Planning with Sensing Actions and Incomplete Information using Logic Programming

Tran Cao Son[‡] Phan Huy Tu[‡] Chitta Baral[†]

[‡] Department of Computer Science
New Mexico State University
PO Box 30001, MSC CS
Las Cruces, NM 88003, USA
{tson,tphan}@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 plans and 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. Finally, we present 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 [11, 15, 16, 22, 24, 26]. The second question led to the notions of *conditional plans* and *conformant plans* which lead to the goal regardless of the value of the unknown fluents in the initial situation. The former contains sensing actions and conditionals such as the well-known "if-then-else" construct and the latter is a sequence of actions. 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 [9, 10, 20, 19, 27] 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 [25], implemented in constraint logic programming, is capable of generating and verifying conditional plans. A conditional planner based on a QBF theorem prover was recently developed [21].

Conformant planning [2, 3, 6, 7, 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 situation. 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 based on heuristic search algorithms [2] or those that extends Graphplan [23]. A detailed comparison in [7] demonstrates that a logic programming based conformant planner is competitive with other approaches to planning.

The most important difference between conditional planners and conformant planners lies in the fact that conditional planners can deal with sensing actions and conformant planners cannot. Consequently, there are planning problems solvable by conditional planners but not by conformant planners. Following is one of such examples.

Example 1 (Defusing A Bomb). (From [24]) An agent needs to defuse a bomb. He was given a procedure for completing the job, which works when a special lock on the bomb is in the *off* position (i.e., $locked$). Applying the procedure when the lock is in the $unlocked$ position will cause the bomb to explode and kill the agent. The agent can determine if the lock is locked or not by $looking$ at the lock. He can also $turn$ the lock from the $locked$ position to the $unlocked$ position and vice-versa. He can only execute the above actions if the bomb has not exploded. Initially, the agent knows that the bomb is not defused and is not exploded. His goal is to safely defuse the bomb.

Intuitively, the agent can safely defuse the bomb by **(a)** looking at the lock to determine the position of the lock. He will then know exactly whether the lock is locked or not. If the lock is not locked, he **(b1)** turns the lock to the locked position and applies the procedure to defuse the the bomb; otherwise, he simply needs to **(b2)** apply the procedure to defuse the bomb.

Observe that no sequence of actions can achieve the goal from every possible initial situation, i.e., *there exists no conformant plan* that can help the agent to achieve his goal.

In this paper, we investigate the use of *answer set programming* in conditional planning. Given a planning problem with sensing actions and incomplete information about the initial situation, we translate it into a logic program whose answer sets – which can be computed using the well-known answer set solvers **smodels** [17] or **dlv** [5] – correspond to conditional plans satisfying the goal. We discuss the advantages and disadvantages of the approach and provide the results of our initial experiments. We will review the basics of an action language with sensing actions in the next section. Afterward, we present the main result of this paper, a logic programming encoding of 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

In this paper, we use a variation of the language $\mathcal{A}_K$ with its 0-approximation [24] as the representation language for our planner. 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 answer sets of logic programs in our implementation. $\mathcal{A}_K$ extends the high-level action description language $\mathcal{A}$ of Gelfond and Lifschitz [8] by introducing two new types of propositions, the *knowledge producing proposition* and the *executability condition*. In this paper, we will also allow *static causal laws* and sensing actions which determine the truth value of more than one fluent with the restriction that the action theories are deterministic (precise definition follows shortly). We now review the basics of the language $\mathcal{A}_K$ and discuss the notion of a conditional plan.

2.1 Action Language $\mathcal{A}_K$ – Syntax

Propositions in $\mathcal{A}_K$ are of the following form:

$$\textbf{causes}(a, f, \{p_1, \ldots, p_n\}) \tag{1}$$
$$\textbf{executable}(a, \{p_1, \ldots, p_n\}) \tag{2}$$
$$\textbf{if}(f, \{p_1, \ldots, p_n\}) \tag{3}$$
$$\textbf{determines}(a, \{l_1, \ldots, l_m\}) \tag{4}$$
$$\textbf{initially}(f) \tag{5}$$

where f and p_i's are fluent literals (a *fluent literal* is either a fluent g or its negation $\neg g$), each l_i is a fluent literal, and a is an action.

(1) represents the conditional effect of action a while (2) states an executability condition of a. A *static causal law* is of the form (3) and states that whenever $p_1, \ldots, p_n$ hold then f must hold. A proposition of the form (4) states that the value of fluent literals l_i's, called *sensed-fluent literals*, will be known after a is executed. We will require that the l_i's are mutually exclusive, i.e., in every state of the real world at most one of the l_i's holds. Propositions of the form (5), 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)-(4) and I consists of propositions of the form (5). D and I will be called the *domain description* and *initial situation*, respectively.

Actions occurring in (1) and (4) 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. We will also assume that each sensing action occurs in only one proposition of the form (4).

Example 2. The domain in Example 1 can be represented by the following action theory:

$$D_1 = \left\{ \begin{array}{ll} \textbf{causes}(disarm, exploded, \{\neg locked\}) & \textbf{if}(dead, \{exploded\}) \\ \textbf{causes}(disarm, disarmed, \{locked\}) & \textbf{executable}(disarm, \{\neg exploded, \neg dead\}) \\ \textbf{causes}(turn, \neg locked, \{locked\}) & \textbf{executable}(turn, \{\neg exploded, \neg dead\}) \\ \textbf{causes}(turn, locked, \{\neg locked\}) & \\ \textbf{determines}(look, \{locked, \neg locked\}) & \textbf{executable}(look, \{\neg exploded, \neg dead\}) \end{array} \right\}$$

$$I_1 = \left\{ \textbf{initially}(\neg dead) \;\; \textbf{initially}(\neg disarmed) \;\; \textbf{initially}(\neg exploded) \right\}$$

2.2 Conditional Plan

In the presence of incomplete information and sensing 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. [12, 15, 24]). Notice that an if-then-else statement can be replaced by a case-endcase statement. Moreover, if we are interested only in plans with bounded length, then whatever can be represented by a while-do statement with a non-empty body could be represented by a set of case-endcase statements. Therefore, in this paper, we limit ourselves to conditional plans with only the case-endcase construct. Formally, we consider conditional plans defined as follows.

Definition 1 (Conditional Plan).

1. *A sequence of non-sensing actions $a_1; \ldots; a_k$ ($k \geq 0$), is a conditional plan.*
2. *If $a_1; \ldots; a_{k-1}$ is a non-sensing action sequence, a_k is a sensing action that determines $f_1, \ldots, f_n$, and $c_1, \ldots, c_n$ are conditional plans, then $a_1; \ldots; a_{k-1}; a_k; case(\{f_i \rightarrow c_i\}_{i=1}^n)$ is a conditional plan.*
3. *Nothing else is a conditional plan.*

To execute a plan $a_1; \ldots; a_{k-1}; a_k; case(\{f_i \rightarrow c_i\}_{i=1}^n)$, the agent first executes $a_1, \ldots a_k$. It then evaluates f_i with respect to its knowledge. If it knows that one of the f_i is true it executes c_i. If none of the f_i's is true, then the plan fails and the execution of the conditional plan which contains this conditional plan also fails. It is easy to see that the following conditional plan achieves the goal of the agent in Example 1:

$$c_1 = look;\ case(\{locked \rightarrow disarm,\ \neg locked \rightarrow turn; disarm\}).$$

2.3 0-Approximation of Semantics of $\mathcal{A}_K$

The 0-approximation of an action theory (D, I) is defined by a transition function Φ that maps pairs of states and actions into sets of states. An *a-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 known to be true (resp. known to be false) in $\langle T, F\rangle$.

Given a fluent f and an a-state $\sigma = \langle T, F\rangle$, we say that a fluent f is *true* (resp. *false*) in σ 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 σ if $f \in T$ (resp. $\bar{f} \in F$). A set of fluent literals X holds in σ if each $l \in X$ holds in σ. An a-state is *complete* if for every fluent f of the theory, $f \in T \cup F$; it is incomplete otherwise. An a-state σ satisfies **if**$(f, \{f_1, \ldots, f_n\})$ if it is the case that if $\{f_1, \ldots, f_n\}$ holds in σ then f holds in σ. The truth value of fluent formula in σ is defined in the standard way, e.g., $f \wedge g$ (resp. $f \vee g$) holds in σ iff f holds in σ and (resp. or) g holds in σ. A *state* in (D, I) is an a-state that satisfies all static causal laws of the theory.

A state $\sigma = \langle T, F\rangle$ is said to extend an a-state $\sigma' = \langle T', F'\rangle$, denoted by $\sigma' \preceq \sigma$, iff $T' \subseteq T$ and $F' \subseteq F$; in addition, if $T' \neq T$ or $F' \neq F$, we say that σ properly extends σ' and write $\sigma' \prec \sigma$. If $\sigma' \preceq \sigma$, and there exists no state σ^+ such that $\sigma' \preceq \sigma^+ \prec \sigma$, we say that σ *minimally extends* σ'. For an a-state σ, by $Cl_D(\sigma)$ we denote the set of states of D that minimally extends σ. Notice that $Cl_D(\sigma)$ might be empty. It is also possible that $Cl_D(\sigma)$ contains more than one elements. In this case, we say that D is *non-deterministic*. Since our main goal in this paper is to generate plans in the presence of sensing actions and incomplete knowledge about the initial situation, we

will assume that action theories in consideration are deterministic, i.e., given an a-state σ, $|Cl_D(\sigma)| \leq 1$.

Given an action a and a state σ, $\Phi(a,\sigma)$ denotes the set of states that may be reached after the execution of a in σ. An action a is executable in a state σ if D contains a proposition **executable**$(a, \{p_1, \ldots, p_n\})$ and p_i's hold in σ. Executing a sensing-action a in a state will cause one of the sensed-fluent literals that occur in the proposition of the form (4) containing a to be true. Executing a non-sensing action a in a state σ will cause some fluents to become true, some become false, *some may become true*, and *some may become false*[1]. 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 is defined by $Res(a, \langle T, F\rangle) = \langle (T \cup e_a^+) \setminus F_a^-, (F \cup e_a^-) \setminus F_a^+\rangle$.

Definition 2 (Transition Function). *Given a domain description D, and action a and a state $\sigma = \langle T, F\rangle$, $\Phi(a,\sigma)$ is defined as follows:*

- *If a is not executable in σ then $\Phi(a,\sigma) = \{\bot\}$, where $\bot$ denotes an undefined state.*
- *If a is executable in σ and a is a non-sensing action then*

$$\Phi(a,\sigma) = \begin{cases} \{\bot\} & \text{if } Cl_D(Res(a,\sigma)) = \emptyset \\ Cl_D(Res(a,\sigma)) & \text{otherwise} \end{cases}$$

- *If a is executable in σ and a is a sensing action that occurs in a proposition of the form* **determines**$(a, \{l_1, \ldots, l_k\})$ *then*

$$\Phi(a,\sigma) = \begin{cases} \{\bot\} & \text{if } Cl_D(\langle T, F\rangle \cup \{l_i\}) = \emptyset \text{ for some } i \\ \bigcup_{i=1}^{k} Cl_D(\langle T, F\rangle \cup \{l_i\}) & \text{if } Cl_D(\langle T, F\rangle \cup \{l_i\}) \neq \emptyset \text{ for all } i's \\ & \text{and none of the } l_j's \text{ is known in } \sigma \\ \{\sigma\} & \text{otherwise} \end{cases}$$

where, for a fluent l, $\langle T, F\rangle \cup \{l\} = \langle T \cup \{l\}, F\rangle$ and $\langle T, F\rangle \cup \{\neg l\} = \langle T, F \cup \{l\}\rangle$.

We say that a domain description D is *consistent* if for every pair of a state σ and an action a, if a is executable in σ then $\bot \notin \Phi(a,\sigma)$. The extended transition function $\hat{\Phi}$ which maps pairs of conditional plans and states into sets of states is defined next.

Definition 3 (Extended Transition Function). *Given a state σ and a conditional plan c, $\hat{\Phi}(c,\sigma)$ is defined as follows:*

- *If $c = []$ then $\hat{\Phi}(c,\sigma) = \{\sigma\}$,*
- *If $c = a_1; \ldots; a_k$, where $k > 0$ and a_i's are non-sensing actions then $\hat{\Phi}(c,\sigma) = \bigcup_{\sigma' \in \Phi(a_1,\sigma)} \hat{\Phi}([a_2; \ldots; a_k], \sigma')$,*
- *If $c = a; case(\{f_i \rightarrow c_i\}_{i=1}^n)$ and a is a sensing action then $\hat{\Phi}(c,\sigma) = \bigcup_{\sigma' \in \Phi(a,\sigma)} \hat{E}(case(\{f_i \rightarrow c_i\}_{i=1}^n), \sigma')$ where*

$$\hat{E}(case(\{f_i \rightarrow c_i\}_{i=1}^n), \gamma) = \begin{cases} \hat{\Phi}(c_i, \gamma) & \text{if } f_i \text{ holds in } \gamma \\ \{\bot\} & \text{if none of the } f_i \text{ holds in } \gamma, \end{cases}$$

[1] Space limitation does not allow us to include a detailed review of the 0-approximation of $\mathcal{A}_K$ (see [24]).

– *If* $c = a_1; \ldots, a_k; case(\{f_i \rightarrow c_i\}_{i=1}^n)$ *and* $k > 1$ *then*
$\hat{\Phi}(c, \sigma) = \bigcup_{\sigma' \in \hat{\Phi}([a_1; \ldots, a_{k-1}], \sigma)} \hat{\Phi}([a_k; case(\{f_i \rightarrow c_i\}_{i=1}^n)], \sigma')$,
– *For every conditional plan* c, $\hat{\Phi}(c, \bot) = \{\bot\}$.

For an action theory (D, I), let $T_0 = \{f \mid f$ is a fluent, **initially**$(f) \in I\}$ and $F_0 = \{f \mid f$ is a fluent, **initially**$(\neg f) \in I\}$. A state σ is called an *initial state* of (D, I) if $\sigma \in Cl_D(\langle T_0, F_0 \rangle)$. A model is a pair (σ_0, Φ) where σ_0 is an initial state and Φ is the transition function of (D, I). The entailment relation of (D, I) is defined next.

Definition 4. *Let* (D, I) *be a consistent action theory,* c *be a conditional plan, and* ϕ *be a fluent formula. We say* $D \models_I \phi$ **after** c *if for every model* (σ_0, Φ) *of* (D, I) *it holds that* $\bot \notin \hat{\Phi}(c, \sigma_0)$ *and* ϕ *holds in every state* σ *belonging to* $\hat{\Phi}(c, \sigma_0)$.

Example 3. For the action theory (D_1, I_1) in Example 2, we have that

$$D_1 \models_{I_1} \neg dead \wedge \neg exploded \wedge disarmed \textbf{ after } c_1$$

where $c_1 = look;\ case(\{locked \rightarrow disarm,\ \neg locked \rightarrow turn; disarm\})$.

3 A Logic Programming Based Conditional Planner

We now present a logic programming based conditional planner. Given a planning problem $\mathcal{P} = (D, I, G)$, where (D, I) is an action theory and G is a conjunction of fluent literals, representing the goal, we will translate $\mathcal{P}$ into a logic program $\pi(\mathcal{P})$ whose answer sets 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 (Figure 1) defined as follows.

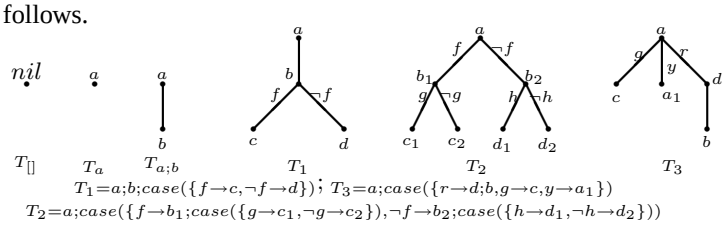

Fig. 1. Sample plan trees

- For $c = a_1; \ldots; a_k$, where a_i's are non-sensing actions, T_c is a tree with k nodes labeled $a_1, \ldots, a_k$, respectively, and a_{i+1} is the child of a_i for $i = 1, \ldots, k-1$. If $c = []$, T_c is the tree with only one node, whose label is nil (the empty action). The label for each link of T_c is empty string.
- For $c = a_1; \ldots; a_k; case(\{f_1 \rightarrow c_i\}_{i=1}^n)$, where a_k is a sensing action that determines f_i's and other a_i's are non-sensing actions, T_c is a tree whose root has the label a_1, a_i has a child with the label a_{i+1} for $i = 1, \ldots, k-1$, a_k has n children, the i^{th} child (from left to right) is the root of T_{c_i} and the label of the link is f_i. The label of the link between a_i and a_{i+1} ($1 \leq i \leq k-1$) is empty.

For a conditional plan c, let w be the number of leaves of T_c and h be the number of nodes along a longest path between the root and nodes of T_c. w and h will be called the *width* and *height* of T_c respectively. Let us denote the leaves of T_c by $l_1, \ldots, l_w$. We map each node k of T_c to a pair of integers n_k = (t_k,p_k), where $1 \leq t_k \leq h$ and $1 \leq p_k \leq w$, in the following way:

– For the i-th leaf l_i of T_c, $n_{l_i} = (t_{l_i}, i)$ where t_{l_i} is the number of nodes along the path from l_i to the root of T_c.
– For each interior node k of T_c with children $k_1, \ldots, k_r$ where $n_{k_j} = (t_{k_j}, p_{k_j})$, $n_k = (\min\{t_{k_1}, \ldots, t_{k_r}\}, \min\{p_{k_1}, \ldots, p_{k_r}\})$.

From the construction of T_c, it is easy to see that, independent of how we number the leaves of T_c, we have that

1. If $k_1, \ldots, k_r$ are the children of k with $n_{k_j} = (t_{k_j}, p_{k_j})$, then $t_{k_i} = t_{k_j}$ for every pair of i, j, $1 \leq i, j \leq r$.
2. The root of T_c is always mapped to the pair $(1, 1)$.

One of the possible mappings for the trees in Figure 1 is given in Figure 2.

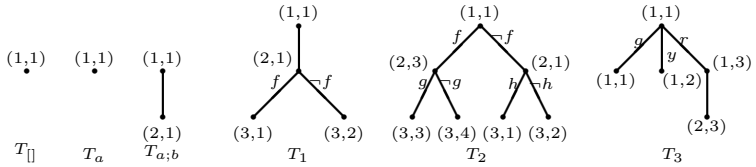

Fig. 2. A possible mapping for the trees in Figure 1

The above observation suggests us to add one more parameter to encode conditional plans. In our representation, besides the maximal length of the plan (the height of the tree), denoted by $length$, we have another parameter denoting its maximal width, denoted by $level$. A plan is encoded on the grid $length \times level$. Action occurrences will happen at the nodes of the grid and links connecting different paths representing case statements. The root of the tree will be at the node $(1, 1)$. We use variables of the type $time$ and $path$, respectively, to denote the position of a node relatively to the root. Instead of the predicate $holds(F, T)$ (see e.g. [4, 13]) that the fluent literal F holds at the time T, we use $h(F, T, P)$ ($unknown(F, T, P)$) to represent the fact that F is true (unknown) at the node (T, P) (the time moment T, on the path number P).

We now describe the program $\pi(\mathcal{P})$. Due to the fact that we use the **smodels** system in our experiment, $\pi(\mathcal{P})$ contains some rules that are specific to **smodels** such as the choice rule. In describing the rules of $\pi(\mathcal{P})$, the following convention is used: T denotes a variable of the sort $time$, $P, P_1, \ldots$ denote variables of the sort $path$, F, G are variables denoting *fluent literals*, $S, S_1, \ldots$ denote variables representing sensed-fluent literals, and A, B are variables representing $action$. We will also write $\bar{F}$ to denote the contrary of F. The main predicates of $\pi(\mathcal{P})$ are listed below:

– $h(L, T, P)$ is true if literal L holds at (T, P),
– $possible(A, T, P)$ is true if action A is executable at (T, P),
– $occ(A, T, P)$ is true if action A occurs at (T, P),
– $used(T, P)$ is true if starting from T, the path P is used in constructing the plan,
– $ph(L, T, P)$ is true if literal L possibly holds at (T, P), and
– $br(S, T, P, P_1)$ is true if node $(T + 1, P_1)$ is a child of (T, P) where S (a sensed-fluent literal) holds at $(T + 1, P_1)$.

The main rules, divided into different groups, are given next. Notice that they are just shown in an abstract form in which atoms indicating the sort of variables, such as $time, path, literal,$ and $sense$, that occur in the rule are omitted. For example, if a rule contains a variable T of sort $time$, it should be understood implicitly that the body of the rule also contains the atom $time(T)$. Similarly, if $\bar{F}$, where F is a literal, appears in the rule, then its body also contains the two atoms $literal(F)$ and $contrary(F, \bar{F})$

– *Rules encoding the initial situation.* If **initially**$(f) \in I$, $\pi(\mathcal{P})$ contains the fact

$$h(f, 1, 1) \leftarrow \tag{6}$$

This says that if **initially**$(f) \in I$ then f must hold at $(1, 1)$, the root of our tree[2].

– *Rule for reasoning about action's executability.* For each proposition of the form (2), $\pi(\mathcal{P})$ contains the rule

$$possible(a, T, P) \leftarrow h(p_1, T, P), \ldots, h(p_n, T, P). \tag{7}$$

– *Rule for reasoning with static causal laws.* For each proposition of the form (3), $\pi(\mathcal{P})$ contains the rule

$$h(f, T, P) \leftarrow h(p_1, T, P), \ldots, h(p_n, T, P). \tag{8}$$

– *Rules for reasoning about the effects of non-sensing actions.* For each proposition of the form (1), we add to $\pi(\mathcal{P})$ the following rules:

$$h(f, T{+}1, P) \leftarrow occ(a, T, P), possible(a, T, P), h(p_1, T, P), \ldots, h(p_n, T, P). \tag{9}$$
$$ab(\overline{f}, T, P) \leftarrow occ(a, T, P), possible(a, T, P), ph(p_1, T, P), \ldots, ph(p_n, T, P). \tag{10}$$
$$h(F, T{+}1, P) \leftarrow h(F, T, P), not\ h(\overline{F}, T{+}1, P), not\ ab(F, T, P). \tag{11}$$
$$ph(F, T, P) \leftarrow not\ h(\overline{F}, T, P). \tag{12}$$

Here, a is a non-sensing action. It 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 second rule indicates that the value of f could potentially be changed by the action a. This rule, when used in conjunction with the inertial rule (11), will have the effect 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.* For each proposition **determines**$(a, \{l_1, \ldots, l_n\})$ in D, $\pi(\mathcal{P})$ contains the set of facts $\{sense(l_i) \mid i = 1, \ldots, n\}$ and the following rules:

$$br(l_1, T, P, P) \leftarrow occ(a, T, P), possible(a, T, P). \tag{13}$$
$$1\{br(l_i, T, P, P_1) : new\_br(P, P_1)\}1 \leftarrow occ(a, T, P), possible(a, T, P). \tag{14}$$
$$(\text{for } i = 2, \ldots, n)$$
$$\leftarrow occ(a, T, P), known(l_i, T, P). \tag{15}$$

[2] Recall that the root of the tree always has the label $(1, 1)$.

The first two rules, in conjunction with the rules (20)–(22), state that when a sensing action a that senses the fluent literals l_i's occurs at the node (T, P), $n - 1$ new branches will be created and on each of the branch, one and only one of the sensed-fluent literal will become true. The last rule is a constraint that prevents a from occurring if one of the l_i's is already known. We note that the rule (14) has a different syntax than normal logic programming rule. They are introduced in [18] to ease the programming with **smodels**. The next rules describe the effect of the execution of a sensing actions.

$$new\_br(P, P_1) \leftarrow P < P_1. \tag{16}$$
$$used(T+1, P_1) \leftarrow br(S, T, P, P_1). \tag{17}$$
$$h(S, T{+}1, P_1) \leftarrow br(S, T, P, P_1). \tag{18}$$
$$h(G, T{+}1, P_1) \leftarrow br(S, T, P, P_1), h(G, T, P). \tag{19}$$
$$\leftarrow S \neq S_1, br(S, T, P_1, P_2), br(S_1, T, P_3, P_2). \tag{20}$$
$$\leftarrow P_3 \neq P_1, br(S_1, T, P_1, P2), br(S_2, T, P_3, P2). \tag{21}$$
$$\leftarrow T > 0, P_1 \neq P_2, br(S, T, P_1, P_2), used(T, P_2). \tag{22}$$

Rule (16) and the last three constraints make sure that none of the already in-use branches is selected when a new branch is created. The next two rules record that the path P is used starting from $(T{+}1, P)$ and that S will be true at $(T{+}1, P)$ when the literal $br(S, T, P_1, P)$ is true. Rule (19) plays the role of the inertial rule with respect to sensing actions. It makes sure whatever holds at (T, P) will continue to hold at $(T{+}1, P)$.

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

$$unknown(F, T, P) \leftarrow not\ known(F, T, P). \tag{23}$$
$$known(F, T, P) \leftarrow h(F, T, P). \tag{24}$$
$$known(F, T, P) \leftarrow h(\overline{F}, T, P). \tag{25}$$

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.*

$$1\{occ(A, T, P) : action(A)\}1 \leftarrow used(T, P), not\ sgoal(T, P), \tag{26}$$
$$\leftarrow occ(A, T, P), not\ possible(A, T, P). \tag{27}$$

The first rule is a choice rule that makes sure that only one action is executed at a node of the tree where the goal has not been achieved. The next rule requires that an action is executed only when it is executable.

– *Auxiliary Rules.*

$$literal(G) \leftarrow fluent(G). \tag{28}$$
$$literal(\neg G) \leftarrow fluent(G). \tag{29}$$
$$contrary(F, \neg F) \leftarrow fluent(F). \tag{30}$$

$$contrary(\neg F, F) \leftarrow fluent(F). \quad (31)$$
$$used(1,1) \leftarrow \quad (32)$$
$$used(T{+}1, P) \leftarrow used(T, P). \quad (33)$$
$$sub\_goal(T, P) \leftarrow not\ not\_sub\_goal(T, P). \quad (34)$$
$$not\_sub\_goal(T, P) \leftarrow finally(F), not\ h(F, T, P). \quad (35)$$
$$\leftarrow used(length, P), not\ sub\_goal(length, P). \quad (36)$$

The first two rules define what is a fluent literal. The next two rules specify that F and $\neg F$ are contrary literals. Rules (32) and (33) mark what nodes are used. Finally, rules (35) and (36) represent the fact that the goal must be achieved at every path of the plan-tree.

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

In previous answer set based planners [4, 6, 7, 13], reconstructing a plan from an answer set 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 answer set contains $occ(a_1, 1)$, ..., $occ(a_n, n)$ then the plan is $a_1, \ldots, a_n$. For $\pi(\mathcal{P})$, the reconstruction process is not that simple because each answer set of $\pi(\mathcal{P})$ represents a conditional plan which may contain case-statements. 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 branching literals of the form $br(F, T, P, P_1)$. Let $\mathcal{P} = (D, I, G)$ be a planning problem and S be an answer set of $\pi(\mathcal{P})$, and i, k be integers. We define:

$$p_i^k(S) = a_{i,k}; \ldots; a_{i+l-1,k}; a_{i+l,k}; \textit{case}(\{l_j \rightarrow p_{i+l+1}^{k_j}(S)\}_{j=1}^{t})$$

where $0 \leq l$, $a_{i,k}, \ldots, a_{i+l-1,k}$ are non-sensing actions, $a_{i+l,k}$ is a sensing action with the proposition **determines**$(a_{i+1,k}, \{l_1, \ldots, l_t\})$ in D, S contains the action occurrences $occ(a_{j,k}, t, k)$ for $j = i, \ldots, i + l$ and the branching literals $br(l_j, i + l, k, k_j)$ for $j = 1, \ldots, t$; $p_i^k(S) = []$ 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 the 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 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 an answer set of $\pi(\mathcal{P})$, we define $s_{i,k}{=}\langle T, F\rangle$ where $T{=}\{f \mid f$ is a fluent, $h(f, i, k) \in S\}$ and $F{=}\{f \mid f$ is a fluent, $h(\neg f, i, k) \in S\}$.

Lemma 1. *Let S be an answer set of $\pi(\mathcal{P})$ whose input parameters are $length$ and $level$, i, k be integers, and $occ(a, i, k) \in S$. Then,*

1. *if a is a non-sensing action then a is executable in $s_{i,k}$, $s_{i+1,k} \in \Phi(a, s_{i,k})$, and $br(f, i, k, k_1) \notin S$ for every pair of a fluent f and an integer k_1; and*
2. *if a is a sensing action with the proposition* **determines**$(a, \{l_1, \ldots, l_m\})$ *in D, then a is executable in $s_{i,k}$ and for every j, $1 \leq j \leq m$,*
   - *$unknown(l_j, i, k) \in S$, and*
   - *if $j > 1$, there exists some integer $k_j \leq level$ such that $used(i, k_j) \notin S$, and $br(l_j, i, k, k_j) \in S$, and $\Phi(a, s_{i,k}) = \{s_{i+1,k}\} \cup \{s_{i+1,k_2}, \ldots, s_{i+1,k_m}\}$.*

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 an answer set of $\pi(\mathcal{P})$. Then $p_1^1(S)$ is a conditional plan satisfying that $D \models_I G$* **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 an answer set S of $\pi(\mathcal{P})$ such that $p = p_1^1(S)$.*

Remark 1. **($\pi(\mathcal{P})$ as a conformant planner).** It is worth noticing that we can view $\pi(\mathcal{P})$ as a conformant planner. To see why, let us consider an answer set S of $\pi(\mathcal{P})$ and $p_1^1(S)$. Theorem 1 implies that $p_1^1(S)$ achieves the goal of $\mathcal{P}$ from every possible initial state of the domain. From the construction of $p_1^1(S)$, we know that if S does not contain a branching literal then $p_1^1(S)$ is a sequence of actions, and hence, a conformant plan. Furthermore, the definition of $\pi(\mathcal{P})$ implies that S contains a branching literal only if $level > 1$. Thus, if we set $level = 1$, $\pi(\mathcal{P})$ is indeed a conformant planner.
Remark 2. **(Size of $\pi(\mathcal{P})$).** It is obvious that we can characterize the size of an action theory by the number of fluents, actions, propositions, literals in the goal, and the number of sensed-fluent literals occurring in propositions of the form (4). Let n be the maximal number among these numbers. It is easy to see that the size of $\pi(\mathcal{P})$ is polynomial in n since the size of each group of rules is polynomial in n.
Remark 3. Because the 0-Approximation is incomplete w.r.t. to the full $\mathcal{A}_K$ semantics [24], there are situations in which $\mathcal{P}$ has solutions but $\pi(\mathcal{P})$ cannot find a solution. E.g., for $\mathcal{P}$=(D, I, G) with D=$\{$ **causes**$(a, f, \{g\})$, **causes**$(a, f, \{\neg g\})\}$, I=$\emptyset$, and G=f, p=a is a plan achieves f from every initial state; this solution cannot be found by $\pi(\mathcal{P})$.

5 Experiments and Discussions

We have tested our program with a number of domains from the literature. We concentrate on the generation of conditional plans in domains *with sensing actions*. In particular, we compare $\pi(\mathcal{P})$ with the system SGP, one of a few planners that are capable of generating both conditional plans and conformant plans in the presence of sensing actions and incomplete information available, from `http://www.cs.washington.edu/ai/sgp.html` in the *bomb in the toilet with sensing actions*, the *cassandra*, and the *illness* domains. To facilitate the testing, we develop a Sicstus program that translates a planning problem $\mathcal{P} = (D, I, G)$ – specified as a Sicstus program – into the program $\pi(\mathcal{P})$ and then use the **smodels** to compute one solution. All experiments were run on a Pentium III, Fujitsu FMV-BIBLO NB9/1000L laptop, 1GHz with 256 Mb RAM and 40 GB Harddisk. The operating environment is GNU/Linux 2.4 20-8. **lparse** version 1.0.13 and **smodels** version 2.27 are used in our computation. LispWorks 4.2.0 is used in running the SGP system. The source code of the translator and the problem in $\mathcal{A}_K$ ontologies are available at `http://www.cs.nmsu.edu/~tson/ASPlan/Sensing`. We detailed the results in Table 1.

For each problem, we record the computation time needed to find the first solution in both systems in three tries. The average time is also reported. Columns with the heading

'+' under the header $\pi(\mathcal{P})$ record the total time (**smodels** and **lparse**) needed to find one answer using $\pi(\mathcal{P})$. The other columns record the time reported by **smodels**. As it can be seen, $\pi(\mathcal{P})$ performs better than SGP in some domains and does not do as well in some other domains. We observe that in many domains, where SGP performs better, the initial state was specified using the PDDL 'oneOf' construct (i.e., the initial state is one of the possible given states). This construct provides SGP with extra information. In our encoding, we omit this information whenever the set of sensing actions is sufficient for the planner to acquire it; otherwise, we add an extra sensing action to allow our planner to determine the initial state. When we do so, $\pi(\mathcal{P})$ does better than SGP (e.g., the problem 'a6-prob'). We also observe that in some problems, where SGP performs better, the search space is rather huge with a lot of repetitions of an action. This can be seen in the problems of the 'Bomb' domain. In this problems, there are several alternatives of the sensing action that detects metal in a package. SGP consistently returns the same solution whereas $\pi(\mathcal{P})$ returns different solutions in different runs.

Domains/ Problem	SGP				$\pi(\mathcal{P})$							
	1^{st}	2^{nd}	3^{rd}	Avg.	1^{st}		2^{nd}		3^{rd}		Avg.	
					+	S	+	S	+	S	+	S
Cassandra												
a1-prob	140	170	140	150	387	190	358	210	390	190	378	197
a2-prob	40	40	40	40	811	500	810	510	809	460	810	490
a3-prob	50	50	40	47	282	110	284	110	282	130	283	117
a4-prob	300	300	300	300	704	470	705	480	705	490	705	480
a5-prob	50	30	40	40	185	80	219	70	185	80	196	77
a6-prob	NA [3]	NA	NA	NA	7,990	6,860	7,977	6,850	7,971	6,930	7,979	6,880
a7-prob	120	120	120	120	238	80	239	90	239	80	239	83
Bomb												
bt-1sa	1,790	1,630	1,660	1,693	4,491	3,660	4,420	3,510	4,702	3,640	4,538	3,603
bt-2sa	1,760	1,760	1,710	1,743	5,386	4,220	5,349	4,070	5,285	4,080	5,340	4,123
bt-3sa	1,940	1,910	1,900	1,917	5,404	3,950	5,349	4,140	5,369	4,070	5,374	4,053
bt-4sa	2,130	2,150	2,080	2,120	7,387	5,650	7,286	5,530	7,335	5,530	7,336	5,570
Sick												
sick-3-1	20	20	10	17	437	70	459	50	449	60	448	60
sick-3-2	130	130	160	140	810	320	812	320	791	300	804	313
sick-3-3	500	480	550	510	2,740	1,640	2,735	1,790	2,705	1,760	2,727	1,730
sick-3-4	2,630	2,610	2,650	2,630	3,632	2,390	3,652	2,400	3,633	2,460	3,639	2,417
sick-3-5	17,070	17,370	17,690	17,377	3,749	2,510	3,810	2,620	3,613	2,750	3,724	2,627

Table 1. Comparison with SGP (Time in milliseconds)

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. This distinguishes our planner from other model-based planners that do not deal with sensing actions [2, 3, 6, 7, 23] . We compare our planner with the system SGP and obtain encouraging results. In the future, we would also like to investigate methods such as use of domain knowledge to speed

[3] LispWorks stops with the error "Memory limit exceeded".

up the planning process. Furthermore, due to the fact that our planner can be viewed as a conformant planer, we would like to test our planner against other model-based conformant planners as well.

Acknowledgment. We would like to thank Michael Gelfond for his valuable comments on an earlier draft of this paper. We would also like to thank the anonymous reviewers of the paper for their constructive comments which help us to improve the paper in several ways. The work is partially supported by NSF grants EIA-0130887 and EIA-0220590.

References

1. C. Baral, V. Kreinovich, and R. Trejo. Planning and approximate planning in presence of incompleteness. In *IJCAI*, pages 948–953, 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 *ECP*, 21–34, 1999.
4. 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.
5. T. Eiter et al. The KR System dlv: Progress Report, Comparisons and Benchmarks. *KR*'98.
6. T. Eiter et al. Planning under incomplete information. In CL'2000.
7. T. Eiter et al. A Logic Programming Approach to Knowledge State Planning, II: The DLV$^{\mathcal{K}}$ System. Technical Report, TU Wien, 2003.
8. M. Gelfond and V. Lifschitz. Representing actions and change by logic programs. *Journal of Logic Programming*, 17(2,3,4):301–323, 1993.
9. K. Golden. Leap Before You Look: Information Gathering in the PUCCINI planner. In *Proc. of the 4th Int. Conf. on Artificial Intelligence Planning and Scheduling Systems*, 1998.
10. K. Golden, O. Etzioni, and D. Weld. Planning with execution and incomplete informations. Technical report, University of Washington, TR96-01-09, February 1996.
11. K. Golden and D. Weld. Representing sensing actions: the middle ground revisited. In *KR 96*, pages 174–185, 1996.
12. H. Levesque. What is planning in the presence of sensing? In *AAAI*, 1139–1146, 1996.
13. V. Lifschitz. Answer set planning. In *ICLP*, 23–37, 1999.
14. J. Lobo. COPLAS: a COnditional PLAnner with Sensing actions. FS-98-02, AAAI, 1998.
15. J. Lobo, S. Taylor, and G. Mendez. Adding knowledge to the action description language $\mathcal{A}$. 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 *Proc. ICLP & LPNMR*, pages 420–429, 1997.
18. I. Niemelä, P. Simons, and T. Soininen. Stable model semantics for weight constraint rules. In *LPNMR*, 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, *JAIR*, 1996.
21. J. Rintanen. Constructing conditional plans by a theorem prover. *JAIR*, 10:323–352, 2000.
22. R. Scherl and H. Levesque. The frame problem and knowledge producing actions. AAAI'96.
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. M. Thielscher. Programming of Reasoning and Planning Agents with FLUX. KR'02, 2002.
26. M. Thielscher. Representing the knowledge of a robot. In *KR'00*, 109–120, 2000.
27. D. Weld, C. Anderson, and D. Smith. Extending Graphplan to handle uncertainty and sensing actions. In *Proceedings of AAAI 98*, 1998.