Planning with Observations and Knowledge

Robert P. Goldman and **Mark S. Boddy** and **Louise Pryor**
{goldman,boddy}@src.honeywell.com
louisep@aisb.ed.ac.uk

Introduction

We are concerned with increasing the expressiveness of planning methods to handle observations and conditional execution. In previous work, two of us have argued that one cannot correctly construct such a planner without an explicit distinction between what is true in the world and what the agent knows (believes) to be true (Goldman & Boddy 1994b)[1]. Also in previous work, one of us has created a planner, Cassandra, that appears to construct flexible plans with conditional execution and knowledge-gathering *without* such a representation (Pryor & Collins 1995). The work described here was done to address the tension between these two results.

To address these issues, we have developed a formalization of action, WCPL, based on propositional dynamic logic (PDL). We have chosen dynamic logic as a framework because it already has constructs we need, like conditionals and nondeterministic actions. PDL is also closer to the kinds of action representation (e.g., STRIPS, ADL) used by implemented planners than are competing languages like the situation calculus. We have used WCPL to give a semantics to Cassandra's plan and have shown that the Cassandra algorithm is sound with respect to this semantics.

Cassandra

Cassandra[2] is a contingency planner whose design is based on the notion of *deferred decisions*. A major problem with planning in the real world is that the information that is needed to decide on a course of action may not be available when the plan is constructed. For example, you may not know if a road is blocked by snow without going to look at it. Under these circumstances Cassandra will construct plans that include provision for making the decisions that cannot be made during plan construction. Such plans must provide for the acquisition of the information that will be needed to make the deferred decisions. The presence of knowledge goals in these plans is thus due to the presence of deferred decisions.

Cassandra

Cassandra is a partial order planner based on Penberthy and Weld's UCPOP which extends the SNLP (McAllester & Rosenblitt 1991) algorithm to handle context-dependent effects and a limited form of universal quantification (Penberthy & Weld 1992). Cassandra, like UCPOP, represents actions using modified STRIPS operators, each defined by the *preconditions* for executing an action and the *effects* that may become true as a result of executing it (Fikes & Nilsson 1971). For each possible effect there is in addition an associated set of *secondary preconditions*, which specify the conditions under which the action will have that effect (Pednault 1988).

The use of secondary preconditions is critical to Cassandra's ability to represent uncertain effects. It does so by assigning them *unknowable* preconditions, constructed using the pseudo-predicate **:unknown**. Unknowable preconditions carry two parameters, one designating a particular source of uncertainty (*e.g.*, an instance of a coin toss), the other a particular possible outcome of that uncertainty ("heads"). One must specify a set of mutually exclusive and exhaustive outcomes for each source of uncertainty. Cassandrauses this information to perform two types of reasoning that are needed in contingency planning: (1) that two actions or effects may not co-occur because they depend upon different outcomes of the same uncertainty, and (2) that a goal will necessarily be achieved by a plan, because it will be achieved for every possible outcome of every relevant uncertainty.

Cassandra does not construct a contingency plan until it encounters an uncertainty. Until this point, it proceeds in much the same manner as other planners in the SNLP family, using a means-ends analysis involving the alternation of two processes: *planning for open subgoals* and *avoiding bad interactions*. An uncertainty is introduced into a plan when a subgoal in

[1] without making substantial simplifying assumptions as in CNLP (Peot & Smith 1992) or PLINTH (Goldman & Boddy 1994a).

[2] Cassandra was a Trojan prophet who was fated not to be believed when she accurately predicted future disasters.

the plan is achieved by an effect with an unknowable precondition.

When an uncertainty has been introduced, the plan depends on a particular outcome of that uncertainty and the plan so far is effectively a contingency plan for that outcome. If the plan is to be sound, contingency plans must be constructed for all other possible outcomes of the uncertainty as well. Cassandra does this by splitting the plan into a set of *branches*, one for each possible outcome of the uncertainty. Cassandra labels the components in the existing plan to show that they depend on a particular outcome of the uncertainty, and introduces a new set of goals for each other possible outcome. The plan is not complete until all the goals are achieved in every possible outcome of every uncertainty.

Each plan component is relevant only to certain branches of the plan and is labeled accordingly. Cassandra propagates labels through the plan to indicate both the contingencies in which elements play a rôle, and the contingencies with which they are compatible. Note that these are two different issues: a plan element may be compatible with a contingency, in that it does not *interfere* with the achievement of any goals in that contingency, without actually helping to establish them. Positive labels, indicating the branches in which a component is required for goal achievement, propagate backwards from goals to the steps that achieve them.

Negative labels are introduced when an *unsafe* link is detected: a causal link that is threatened by another effect (the *clobberer*) in the plan. There are two standard methods for resolving an unsafe link in planners in the SNLP family: *separate* the clobberer and the unsafe link by adding codesignation constraints ensuring that they don't unify; or *reorder* the actions in the plan to ensure that the clobberer occurs outside the range of the link. The first of these methods is extended in Cassandra: another method of separating the clobberer from the unsafe link is to ensure that they only occur in different contingencies. This method of resolving unsafe links results in plan components being labeled with contingencies in which they may not occur. Negative labels are propagated forwards from the problematic effect to the subgoals it establishes.

Decisions in Cassandra

When an agent executes a branching plan, it must at some point decide which branch to take. Rather than treating this decision as an explicit part of the plan, previous work has in effect simply assumed that the agent will execute those steps that are consistent with the contingency that actually obtains (Warren 1976; Peot & Smith 1992). However, in order to know which contingency holds during execution, an arbitrarily large amount of work may be necessary in order to gather the information on which the decision is based (Pryor 1994). For the plan to be viable the planner must be able to ensure that it will be in a position to make the decision and to guarantee that the steps required to gather information do not conflict with those required to carry out the rest of the plan. The approach taken in the design of Cassandra to this problem is to treat decisions as explicit plan steps. By doing this Cassandra can make use of its existing mechanisms for achieving preconditions and avoiding conflicts.

Cassandra adds an explicit decision step each time it introduces a new branch into a plan. The decision step is added to the plan along with ordering constraints ensuring that it occurs after the step with which the uncertainty is associated and before any step with a subgoal the achievement of which depends upon a particular outcome of the uncertainty.

For a decision step to be operational, there must be an effective procedure by which the agent executing the plan can determine which decision to make. In Cassandra, each decision step has associated with it a set of rules, derived from an analysis of the uncertainty that led to the introduction of the new branch. The action of deciding which contingency to execute is modeled in Cassandra as the evaluation of a set of condition–action rules of the form:

If *condition1* then *contingency1*
If *condition2* then *contingency2*
...

The executing agent can decide which branch to pursue by evaluating these rules when it comes to the decision step. Since the intended effect of evaluating the decision-rules is to choose an appropriate contingency for each outcome of a particular uncertainty, the conditions are meant to be diagnostic of particular outcomes of the uncertainty. The agent cannot directly determine the outcome of an uncertainty, so it must infer it from the presence or absence of effects that depend upon that outcome.

Cassandra constructs its decision rules by checking for the effects of a given outcome that are used to establish preconditions in the contingency associated with that outcome. For example, in one of the two outcomes of the act of selecting a soft drink from a faulty soda machine, the machine dispenses a drink and takes the agent's money. Let us assume that this action was done as part of a plan that requires the agent to have a drink despensed to it. Cassandra would generate a plan containing an observation action to verify that the soda was, in fact, dispensed.

Cassandra does not need to analyze the plan operators to identify all the effects that could be expected to result from a given outcome of the uncertainty, and make the antecedent be the conjunction of these effects, as the important issue is to verify only that the contingency plan can actually succeed.[3] To return to

[3]This has the interesting consequence that the executing agent might, in principle, end up selecting a contingency

the soft-drink machine, Cassandra's plan would *not* contain an observation action to verify that the soda machine had taken the agent's money.

The antecedent condition of the decision-rule constructed by Cassandra is thus simply a conjunction of all the direct effects of a particular outcome that are used to establish preconditions in the contingency plan for that outcome. Decision-rules are constructed incrementally as the plan is elaborated and effects are used to establish preconditions (Pryor & Collins 1993).

In order to evaluate a decision-rule, the executing agent must be able to determine whether the rule's antecedent holds. The preconditions for the decision step must thus include goals to know the current status of each condition that appears as an antecedent of a rule in this condition. The preconditions of a decision step become open subgoals in the plan in the same way as do the preconditions of any other step. In other words, Cassandra generates a set of goals to acquire the knowledge necessary to evaluate the decision-rules.

WCPL

WCPL is an action representation created to support the development and analysis of contingency planning systems (Goldman & Boddy 1996). WCPL is based on propositional dynamic logic. WCPL has constructs for STRIPS-like actions, augmented with context-dependent effects and uncertain outcomes. A modality for knowledge permits plans with observation actions. Using the above machinery, plans for both versions of the "bomb in the toilet" (McDermott 1987) problem can be provided.

PDL

Propositional dynamic logic (PDL) is a modal propositional logic designed to support reasoning about simple programs (Harel 1984). In PDL, propositions are interpreted in the usual way. The modality corresponds to the execution of programs; we speak of propositions that possibly or necessarily hold after a program halts. PDL has constructs for sequences, repetition (which is not used in WCPL) and nondeterministic choice. These constructs can be used to compose programs (complex modalities).

Syntax of PDL Given a set of atomic propositions, Φ, and a set of atomic actions, $\aleph$, we define the wffs, $\mathcal{P}$, and programs, $\mathcal{A}$, over Φ and $\aleph$ as follows:

1. if $\alpha \in \aleph$, then $\alpha \in \mathcal{A}$
2. if $\phi \in \Phi$, then $\phi \in \mathcal{P}$
3. if $P, Q \in \mathcal{P}$, then $\neg P, P \vee Q \in \mathcal{P}$
4. if $P \in \mathcal{P}$, and $A \in \mathcal{A}$, then $\langle A \rangle P \in \mathcal{P}$
5. $\epsilon \in \aleph$
6. if $P \in \mathcal{P}$ and P contains no sub-formula of the form $\langle A \rangle P$, then $P? \in \mathcal{A}$.
7. If $A, B \in \mathcal{A}$, then $A; B \in \mathcal{A}$
8. If $A, B \in \mathcal{A}$, then $A \cup B \in \mathcal{A}$

A literal is an atomic proposition or its negation—if $\phi \in \Phi$, then ϕ and $\neg\phi$ are literals. ϵ is the null action. We provide the conventional extensions of notation, e.g. $\wedge$ for conjunction, as syntactic sugar. We also provide $[]$ as a dual for the $\langle\rangle$ modality (see discussion of semantics, below).

Semantics of PDL The semantics of PDL is defined in terms of Kripke structures, S, written (W, π, m). W is a non-empty set of world states. π is an interpretation of Φ, mapping each proposition to those states in W in which that proposition is satisfied. π is extended to expressions using the propositional connectives in the conventional way. m is an interpretation of the singleton programs (those consisting of a single action), mapping each action α to a set of pairs of world states denoting permitted state transitions for α. The interpretation of programs is defined recursively as follows, based on the relations in m:

Definition 1 (Semantics of programs)

$$m(\epsilon) = \{(s,s) \mid s \in W\} \quad (1)$$

$$m(P?) = \{(s,s) \mid s \in \pi(P)\} \quad (2)$$

$$m(A;B) = m(A) \circ m(B) \quad (3)$$

$$m(A \cup B) = m(A) \cup m(B) \quad (4)$$

'?' can be read as 'test.' $P?$ limits the possible subsequent world states to the set of worlds where P holds. ';' denotes sequential composition of programs. We will omit the ';' operator when doing so does not obscure the meaning. $\circ$ denotes composition of relations.

$\langle A \rangle P$ is satisfied in those worlds w for which there exists another world w' and a program A such that P is satisfied in w' and $(w, w') \in m(A)$. Intuitively, P is a possible outcome of A. The dual of $\langle\rangle$, corresponding to necessity, $[A]P$ may be read as P necessarily holds after A.

WCPL

WCPL provides a representation for STRIPS-style actions, augmented with context-dependent effects and uncertain outcomes. WCPL also has a modality for knowledge, that permits us to represent and reason about action with incomplete information and about information-gathering actions. The syntax and semantics of WCPL are drawn from PDL, of which it is a dialect.

A simple STRIPS operator has a set of preconditions, γ, and a set of postconditions, δ, where both γ and δ are conjunctions of literals. Preconditions represent

plan even though the outcome of the uncertainty were not the one with which that plan was associated. Notice that this would not cause a problem in the execution of the plan, since it would only occur if all the conditions for the plan's success were met.

limits on the situations in which an action can be applied. The effects of performing an action in a state in which its preconditions are not satisfied are undefined.

PDL cannot conveniently represent this state of affairs, because of the nature of the necessity modality. Since this is the dual of possibility, $[A]P$ will hold for any P if A cannot be executed successfully. Accordingly, WCPL has a distinguished atomic proposition **planfail** $\notin \Phi$. WCPL constrains the accessibility relation over possible worlds such that executing an action when one of its preconditions fails to hold causes **planfail** to become true (in the initial conditions **planfail** will always be false), and once true, **planfail** persists through all subsequent actions. Furthermore, we require that every action be regarded as executable in any situation — although now the effect may be to cause **planfail**. More formally:

$$[A]\,\textbf{planfail} \equiv \textbf{planfail} \vee \neg\gamma(A)$$

Actions in WCPL may have context-dependent effects and there may be uncertainty about their outcomes. WCPL represents context-dependent effects by associating with each action a set of mutually exclusive and exhaustive postcondition specifications (primitive outcomes), each associated with one of a set of *triggers*. WCPL permits actions whose outcomes are *in principle* not predictable by the planner, such as coin tosses. To handle such actions, each trigger is permitted to have not just one outcome, but a set of mutually-exclusive and exhaustive outcomes. We do not have space to present them here, but elsewhere we present a formalization of STRIPS-style actions, augmented by nondeterminism and context-dependent actions (Goldman & Boddy 1996).

In order to reason about the executing agent's state of knowledge, WCPL provides a second modality. The PDL semantics is augmented by a second accessibility relation, k, and an interpretation κ over the worlds (the interpretation is defined as π, *mutatis mutandis*):

The accessibility relationship is intended to hold between worlds that are consistent with the agent's beliefs. The agent's beliefs are founded on knowledge of the initial state and a correct understanding of the outcomes of actions.

Planning is the process, given a description of an initial state, and a goal, of finding a plan that will achieve the goal when executed in the initial state. In the PDL framework, the classical planning problem is posed as follows: given a goal G and an initial state description, D, find a program, A, such that:

$$D \rightarrow [A]\,G \wedge D, \neg\textbf{planfail} \rightarrow [A]\,\neg\textbf{planfail}$$

The first condition enforces that the plan achieve the goal, the second that actions only be attempted when their preconditions hold.

Cassandra plans in WCPL

The first step in our analysis of the Cassandra algorithm is to provide an independent semantics for Cassandra's plans, using WCPL. To do so, we must translate Cassandra's actions into WCPL equivalents. actions. We then translate Cassandra's plans into sets of WCPL programs. The first step is complicated by a number of factors: Cassandra and WCPL differ in their representations of context-dependent effects and uncertain outcomes; Cassandra's observation actions need explication; and there is no direct counterpart in WCPL for Cassandra's decision nodes. A single Cassandra plan corresponds to a set of WCPL programs both because Cassandra's plans are partially-ordered and because Cassandra uses labels as a compact representation for branching plans. In this section, we address these issues in turn, concluding with the translation of a Cassandra plan into WCPL.

We would like to emphasize that providing a semantics for Cassandra actions in WCPL is not a superfluous exercise. This is not a matter of providing an *alternate* semantics for Cassandra's actions: there is no previously-existing semantics.

Cassandra's actions

In this section we discuss a number of issues related to translating Cassandra's actions into the WCPL representation, including the representation of context-dependent effects and uncertain outcomes. We also discuss how Cassandra's method of simplifying reasoning about knowledge and information-gathering actions may be reflected in WCPL. We conclude with the translation of a sample Cassandra action.

Context-dependent effects are represented differently in Cassandra and WCPL. Cassandra's representation is an extension of that used in UCPOP(see Section). This representation is well-suited to use in domains where there is no uncertainty, since it supports the planner in identifying a minimal set of propositions that must be established in order to bring about a desired effect. It does not, however, support a planner in reasoning about the *correlations* between different effects.[4] WCPL's representation, using triggers, is designed to support this kind of reasoning. In a propositional setting, the expressive power of the two representations does not differ: from a set of effects and secondary preconditions, we may generate a set of mutually exclusive and exhaustive triggers.

Cassandra extends UCPOP's secondary precondition mechanism to represent actions with uncertain outcomes. An action whose outcomes have some *irreducible* uncertainty, will have effects whose secondary preconditions include a special **:unknown** proposition. An advantage of this mechanism is that it facilitates the extension of UCPOP into Cassandra.

These unknowable propositions are not propositions like others. Rather, they provide a method of encoding nondeterministic disjunction: the same state of affairs

[4] We conjecture that this representational choice contributes to the kind of incompleteness we discuss in the following section.

Definition 2 (Accessibility relation)

$$\begin{aligned} k(\alpha) &= \{(s,t) \mid s,t \in W \wedge [\exists((s',t') \in m(\alpha)) \ \ st\ s = s' \wedge t = t']\} \\ k(\epsilon) &= \{(s,s) \mid s \in W\} \\ k(P?) &= \{(s,s) \mid s \in \kappa(P)\} \\ k(A;B) &= k(A) \circ k(B) \\ k(A \cup B) &= k(A) \cup k(B) \end{aligned}$$

Definition 3 (Knowledge) *The knowledge modality is defined in terms of quantification over belief-accessible worlds.*

$$\begin{aligned} [A]\,\mathbf{Know}P &\triangleq \{s \in W \mid \forall t \in W, (s,t) \in k(A) \rightarrow t \in \kappa(P)\} \\ \langle A \rangle\,\mathbf{Know}P &\triangleq \{\exists s \in W \mid \forall t \in W, (s,t) \in k(A) \rightarrow t \in \kappa(P)\} \end{aligned}$$

as is encoded by WCPL's alternative outcomes. In our semantics for Cassandra, we will introduce disjunction directly.

Although it is acceptable semantically to replace Cassandra's representation of uncertainty with the disjunction provided by WCPL, doing so does not capture the pragmatic effect of the `:unknown` preconditions. These preconditions serve to partition the space of primitive outcomes into pragmatically useful equivalence classes, based on predictions of likely uses of an action. This prevents Cassandra's plans from unnecessary branching. This function of `:unknown` preconditions is the same as that of other secondary preconditions. Accordingly, for each `:unknown` precondition, we construct a partition of the primitive outcomes of the action.

Cassandra introduces a special kind of proposition, **Knowif**, to represent the effects of information-gathering actions. The semantics of **Knowif**P are **Know**$P \vee$ **Know**$\neg P$. Cassandra does not explicitly represent an agent's knowledge state. Instead, Cassandra can assume that agents will know the truth value of a proposition except when this knowledge is threatened by the introduction of an action with uncertain outcomes that can affect the proposition in question. Doing so requires that Cassandra be able to conjoin the outcomes of uncertain actions with appropriate **Knowif** propositions. We will see in the next section that Cassandra's decision nodes provide this mechanism.

Pryor and Collins give an example involving the operation of a faulty soft-drink machine: one that may or may not actually dispense a soft drink. Their representation of a simplified version of this action is given in Figure 1 (adapted from (Pryor & Collins 1995)). The corresponding WCPL formalism is given in Figure 2.

Cassandra's plans

A single Cassandra plan data structure corresponds to a set of WCPL plans, because Cassandra plans impose only a partial order over operators and decision nodes. Given the classical assumption of sequential execution by a single agent, any partially-ordered ("nonlinear") plan can be treated as denoting the set of totally-ordered action sequences consistent with that partial order. We have developed an algorithm that characterizes the set of totally-ordered WCPL programs that correspond to a given Cassandra plan.

The understanding of Cassandra's decision nodes is complicated by the fact that these nodes serve two purposes. First, they introduce branching into Cassandra's plans. Second, they provide the mechanism ensuring the appropriate introduction of observation actions.

Decision nodes indicate branch-points in a contingent plan; their ordering determines the structure of the resulting totally-ordered plan. Propagation of labels determines which actions are executed along a given branch. Our algorithm provides a unique totally-ordered plan for any given sequence of decisions by restricting the actions executed to those whose positive labels subsume the current context.

Decision nodes serve to "marshal" knowledge preconditions for Cassandra's plans. Recall that Cassandra behaves almost everywhere as a conventional STRIPS planner, *i.e.*, one with complete knowledge. The only time when Cassandra behaves differently is when it tries to use an uncertain outcome as an establisher. At this point the Cassandra algorithm calls for the introduction of a decision node, which has **Knowif** subgoals. The decision node acts as a point of conjunction between the uncertainty-introducing action and the knowledge-restoring action. Protections ensure that the action and the observation are not separated, so that the observation will function correctly.

Proof of soundness

In this section we prove the soundness of Cassandra's plans with respect to the semantics presented in the previous section. To do so, we adapt the proof of

```
Action:          (enter-selection ?machine)

Preconditions:   (:and (money-entered ?machine)
                       (plugged-in ?machine))

Effects:         (:when (:and (available ?machine ?selection)
                              (:unknown ?ok T))
                        :effect (dispensed ?selection)) uncertain effect
                 (:when (:and (available ?machine ?selection)
                              (:unknown ?ok F))
                  :effect (:not (dispensed ?selection))) uncertain effect
                 (:when (available ?machine ?selection)
                  :effect (:not (money-entered ?machine)))
                 (:when (:not (available ?machine ?selection))
                  :effect (another-selection-indicator-on ?machine))
```

Figure 1: Cassandra and a faulty soft-drink machine.

operator: enter-selection
preconditions(γ): money-entered, plugged-in
postconditions:

available	$\rightarrow$	dispensed, $\neg$ money-entered (1)
		$\neg$ dispensed, $\neg$ money-entered (2)
$\neg$ available	$\rightarrow$	selection-indicator-on

Figure 2: Cassandra and a faulty soft-drink machine, WCPL version.

soundness for UCPOP (Penberthy & Weld 1992), from which Cassandra was derived. The proof proceeds by defining a loop invariant such that when the planner halts, it will have a well-formed plan that will achieve the goal. In service of this proof we characterize two transformations that Cassandra can perform that are outside the realm of UCPOP.

For UCPOP, the necessary and sufficient conditions for achieving ϕ for step α are as follows:

1. ϕ holds in the initial conditions and for each intervening action, β, the preservation preconditions for that action for ϕ ($\prod_{\phi}^{\beta}$) hold when it is performed; *or*
2. At some time prior to α, some other action β is executed, and the causation preconditions for ϕ for that action hold just prior to the execution of β. For each intervening action ζ between β and α the preservation preconditions for that action for ϕ hold when it is performed.

These conditions are *sufficient* to establish ϕ in a Cassandra plan as well, but they are not necessary as they do not cover the establishment of ϕ using a nondeterministic effect of an action.

We need to establish a Cassandra Loop invariant as follows:

Definition 4 (Cassandra Loop Invariant) *If the subgoals in the goal agenda, Ag, are met by A, and* $\neg$**planfail** $\rightarrow \neg \langle A \rangle$ **planfail**, *then A will be a solution to the planning problem.*

We do not have the space to give the full proof of soundness here (see the full paper). We prove that the Cassandra loop invariant holds initially and then, by induction, throughout the planning process. The induction step relies on two propositions that show that the loop invariant is maintained when we insert a branch into the plan and when we insert additional steps into branches of the plan. The key to goal achievement is that when new steps are added to branches, additional knowledge acquisition goals are also added, to observe each consumed proposition whose value may be affected by a previously-occuring uncertain action.

Completeness

We show that Cassandra is not complete with respect to its WCPL semantics. We do so by presenting a simple plan in WCPL that Cassandra cannot find. This incompleteness should not be surprising: completeness with respect to the semantics would require that Cassandra solve the test problem in the course of generating its plans. We conjecture that Cassandra has a more limited form of completeness, but have yet to prove this.

One reason for Cassandra to be incomplete is that it cannot reason about interdependencies between its outcomes, except in a very limited way. For example, consider what would happen if Cassandra wanted to go water-skiing at a mountain resort. Let us further suppose that in order to successfully water ski, Cassandra must get to the resort and the lake must be unruffled.

Cassandra can only get to the resort through the mountain pass when it is not snowing, which depends on an uncertainty, whether or not the weather is windy. The lake is unruffled iff it is not windy. Finally, Cassandra can observe the condition of the mountain pass.

This scenario leaves Cassandra in the tantalizing situation of being able to get to the resort, but unable to enjoy her favorite sport.[5] Even after observing that the pass is clear, Cassandra cannot determine that the lake will be unruffled. However, if we axiomatize the above situation as suggested in Section , it is a theorem that the lake is known to be unruffled after the agent gets to the lake.

We believe that part of the reason for this incompleteness is the action representation chosen by Pryor and Collins. The ADL-based representation does not support reasoning about such correlations. However, we do not believe that a complete planner for WCPL will prove practical. Verifying the correctness of WCPL programs requires us to reason about a possibly exponential number of branches. Ergo, we believe that restricted systems like Cassandra will be a practical necessity.

We conjecture that Cassandra is complete in a more restricted sense. We believe that Cassandra is complete with respect to the two plan expansion operations discussed in the proof of soundness: the addition of new branches and the addition of new steps to branches. Recall that when new steps are added to branches, additional knowledge acquisition goals are also added, to observe each consumed proposition whose value may be affected by a previously-occuring uncertain action. These correspond to a kind of "show-me" concept of knowledge[6] that may turn out to have practical advantages in environments that are less than perfectly modeled. We have yet to construct a proof, but are actively working in that direction.

Discussion

In this paper, we have presented a formal description and analysis of Cassandra's planning algorithm and the resulting plans, with particular attention to the implications of some of the representational and algorithmic choices made in the design of Cassandra. The restriction on reasoning about knowledge to the application of **Knowif** tests on uncertain outcomes simplify the planning model and the inference that must be performed to determine whether a plan is well-formed, but complicates the semantics and limits the plans that Cassandra will find. In particular, propositions whose truth value is entailed rather than directly observable will never be known.

It may not be immediately obvious, and is thus worth pointing out, that Cassandra's operators encode in **:unknown** propositions not only sources of uncertainty, but heuristic choices about what partitions of the possible outcomes of an action are most suited for planning within a given domain. These secondary preconditions serve both to encode the physics of the domain, and heuristic knowledge about planning within that domain.

Finally, the structural freedom allowed in the decision rules applicable at a given decision node complicates both the execution model and any attempt to analyze the space of plans generated by Cassandra. The problem is that the decision rules are not guaranteed to be mutually exclusive, so that more than one may be applicable. Modification of this structure to ensure mutual exclusion should be straightforward, but would substantially alter the semantics of the plans that are generated.

We are pleased with the experience of applying WCPL to formalizing Cassandra. We have provided a clear execution model for Cassandra's plans, that can be taken as a point of departure for future developments. The two propositions in our proof of soundness also provide a precise characterization of the extension Cassandra makes to UCPOP. Having a language that provided facilities for reasoning about program execution and for the knowledge of the executing agent was a material assistance to us in this task.

We believe it to be both possible and fruitful to show that Cassandra is complete, in the restricted sense that Cassandra will find any plan in the set of those plans recognized as well-formed by Cassandra. This is a significant restriction from the set of all plans that are well-formed in the WCPL translation of Cassandra's planning language, given the traditional semantics for knowledge.

[5] No doubt learned from Laocoon.

[6] Perhaps Cassandra is from Troy, Mo.?

References

Fikes, R. E., and Nilsson, N. J. 1971. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 2:189–208.

Goldman, R. P., and Boddy, M. S. 1994a. Conditional linear planning. In Hammond, K. J., ed., *Artificial Intelligence Planning Systems: Proceedings of the Second International Conference*. Los Altos, CA: Morgan Kaufmann Publishers, Inc.

Goldman, R. P., and Boddy, M. S. 1994b. Representing uncertainty in simple planners. In Doyle, J.; Sandewall, E.; and Torasso, P., eds., *Principles of Knowledge Representation and Reasoning: Proceedings of the Fourth International Conference (KR94)*. San Mateo, CA: Morgan Kaufmann Publishers, Inc.

Goldman, R. P., and Boddy, M. S. 1996. Expressive planning and explicit knowledge. In *AIPS-96*.

Harel, D. 1984. Dynamic logic. In Gabbay, D., and Guenther, F., eds., *Handbook of Philosophical Logic*, volume II. D. Reidel Publishing Company. chapter II.10, 497–604.

McAllester, D., and Rosenblitt, D. 1991. Systematic nonlinear planning. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, 634–639. Cambridge, MA: MIT Press.

McDermott, D. V. 1987. A critique of pure reason. *Computational Intelligence* 3:151–160.

Pednault, E. 1988. Synthesizing plans that contain actions with context-dependent effects. *Computational Intelligence* 4(4):356–372.

Penberthy, J. S., and Weld, D. S. 1992. UCPOP: a sound, complete, partial order planner for ADL. In Nebel, B.; Rich, C.; and Swartout, W., eds., *Principles of Knowledge Representation and Reasoning:Proceedings of the Third International Conference*, 103–114. Los Altos, CA: Morgan Kaufmann Publishers, Inc.

Peot, M. A., and Smith, D. E. 1992. Conditional nonlinear planning. In Hendler, J., ed., *Artificial Intelligence Planning Systems: Proceedings of the First International Conference*, 189–197. Los Altos, CA: Morgan Kaufmann Publishers, Inc.

Pryor, L., and Collins, G. 1993. Cassandra: Planning for contingencies. Technical Report 41, The Institute for the Learning Sciences, Northwestern University.

Pryor, L., and Collins, G. 1995. Planning for contingencies: A decision-based approach. Unpublished manuscript.

Pryor, L. 1994. Opportunities and planning in an unpredictable world. Technical Report 53, The Institute for the Learning Sciences, Northwestern University.

Warren, D. H. 1976. Generating conditional plans and programs. In *Proceedings of the AISB Summer Conference*, 344–354.