S. Hanks, and D. Weld, 'Probabilistic planning with information gathering and contingent execution', in *Proceedings AIPS'94*, pp. 31–36. (1994).

[5] O. Etzioni, S. Hanks, D. Draper, N. Lesh, and M. Williamson, 'An approach to planning with incomplete information', in *Proceedings KR'92*, pp. 115–125. Morgan Kaufmann, (1992).

[6] H. Geffner and B. Bonet, 'High-level plannnig and control with incomplete information using POMDP's'. Available at `http://www.ldc.usb.ve/~hector`, 1998.

[7] M. Gelfond and V. Lifschitz, 'Representing action and change by logic programs', *J. of Logic Programming*, **17**, 301–322, (1993).

[8] S. Kripke, 'Semantical considerations on modal logic', in *Reference and Modality*, ed., L. Linsky, 63–72, Oxford University Press, (1971).

[9] H. Levesque, 'What is planning in the presence of sensing', in *Proceedings AAAI-96*, pp. 1139–1146, Portland, Oregon, (1996). MIT Press.

[10] J. Lobo, G. Mendez, and S. Taylor, 'Adding knowledge to the action description language $\mathcal{A}$', in *Proceedings AAAI-97*, pp. 454–459, (1997).

[11] R. Moore, 'A formal theory of knowledge and action', in *Formal Theories of the Commonsense World*, eds., J. Hobbs and R. Moore, Ablex Publishing Co., Norwood, N.J., (1985).

[12] L. Padulo and M. Arbib, *System Theory*, Hemisphere Publishing Co., 1974.

[13] M. Puterman, *Markov Decision Processes – Discrete Stochastic Dynamic Programming*, John Wiley and Sons, Inc., 1994.

[14] J. Ramírez and H. Geffner, 'Analysis of robot control programs using models of actions', in *Proceedings AAAI-97 Workshop on Robots*, (1997).

[15] R. Reiter, 'The frame problem in the situation calculus', in *AI and Math Theory of Computation*, ed., V. Lifschitz, 359–380, Academic Press, (1991).

[16] E. Sandewall. 'Features and Fluents', Oxford University Press, 1994.

[17] M. Shanahan, *Solving the Frame Problem*, MIT Press, 1997.

[18] R. Scherl and H. Levesque, 'The frame problem and knowledge producing actions', in *Proceedings of AAAI-93*, pp. 689–695. MIT Press, (1993).

[19] E. Sondik, *The Optimal Control of Partially Observable Markov Processes*, Ph.D. dissertation, Stanford University, 1971.