

Planning with Sensing Actions and Incomplete Information using Logic Programming

Tran Cao Son[‡] Chitta Baral[†]

[‡] Department of Computer Science
New Mexico State University
PO Box 30001, MSC CS
Las Cruces, NM 88003, USA
tson@cs.nmsu.edu

[†] Department of Computer Science and Engineering
Arizona State University
Tempe, AZ 85287, USA
chitta@asu.edu

Abstract. We present a logic programming based conditional planner that is capable of generating both conditional and sequential conformant plans in the presence of sensing actions and incomplete information. We prove the correctness of our implementation and show that our planner is complete with respect to the 0-approximation of sensing actions and the class of conditional plans considered in this paper which is large enough to cover conditional plans with bounded length and branching factor. Finally, we present some preliminary experimental results and discuss further enhancements to the program.

1 Introduction

Classical planning assumes that agents have complete information about the world. For this reason, it is often labeled as unrealistic because agents operating in real-world environment often do not have complete information about their environment. Two important questions arise when one wants to remove this assumption: *how to reason about the knowledge of agents* and *what is a plan* in the presence of incomplete information. The first question led to the development of several approaches to reasoning about effects of sensing (or knowledge producing) actions [10, 15, 16, 22, 24, 27]. The second question led to the notions of *conditional plans* and *conformant plans*. The former contains sensing actions and conditionals such as the well-known “if-then-else” construct and the latter is a sequence of actions which leads to the goal regardless of the value of the unknown fluents in the initial state. In this paper, we refer to conditional planning and conformant planning as approaches to planning that generate conditional plans and conformant plans, respectively.

Approaches to conditional planning can be characterized by the techniques employed in its search process or by the action formalism that supports its reasoning process. Most of the early conditional planners implement a partial-order planning algorithm [8, 9, 20, 19, 28] and use Situation Calculus or STRIPS as the main vehicle in representing and reasoning about actions and their effects. Among the recent ones, CoPlas

[14] is a regression planner implemented in Sicstus Prolog that uses a high-level action description language to represent and reason about effects of actions, including sensing actions; and FLUX [26], implemented in constraint logic programming, is capable of generating and verifying conditional plans. A conditional planner based on a QBF theorem prover, that does not allow sensing actions, was also recently developed [21].

Conformant planning [3, 6, 2, 23] is another approach to deal with incomplete information. In conformant planning, a solution is a sequence of actions that achieves the goal from every possible initial state. Recent experimental result shows [3] that conformant planning based on model checking is computationally competitive compared to other approaches to conformant planning such as those using heuristics to generate plans [2] or those extending Graphplan [23]. While many conditional planners can deal with sensing actions, none of the conformant planners does. But most the important difference between the two is that, for some initial states and domains there may not exist any conformant plans, even though conditional plans may exist.

In this paper, we develop a logic programming based conditional planner with sensing actions in the presence of incomplete information. The idea of using logic programming for planning was first introduced in [25]. It has become more feasible since the development of fast and efficient answer set solvers such as **smodels** [17] and **dlv** [4]. Answer set planners solve a planning problem by translating it into a logic program whose stable models have a one-to-one correspondence with the plans of the original problem. This divides the planner development into two parts: the development of answer set solvers and the methodology for planner encoding; separating the implementation from the representation details. While there are several logic programming encodings for planning with complete information, there is only one proposal for planning under incomplete information [6] using logic programming. In this paper, we propose a new logic programming encoding for planning under incomplete information. The novelty of our approach lies in that our encoding can deal with sensing actions and can generate both conformant plans and conditional plans. We will prove the soundness of our implementation by proving that each conditional plan generated by the planner achieves the goal. We also show that the planner is complete with respect to the 0-approximation of sensing actions and the class of conditional plans considered in this paper, which is large enough to cover conditional plans with bounded length and branching factor.

We use the language \mathcal{A}_K and its 0-approximation as the language for representing and reasoning with sensing actions [24]. We adopt the 0-approximation of \mathcal{A}_K as the complexity of the planning problem with respect to this approximation (under limited sensing) is **NP**-complete [1] which is lower than the complexity of the problem with respect to the \mathcal{A}_K semantics. This allows us to use different systems capable of computing stable models of logic programs in our implementation. Here, we use **smodels** [17] but we believe that with little modification that takes care of the syntactic differences between **smodels** and **dlv** [4], the code presented in this paper can be used with **dlv** as well¹.

We will review the basics of action language with sensing actions in the next section. Afterward, we present the main result of this paper, a logic programming encoding of

¹ **dlv** is used in [6] to generate conformant plans.

a conditional planner with sensing actions. We then discuss the properties of our logic program. Finally, we discuss some desirable extensions of the current work.

2 Preliminaries

2.1 An Action Language with Sensing Actions

In this section, we review the language \mathcal{A}_K with its 0-approximation [24]. \mathcal{A}_K extends the high-level action description language \mathcal{A} of Gelfond and Lifschitz [7] by introducing two new types of propositions, the *knowledge producing proposition* and the *executability condition*.

Syntax of \mathcal{A}_K In \mathcal{A}_K , an action theory consists of two finite and disjoint sets of names called *actions* and *fluents* and a set of propositions of the following form:

$$\mathbf{determines}(a, g) \tag{1}$$

$$\mathbf{causes}(a, f, \{p_1, \dots, p_n\}) \tag{2}$$

$$\mathbf{executable}(a, \{p_1, \dots, p_n\}) \tag{3}$$

$$\mathbf{initially} f \tag{4}$$

where g is a fluent, f and p_i 's are fluent literals (a *fluent literal* is either a fluent g or its negation $\neg g$, written as $neg(g)$) and a is an action. A *k-proposition* of the form (1) states that the value of fluent g will be known after a is executed. (2), referred as *ef-proposition*, represents the (conditional) effect of action a while (3) (or *ex-proposition*) states a executability condition of a . Propositions of the form (4), also called *v-propositions*, are used to describe the initial situation. An action theory is given by a pair (D, I) where D consists of propositions of the form (1)-(3) and I consists of propositions of the form (4). D and I will be called the *domain description* and *initial state*, respectively.

Two ef-propositions with preconditions p_1, \dots, p_n and q_1, \dots, q_m respectively are said to be *contradictory* if they describe the effect of the same action a on complementary f 's, and $\{p_1, \dots, p_n\} \cap \{\overline{q_1}, \dots, \overline{q_m}\} = \emptyset$ where $\overline{f} = \neg f$ and $\overline{\overline{f}} = f$ for a fluent f . In the following, we will assume that action theories in consideration are *consistent* in that they do not contain contradictory ef-propositions.

Actions occurring in ef-propositions and k-propositions are called non-sensing actions and sensing actions, respectively. In this paper, we assume that the set of sensing actions and the set of non-sensing actions are disjoint. For simplicity, we will also assume that each sensing action occurs in at most one k-proposition.

Conditional Plan In the presence of incomplete information and knowledge producing actions, we need to extend the notion of a plan from a sequence of actions so as to allow conditional statements such as if-then-else, while-do, or case-endcase (see e.g. [11, 15, 24]). In this paper, we limit ourselves to conditional plans with only the if-then-else construct. The reasons are twofold. First, it is easy to see that case-endcase statements can be replaced by if-then-else statements. Second, our intention is to use an answer set solver to generate conditional plans. This implies that we will impose an upper bound on

the length of the plan. Under this condition, conditional plans containing the while-do statements could be represented using the if-then-else statements as well. Formally, we consider conditional plans defined as follows.

Definition 1 (Conditional Plan).

1. A sequence of non-sensing actions $a_1; \dots; a_k$ ($k \geq 0$), is a conditional plan.
2. If $a_1; \dots; a_{k-1}$ is a non-sensing action sequence, a_k is a sensing action that determines f , and c_1 and c_2 are conditional plans, then $a_1; \dots; a_{k-1}; a_k; if(f, c_1, c_2)$ is a conditional plan.
3. Nothing else is a conditional plan.

To execute a plan $a_1; \dots; a_{k-1}; a_k; if(f, c_1, c_2)$, the agent first executes a_1, \dots, a_k . It then evaluates f with respect to its knowledge. If it knows that f (resp. $\neg f$) is true it executes c_1 (resp. c_2). If neither f nor $\neg f$ is true, then the plan fails and the execution of the conditional plan which contains this conditional plan also fails.

0-approximation of Semantics of \mathcal{A}_K In the 0-approximation of \mathcal{A}_K , an a-state (or approximation state) is a pair $\langle T, F \rangle$, where T and F are disjoint sets of fluents. Intuitively, T (resp. F) is the set of fluents which are true (resp. false) in $\langle T, F \rangle$. A fluent f is *true* (resp. *false*) in an a-state $\sigma = \langle T, F \rangle$ if $f \in T$ (resp. $f \in F$); and f is *known* (resp. *unknown*) in σ if $f \in T \cup F$ (resp. $f \notin T \cup F$). A positive (resp. negative) fluent literal f is said to hold in $\langle T, F \rangle$ if $f \in T$ (resp. $\bar{f} \in F$). An action a is executable in an a-state σ if D contains an ex-proposition **executable** $(a, \{p_1, \dots, p_n\})$ and p_i 's hold in σ . Executing a non-sensing action a in an a-state σ will cause some fluents to become true, some become false, *some may become true*, and *some may become false*². These four sets of fluents are denoted by $e_a^+(\sigma)$, $e_a^-(\sigma)$, $F_a^+(\sigma)$, and $F_a^-(\sigma)$, respectively. The result function Res of D in the 0-approximation is defined by $Res(a, \langle T, F \rangle) = \langle T \cup e_a^+ \setminus F_a^-, F \cup e_a^- \setminus F_a^+ \rangle$. Executing a sensing-action a in an a-state will cause a fluent to be known. This leads to the following definition of a 0-transition function:

Definition 2 (Transition Function). Given a domain description D , the 0-transition function Φ of D is defined as follows:

- If a is not executable in σ , then $\Phi(a, \sigma) = \{\perp\}$ where \perp denotes an undefined a-state.
- If a is executable in σ and a is a non-sensing action then $\Phi(a, \sigma) = \{Res(a, \sigma)\}$.
- If a is executable in σ and a is a sensing action then $\Phi(a, \sigma) = \{\langle T \cup \{f\}, F \rangle, \langle T, F \cup \{f\} \rangle\}$ if $f \notin T \cup F$ and $\Phi(a, \sigma) = \{\sigma\}$ otherwise, where $\sigma = \langle T, F \rangle$ and **determines** (a, f) belongs to D ³.

We illustrate this definition by the medication example with sensing action from [23].

² Space limitation does not allow us to include a detailed review of the 0-approximation of \mathcal{A}_K .

³ Recall that we assume here that each sensing action occurs in only one k-proposition.

Example 1. In the medication example, a patient is infected (*inf*). He can take the medicine (*med*) and get cured if he were hydrated (*hyd*); otherwise, the patient will be dead. To become hydrated, the patient can drink (*dr*). The check action (*chk*) allows us to determine if the patient is hydrated or not. This domain can be represented in \mathcal{A}_K as follows.

$$\begin{aligned} & \text{initially}(\text{inf}). && \text{causes}(\text{dr}, \text{hyd}, \{\}). \\ & \text{initially}(\neg\text{dead}). && \text{causes}(\text{med}, \text{dead}, \{\neg\text{hyd}\}). \\ & \text{determines}(\text{chk}, \text{hyd}). && \text{causes}(\text{med}, \neg\text{inf}, \{\text{inf}, \text{hyd}\}). \end{aligned}$$

We assume that every action is executable in any a-state in which $\neg\text{dead}$ holds. The initial a-state is $\sigma_0 = \langle \{\text{inf}\}, \{\text{dead}\} \rangle$. We have that

$$\begin{aligned} e_{dr}^+(\sigma_0) &= \{\text{hyd}\}, & e_{dr}^-(\sigma_0) &= \emptyset, \\ e_{med}^+(\sigma_0) &= \emptyset, & e_{med}^-(\sigma_0) &= \emptyset, \\ F_{dr}^+(\sigma_0) &= \{\text{hyd}\}, & F_{dr}^-(\sigma_0) &= \emptyset, \\ F_{med}^+(\sigma_0) &= \{\text{dead}\}, & \text{and } F_{med}^-(\sigma_0) &= \{\text{inf}\}. \end{aligned}$$

This implies that $\text{Res}(\text{dr}, \sigma_0) = \langle \{\text{inf}, \text{hyd}\}, \{\text{dead}\} \rangle$, $\text{Res}(\text{med}, \sigma_0) = \langle \emptyset, \emptyset \rangle$, and $\text{Res}(\text{chk}, \sigma_0) = \langle \{\text{inf}\}, \{\text{dead}\} \rangle$. And, we have

$$\begin{aligned} \Phi(\text{dr}, \sigma_0) &= \{ \langle \{\text{inf}, \text{hyd}\}, \{\text{dead}\} \rangle \}, & \Phi(\text{med}, \sigma_0) &= \{ \langle \emptyset, \emptyset \rangle \} \\ \Phi(\text{chk}, \sigma_0) &= \{ \langle \{\text{inf}, \text{hyd}\}, \{\text{dead}\} \rangle, \langle \{\text{inf}\}, \{\text{dead}, \text{hyd}\} \rangle \}. \end{aligned}$$

It is easy to see that for each domain description D , the 0-transition function Φ is unique. The extended transition function $\hat{\Phi}$ which maps pairs of conditional plans and a-states into set of a-states is defined next.

- Definition 3.** 1. $\hat{\Phi}(a, \sigma) = \Phi(a, \sigma)$,
2. $\hat{\Phi}([], \sigma) = \{\sigma\}$,
3. For a sequence of actions a_1, \dots, a_k , $k > 0$,

$$\hat{\Phi}([a_1; \dots; a_k], \sigma) = \bigcup_{\sigma' \in \hat{\Phi}(a_1, \sigma)} \hat{\Phi}([a_2; \dots; a_k], \sigma'),$$

4. For a conditional plan $c = a_1; \dots, a_k; \text{if}(f, c_1, c_2)$,
 $\hat{\Phi}(c, \sigma) = \bigcup_{\sigma' \in \hat{\Phi}([a_1; \dots; a_k], \sigma)} \hat{\Phi}(\text{if}(f, c_1, c_2), \sigma')$ where

$$\hat{\Phi}(\text{if}(f, c_1, c_2), \sigma) = \begin{cases} \hat{\Phi}(c_1, \sigma) & \text{if } f \text{ holds in } \sigma \\ \hat{\Phi}(c_2, \sigma) & \text{if } \neg f \text{ holds in } \sigma \\ \{\perp\} & \text{otherwise} \end{cases}$$

5. For every conditional plan c , $\hat{\Phi}(c, \perp) = \{\perp\}$.

For an action theory (D, I) , an a-state σ is called an *initial a-state* of (D, I) if for any fluent literal f , f holds in σ_0 iff “**initially**(f)” $\in I$. It is easy to see that for each action theory, the initial a-state is unique. A 0-model is a pair (σ_0, Φ) where σ_0 is the initial a-state and Φ is a 0-transition function of (D, I) . The 0-entailment relation of (D, I) is defined next.

Definition 4. Let (D, I) be an action theory and (σ_0, Φ) be a 0-model of (D, I) . Let c be a conditional plan and f be a fluent literal, we say $D \models_I^0 f$ **after** c if $\perp \notin \hat{\Phi}(c, \sigma_0)$ and f holds in every a-state σ belonging to $\hat{\Phi}(c, \sigma_0)$.

Example 2. For the action theory (D, I) in Example 1, we have that $D \models_I^0 \neg\text{dead} \wedge \neg\text{inf}$ **after** c where $c = \text{chk}; \text{if}(\text{hyd}, \text{med}, [\text{dr}; \text{med}])$.

3 A Logic Programming Based Conditional Planner

In this section, we present a logic programming based approach to conditional planning. Given a planning problem $\mathcal{P} = (D, I, G)$, where (D, I) is an action theory and G is a conjunction of fluent literals⁴, we will translate \mathcal{P} into a logic program $\pi(\mathcal{P})$ whose stable models represent solutions to \mathcal{P} . Our intuition behind this task rests on an observation that each conditional plan c (Definition 1) corresponds to a labeled plan tree T_c defined as follows.

- For $c = a_1; \dots; a_k$, where a_i 's are non-sensing actions, T_c is a tree with k nodes labeled a_1, \dots, a_k , respectively, and a_{i+1} is the child of a_i for $i = 1, \dots, k - 1$. If $c = []$, T_c is the tree with only one node, whose label is *nil* (the empty action).
- For $c = a_1; \dots; a_k; if(f, c_1, c_2)$, where a_k is a sensing action that determines f and other a_i 's are non-sensing actions, T_c is a binary tree whose root has the label a_1 , a_i has a child with the label a_{i+1} for $i = 1, \dots, k - 1$, a_k has two children which are the roots of T_{c_1} and T_{c_2} and the labels of the links are f and $\neg f$, respectively.

We demonstrate this in Figure 1.

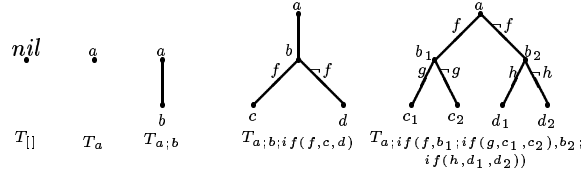


Fig. 1. Sample plan trees

For a conditional plan c , let m and w be the height and width of T_c . It is easy to see that we can assign to each node k of T_c a pair of integers (t, p) , denoted by $n(k)$, such that t is the level of k , $1 \leq p \leq w$, and

1. for every pair of leafs $k_1 \neq k_2$ of T_c , if $n(k_1) = (t_1, p_1)$ and $n(k_2) = (t_2, p_2)$ then $p_1 \neq p_2$, and
2. if k_1, \dots, k_s are the children of k with $n(k_j) = (t, p_j)$, then $n(k) = (t-1, \min\{p_1, \dots, p_s\})$.

It is easy to see that $n(k) = (1, 1)$ where k is the root of the tree⁵. One of the possible mappings for the trees in Figure 1 is given in Figure 2.

The above observation suggests us to add one more parameter to encode conditional plans. In our representation, besides the maximal length of the plan, we have another parameter denoting its maximal width. These two parameters will be represented by variables of the type *time* and *path*, respectively. Instead of the predicate $holds(F, T)$ (see e.g. [5, 12]) that says that the fluent literal F holds at the time T , we use $h(F, T, L)$ ($unknown(F, T, L)$) to represent the fact that F is true (unknown) at the node (T, L) .

⁴ In the sequel, we will show how more complex goals can be encoded.

⁵ We define the height of a tree to be the length (number of nodes) of the longest path in it.

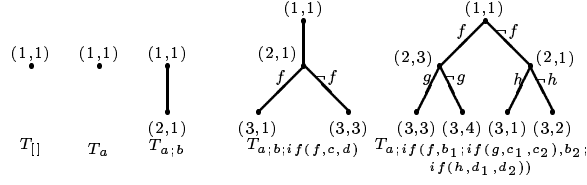


Fig. 2. A possible mapping for trees in Figure 1

We now describe the program $\pi(\mathcal{P})$. In our encoding, $\pi(\mathcal{P})$ consists of two set of rules, a set of domain dependent rules and a set of domain independent rules. More formally,

- The set of domain dependent rules consists of rules describing the initial states I , the goal G , and the set of propositions of D ;
- The set of domain independent rules consists of rules describing the transition functions Φ and auxiliary rules such as those generating action occurrences, defining fluent literals, contrary of fluent literals, etc.

We next describe these sets of rules. We begin with the set of domain dependent rules.

3.1 Domain Dependent Rules

• **Action theory representation.** We use two predicates *set* and *in* to define the sort *set* and to represent the set membership function, respectively. We assign to each set of fluent literals that occurs in a proposition of D a distinguished name. The constant *nil* denotes the set $\{\}$. A set of literals $\{p_1, \dots, p_n\}$ will be replaced by the set of atoms $Y = \{set(s), in(p_1, s), \dots, in(p_n, s)\}$ where s is the name assigned to $\{p_1, \dots, p_n\}$. With this representation, propositions in D and I can be easily translated into a set of facts of $\pi(\mathcal{P})$. For example, a proposition $causes(a, f, \{p_1, \dots, p_n\})$ with $n > 0$ is encoded as a set of atoms consisting of $causes(a, f, s)$ and the set Y (s is the name assigned to $\{p_1, \dots, p_n\}$). For example, the ef-proposition “ **causes** (*med*, *dead*, $\{\neg hyd\}$) is encoded by the set of rules

$$\begin{aligned} &causes(med, dead, set1). \\ &set(set1). \\ &in(neg(hyd), set1). \end{aligned}$$

• **Goal representation.** For each fluent literal F in G , the set of rules encoding G contains the following rule:

$$finally(F). \quad (5)$$

3.2 Domain Independent Rules

The domain independent rules of $\pi(\mathcal{P})$ are adapted mainly from [7]. The key difference is the representation of time that has been used previously in [5, 12, 13] and the used of paths. The main predicates in these rules are⁶:

⁶ Some predicates such as the conventional abnormal predicate ab are used with their standard meaning.

- $h(L, T, P)$: L holds at (T, P) ,
- $possible(A, T, P)$: action A is executable at (T, P) ,
- $occ(A, T, P)$ ($nocc(A, T, P)$): action A occurs (does not occur) at (T, P) ,
- $hs(S, T, P)$: S - a set of literals - holds at (T, P) ,
- $ph(L, T, P)$: L possibly holds at (T, P) ,
- $phs(S, T, P)$: S - a set of literals - possibly holds at (T, P) ,
- $used(T, P)$: starting from T , there is a node lying on the path P ,
- $br(F, T, P, P_1)$: node (T, P) has two children $(T + 1, P)$ and $(T + 1, P_1)$, where F (a fluent) holds at $(T + 1, P)$ and $\neg F$ holds at $(T + 1, P_1)$.

The main rules are given next. In these rules, T is a variable of the sort *time*, P, P_1 are variables of the sort *path*, F, G are variables denoting *fluent literals* (written as F or $\neg F$ for some fluent F), S is a variable set of the sort *set*, and A, B are variables of the sort *action*. The rules can be divided into different groups as follows.

- *Rules encoding the initial situation:*

$$h(F, 1, 1) \leftarrow literal(F), initially(F). \quad (6)$$

$$used(1, 1) \leftarrow \quad (7)$$

The first rule says that if F holds in the initial state ($initially(F) \in I$) then it should hold at the node $(1, 1)$ ($h(F, 1, 1)$). The second rule states that we always start with the construction of our conditional plan (or plan tree) from the node $(1, 1)$.⁷

- *Rules for reasoning about the effects of non-sensing actions:*

$$h(F, T+1, P) \leftarrow occ(A, T, P), causes(A, F, S), \quad (8)$$

$$hs(S, T, P)$$

$$ab(G, T, P) \leftarrow causes(A, \overline{G}, S), \quad (9)$$

$$occ(A, T, P), phs(S, T, P).$$

$$h(F, T+1, P) \leftarrow h(F, T, P), not ab(\overline{G}, T, P). \quad (10)$$

A non-sensing action, say a , changes the world according to the *Res* function. The first rule encodes the effects of a containing in $e_a^+(\sigma)$ and $e_a^-(\sigma)$ while the last two rules capture effects of removing what will possibly become false $F_a^-(\sigma)$ (or true $F_a^+(\sigma)$) from σ , the current state.

- *Rules for reasoning about the effects of sensing actions:*

$$new_br(P, P_1) \leftarrow P \neq P_1. \quad (11)$$

$$1\{br(F, T, P, P_1):new_br(P, P_1)\}1 \leftarrow \quad (12)$$

$$occ(A, T, P), determines(A, F).$$

$$\leftarrow fluent(F), P \neq P_1, br(F, T, P, P_1), used(T, P_1). \quad (13)$$

⁷ This works due to the fact that the root of the plan tree will always has the label $(1, 1)$, no matter how the assignment for the leaves is made, if the number of paths is equal the number of leaves of the tree or there is a leaf with a label $(t, 1)$. We note that our encoding makes sure that one of the leaves will have the label $(length, 1)$, thus allowing us to start building the tree from the node $(1, 1)$.

$$3\{h(F, T+1, P), h(\neg F, T+1, P_1), used(T+1, P_1)\}3 \quad (14)$$

$$\leftarrow P \neq P_1, fluent(F), br(F, T, P, P_1).$$

$$h(G, T+1, P_1) \leftarrow P \neq P_1, br(F, T, P, P_1), h(G, T, P). \quad (15)$$

$$h(G, T+1, P) \leftarrow P \neq P_1, br(F, T, P, P_1), h(G, T, P). \quad (16)$$

When a sensing action a that senses a fluent f occurs in σ and f is unknown in σ , $\Phi(a, \sigma)$ contains two states⁸. In one state, f is true and in the other, $\neg f$ is true. The two new states will lie on different branches of the plan-tree. Rules (11)-(14) encode just that. We note that the rules (12) and (14) have a different syntax than normal logic programming rule. They are introduced in [18] to easy the programming with **smodels**. (12) is satisfied in a stable model of a program if it contains one and only one member of the set $\{br(F, T, P, P_1):new_br(P, P_1)\}$. Rules (15) and (16) make sure whatever holds at the node (T, P) will continue to hold at both $(T+1, P)$ and $(T+1, P_1)$ since sensing actions do not change the world.

– *Rules for reasoning about what is known/unknown:*

$$unknown(F, T, P) \leftarrow fluent(F), not\ known(F, T, P). \quad (17)$$

$$known(F, T, P) \leftarrow fluent(F), h(F, T, P). \quad (18)$$

$$known(F, T, P) \leftarrow fluent(F), h(\neg F, T, P). \quad (19)$$

Rules of this group are rather standard. They say that if a fluent is true (or false) at (T, P) then it is known at (T, P) . Otherwise, it is unknown.

– *Rules for generating action occurrences:*

$$possible(A, T, P) \leftarrow used(T, P), hs(S, T, P), \quad (20)$$

$$executable(A, S).$$

$$occ(A, T, P) \leftarrow used(T, P), not\ sgoal(T, P), \quad (21)$$

$$possible(A, T, P), nocc(A, T, P).$$

$$nocc(B, T, P) \leftarrow used(T, P), not\ goal(T, P), \quad (22)$$

$$A \neq B, not\ occ(A, T, P).$$

$$\leftarrow A \neq B, occ(A, T, P), occ(B, T, P). \quad (23)$$

$$\leftarrow determines(A, F), occ(A, T, P), \quad (24)$$

$$known(F, T, P).$$

The first rule of this group defines when an action can be executed. (21)-(22) are used to generate action occurrences. They make sure that at least one action occurs if the goal has not been achieved. Rule (23) prevents actions to occur in parallel while the last rule does not allow a sensing action that senses f to occur when f is known.

– *Auxiliary Rules:* Rules in this group define the predicates hs , ph and phs and specify when the goal is achieved on a path such as $sgoal$ and $goal$.

$$used(T+1, P) \leftarrow used(T, P). \quad (25)$$

⁸ Under the assumption that a senses only one fluent f and a does not have effects on other fluents.

$$nhs(S, T, P) \leftarrow set(S), in(F, S), not h(F, T, P). \quad (26)$$

$$hs(S, T, P) \leftarrow set(S), not nhs(S, T, P). \quad (27)$$

$$nphs(S, T, P) \leftarrow set(S), in(L, S), literal(L) \quad (28)$$

$$contrary(L, G), h(G, T, P).$$

$$phs(S, T, P) \leftarrow set(S), not nphs(S, T, P). \quad (29)$$

$$ph(F, T, P) \leftarrow contrary(F, G), not h(G, T, P). \quad (30)$$

Rules in this group define when a set of fluents holds (27), when a fluent literal possibly holds (30), and when a set of fluent literals possibly holds (29). The next four rules define fluent literals and the predicate *contrary*.

$$literal(G) \leftarrow fluent(G). \quad (31)$$

$$literal(\neg G) \leftarrow fluent(G). \quad (32)$$

$$contrary(F, \neg F) \leftarrow fluent(F). \quad (33)$$

$$contrary(\neg F, F) \leftarrow fluent(F). \quad (34)$$

Finally, to require that the goal must be achieved at every path of the plan-tree, we add the following rules:

$$sgoal(T, P) \leftarrow not not_sgoal(T, P). \quad (35)$$

$$not_sgoal(T, P) \leftarrow finally(F), not holds(F, T, P). \quad (36)$$

$$\leftarrow used(length, P), not sgoal(length, P). \quad (37)$$

4 Properties of $\pi(\mathcal{P})$

In previous stable model based planners [5, 12], reconstructing a plan from a stable model of the program encoding the planning problem is simple: we only need to collect the action occurrences belonging to the model, order them by the time they occur, and we have a plan, i.e., if the stable model contains $occ(a_1, 1), \dots, occ(a_n, n)$ then the plan is a_1, \dots, a_n . For $\pi(\mathcal{P})$, the reconstruction process is not that simple because each stable model of $\pi(\mathcal{P})$ represents a conditional plan which mig contain conditionals. These conditionals, as we have discussed before, are represented by atoms of the form $br(F, T, P, P_1)$. Thus we have one more dimension (the path number) to deal with and we also need to consider the occurrences of the branching literal of the form $br(F, T, P, P_1)$. Let $\mathcal{P} = (D, I, G)$ be a planning problem and S be a stable model of $\pi(\mathcal{P})$, and i, k be integers. We define:

$$p_i^k(S) = a_{i,k}; \dots; a_{i+l-1,k}; a_{i+l,k}; if(f, p_{i+l+1}^k(S), p_{i+l+1}^{k_1}(S))$$

where $0 \leq l$, $a_{i,k}, \dots, a_{i+l-1,k}$ are non-sensing actions, $a_{i+l,k}$ is a sensing action that determines f and $br(f, i+l, k, k_1) \in S$, and $occ(a_{t,k}, t, k) \in S$ for $t \in \{i, \dots, i+l\}$; $p_i^k(S) = []$ if S does not contain some atoms of the form $occ(a, i_1, k)$ for $i_1 \geq i$. Intuitively, $p_i^k(S)$ is a conditional plan with the root at (i, k) .

We will subsequently prove that $p_1^1(S)$ is a solution to the planning problem \mathcal{P} . First, we prove that $\pi(\mathcal{P})$ correctly implements the 0-transition function Φ (Lemma 1) and no branching is made when non-sensing actions occur whereas branching is

required when sensing actions occur. Assume that S is a stable model of $\pi(\mathcal{P})$, we define $s_{i,k} = \langle T, F \rangle$ where $T = \{f \mid f \text{ is a fluent, } h(f, i, k) \in S\}$ and $F = \{f \mid f \text{ is a fluent, } h(\neg f, i, k) \in S\}$.

Lemma 1. *Let S be a stable model of $\pi(\mathcal{P})$ whose input parameters are length and level (the maximal length and width of the plan), i, k be integers, and $\text{occ}(a, i, k) \in S$. Then,*

1. *if a is a non-sensing action then a is executable in $s_{i,k}$, $\Phi(a, s_{i,k}) = \{s_{i+1,k}\}$, and $\text{br}(f, i, k, k_1) \notin S$ for every pair of a fluent f and an integer k_1 ; and*
2. *if a is a sensing action that senses f then a is executable in $s_{i,k}$ and*
 - *unknown(f, i, k) $\in S$, and*
 - *there exists some integer $k_1 < \text{level}$ such that $\text{used}(i, k_1) \notin S$, and $\text{br}(f, i, k, k_1) \in S$, and $\Phi(a, s_{i,k}) = \{s_{i+1,k}, s_{i+1,k_1}\}$.*

With the help of the above lemma we can prove the following theorem.

Theorem 1. *Let $\mathcal{P} = (D, I, G)$ be a planning problem and S be a stable model of $\pi(\mathcal{P})$. Then $p_1^1(S)$ is a conditional plan satisfying that $D \models_I^0 G$ after $p_1^1(S)$.*

Theorem 1 shows the soundness of $\pi(\mathcal{P})$. The next theorem shows that $\pi(\mathcal{P})$ is complete in the sense that it can generate all conditional plans which are solutions to \mathcal{P} .

Theorem 2. *Let $\mathcal{P} = (D, I, G)$ be a planning problem and p be a conditional plan restricted to the syntax defined in Definition 1. If p is a solution to \mathcal{P} then there exists a stable model S of $\pi(\mathcal{P})$ such that $p = p_1^1(S)$.*

5 Experiments and Discussions

We have tested⁹ our program with a number of domains from the literature, generating conformant plans [3, 2, 23] as well as generating conditional plans for domains given in ftp.cs.washington.edu/pub/ai/domains.pddl.tgz. The first collection is the set of domains without sensing actions but the initial states are incomplete. On the other hand, domains in the second collection contain sensing actions. We describe in short the results below.

Generating Conformant Plans. Conformant plans can be generated by $\pi(\mathcal{P})$ by setting the number of branches of the plan equals one, i.e., $\text{path} = 1$. In that case, each plan generated by $\pi(\mathcal{P})$ is a conformant plan, i.e., an action sequence that achieves the goal from every possible initial state. This is what planners with incomplete information in [3, 2, 23] do. We have done experiments with the toilet domains [2] and the results show that our program performs reasonably well. For example, the running time for the BT problem with 9, 10, and 11 packages, the running time of our program is 2.92, 3.141, and 9.6s, respectively. The running time for the BTC problem with 5, 6, 7, 8, and 9 packages is 0.30, 1.372, 0.630, 11.045, and 90.570s, respectively. We omit here the details due to the limited space.

⁹ All experiments were run on a OmniBook 6000 Laptop with 130544 Kb RAM, using Lparse version 0.99.52 (Windows 2000), April 2000 and SMOBELS version 2.25.

Generating Conditional Plans. For the medication problem (Example 1-2), we run our example in **smodels** with $path = 2$ and $length = 4$. We obtained a stable model with the following action occurrences: $occ(chk, 1, 1)$, $occ(med, 2, 1)$, $occ(dr, 2, 2)$, and $occ(med, 3, 2)$ and the branching literal $br(hyd, 1, 1, 2)$. This represents the plan $chk; if(hyd, med, [dr; med])$ in Example 2.

We tried the program with some benchmarks in planning with incomplete information and sensing actions such as the *travel* and the *montlake* domain. In both cases, $\pi(\mathcal{P})$ was able to generate conditional plans for the goal of the problem in less than a few minutes. In the second domain, it was able to solve a complex goal that the planner CoPlas [14] was not able to find a solution for.

Final Remarks. We present a sound and complete logic programming encoding of the planning problem with sensing actions in the presence of incomplete information. Our encoding shows that model-based approach to planning can be extended to planning with sensing actions and incomplete information. We are not aware of any other model-based planner that deals with sensing actions. In this paper, we concentrate on the representation issue of the problem. We are currently working on a detailed comparison with other planners that deal with sensing actions [8, 9, 20, 19, 28]. So far we have noticed that most of these planners do not have formal correctness proofs (unlike ours). In the future we plan to extend the logic programming encoding so that static causal laws, representing state constraints, can be added. We would also like to investigate methods such as use of domain knowledge to speed up the planning process.

References

1. C. Baral, V. Kreinovich, and R. Trejo. Planning and approximate planning in presence of incompleteness. In *Proceedings of the Sixteen International Joint Conference on Artificial Intelligence*, pages 948–953. Morgan Kaufmann Publishers, San Mateo, CA, 1999.
2. B. Bonet and H. Geffner. Planning with incomplete information as heuristic search in belief space. In *AIPS*, 1998.
3. A. Cimatti and M. Roveri. Conformant planning via model checking. In *European Conference on Planning*, pages 21–34. Springer Verlag, LNAI 1809, 1999.
4. S. Citrigno, T. Eiter, W. Faber, G. Gottlob, C. Koch, N. Leone, C. Mateis, G. Pfeifer, and F. Scarcello. The dlv system: Model generator and application frontends. In *Proceedings of the 12th Workshop on Logic Programming*, pages 128–137, 1997.
5. Y. Dimopoulos, B. Nebel, and J. Koehler. Encoding planning problems in non-monotonic logic programs. In *Proceedings of European conference on Planning*, pages 169–181, 1997.
6. T. Eiter, W. Faber, N. Leone, G. Pfeifer, and A. Polleres. Planning under incomplete information. In *Proceedings of the First International Conference on Computational Logic (CL'00)*, pages 807–821. Springer Verlag, LNAI 1861, 2000.
7. M. Gelfond and V. Lifschitz. Representing actions and change by logic programs. *Journal of Logic Programming*, 17(2,3,4):301–323, 1993.
8. K. Golden. Leap Before You Look: Information Gathering in the PUCINI planner. In *Proceedings of the 4th International Conference on Artificial Intelligence Planning and Scheduling Systems*, 1998.
9. K. Golden, O. Etzioni, and D. Weld. Planning with execution and incomplete informations. Technical report, Dept of Computer Science, University of Washington, TR96-01-09, February 1996.

10. K. Golden and D. Weld. Representing sensing actions: the middle ground revisited. In *KR 96*, pages 174–185, 1996.
11. H. Levesque. What is planning in the presence of sensing? In *Proceedings of the 14th Conference on Artificial Intelligence*, pages 1139–1146. AAAI Press, 1996.
12. V. Lifschitz. Answer set planning. In *International Conference on Logic Programming*, pages 23–37, 1999.
13. V. Lifschitz and H. Turner. Representing transition systems by logic programs. In *Proceedings of the 5th International Conference on Logic Programming and Nonmonotonic Reasoning*, pages 92–106, 1999.
14. J. Lobo. COPLAS: a COnditional PLAnner with Sensing actions. Technical Report FS-98-02, AAAI, 1998.
15. J. Lobo, S. Taylor, and G. Mendez. Adding knowledge to the action description language *A*. In *AAAI 97*, pages 454–459, 1997.
16. R. Moore. A formal theory of knowledge and action. In J. Hobbs and R. Moore, editors, *Formal theories of the commonsense world*. Ablex, Norwood, NJ, 1985.
17. I. Niemelä and P. Simons. Smodels - an implementation of the stable model and well-founded semantics for normal logic programs. In *Proceedings ICLP & LPNMR*, pages 420–429, 1997.
18. I. Niemelä, P. Simons, and T. Soinen. Stable model semantics for weight constraint rules. In *Proceedings of the 5th International Conference on on Logic Programming and Nonmonotonic Reasoning*, pages 315–332, 1999.
19. M.A. Peot and D.E. Smith. Conditional Nonlinear Planning. In *AIPS*, pages 189–197, 1992.
20. L. Pryor and G. Collins. Planning for contingencies: A decision-based approach. *Journal of Artificial Intelligence Research*, 4:287–339, 1996.
21. J. Rintanen. Constructing conditional plans by a theorem prover. *Journal of Artificial Intelligence Research*, 10:323–352, 2000.
22. R. Scherl and H. Levesque. The frame problem and knowledge producing actions. In *Proceedings of the 12th National Conference on Artificial Intelligence*, pages 689–695. AAAI Press, 1993.
23. D.E. Smith and D.S. Weld. Conformant graphplan. In *AAAI*, pages 889–896, 1998.
24. T.C. Son and C. Baral. Formalizing sensing actions - a transition function based approach. *Artificial Intelligence*, 125(1-2):19–91, January 2001.
25. V.S. Subrahmanian and C. Zaniolo. Relating stable models and ai planning domains. In *Proceedings of the International Conference on Logic Programming*, pages 233–247, 1995.
26. M. Thielscher. The Fluent Calculus: A Specification Language for Robots with Sensors in Nondeterministic, Concurrent, and Ramifying Environments. Technical Report CL-2000-01, Computational Logic Group, Department of Computer Science, Dresden University of Technology, October 2000.
27. M. Thielscher. Representing the knowledge of a robot. In *Proceedings of the Seventh International Conference on Principles of Knowledge and Representation and Reasoning (KR'2000)*, pages 109–120, 2000.
28. D. Weld, C. Anderson, and D. Smith. Extending graphplan to handle uncertainty and sensing actions. In *Proceedings of AAAI 98*, 1998.