TLP **7** (4): 377–450, 2007. © 2007 Cambridge University Press
doi:10.1017/S1471068406002948 Printed in the United Kingdom

Reasoning and planning with sensing actions, incomplete information, and static causal laws using answer set programming

PHAN HUY TU, TRAN CAO SON

Department of Computer Science, New Mexico State University, PO Box 30001, MSC CS, Las Cruces, NM 88003, USA
(*e-mail:* {tphan,tson}@cs.nmsu.edu)

CHITTA BARAL

Department of Computer Science and Engineering, Arizona State University, Tempe, AZ 85287, USA
(*e-mail:* chitta@asu.edu)

submitted 4 October 2005; revised 21 March 2006; accepted 14 April 2006

Abstract

We extend the 0-approximation of sensing actions and incomplete information in Son and Baral (2001) to action theories with static causal laws and prove its soundness with respect to the possible world semantics. We also show that the conditional planning problem with respect to this approximation is **NP**-complete. We then present an answer set programming based conditional planner, called ASCP, that is capable of generating both conformant plans and conditional plans in the presence of sensing actions, incomplete information about the initial state, and static causal laws. We prove the correctness of our implementation and argue that our planner is sound and complete with respect to the proposed approximation. Finally, we present experimental results comparing ASCP to other planners.

KEYWORDS: Reasoning about actions and changes, sensing actions, incomplete information, conformant planning, conditional planning, answer set programming.

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 (Golden and Weld 1996b; Lobo et al. 1997; Moore 1985; Scherl and Levesque 2003; Son and Baral 2001; Thielscher 2000b). The second question led to the notions of *conditional plan* and *conformant plan* whose execution is guaranteed to achieve the goal regardless of the values of unknown fluents in the initial situation. The former contains sensing

https://doi.org/10.1017/S1471068406002948 Published online by Cambridge University Press

actions and conditionals such as the well-known "if-then-else" or "cases" construct, while the latter is just a sequence of actions. In this paper, we refer to *conditional planning* and *conformant planning* as planning approaches that generate conditional plans and conformant plans, respectively. We use *plan* as a generic term for both conditional and conformant plan when the distinction between the two is not important.

Approaches to conditional planning can be characterized by the techniques employed in their search process or by the action formalism that supports their reasoning process. Most of the early conditional planners implemented a partial-order planning algorithm (Golden 1998; Golden et al. 1996a; Pryor and Collins 1996; Peot and Smith 1992) and used Situation Calculus or STRIPS as their underlying formalism in representing and reasoning about actions and their effects. Among them, CoPlaS (Lobo 1998), which is implemented in Sicstus Prolog, is a regression planner that uses a high-level action description language to represent and reason about effects of actions, including sensing actions; and FLUX (Thielscher 2000a), a constraint logic programming based planner, is capable of generating and verifying conditional plans. Another conditional planner based on a QBF theorem prover was developed in Rintanen (2000). Some other planners, for example, SGP (Weld et al. 1998) or POND (Bryce et al. 2004), extended the planning graph algorithm (Blum and Furst 95) to deal with sensing actions. The main difference between SGP and POND is that the former searches solutions within the planning graph, whereas the latter uses it as a means of computing the heuristic function.

Conformant planning (Bonet and Geffner 2000; Brafman and Hoffmann 2004; Cimatti et al. 2004; Castellini et al. 2003; Eiter et al. 2003; Smith and Weld 1998) is another approach to deal with incomplete information. In conformant setting, a solution is simply a sequence of actions that achieves the goal from every possible initial situation. A recent study (Cimatti et al. 2004) shows that conformant planning based on model checking is computationally competitive with other approaches to conformant planning such as those based on heuristic search algorithms (Bonet and Geffner 2000; Brafman and Hoffmann 2004) or those that extend Graphplan (Smith and Weld 1998). A detailed comparison in Eiter et al. (2003) demonstrates that a logic programming based conformant planner is able to compete 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 whereas conformant planners cannot. Consequently, there are planning problems solvable by conditional planners but not by conformant planners. The following example demonstrates this issue.

Example 1

Consider a security window with a lock that behaves as follows. The window can be in one of the three states *opened, closed*[1] or *locked*[2]. When the window is closed or

[1] The window is closed and unlocked.
[2] The window is closed and locked.

https://doi.org/10.1017/S1471068406002948 Published online by Cambridge University Press

opened, pushing it *up* or *down* will *open* or *close* it respectively. When the window is closed or locked, flipping the lock will lock or close it respectively.

Now, consider a security robot that needs to make sure that the window is locked after 9 pm. Suppose that the robot has been told that the window is not open (but whether it is locked or closed is unknown).

Intuitively, the robot can achieve its goal by performing the following steps. First, (1) it checks the window to determine the window's status. If the window is closed, (2.a) it locks the window; otherwise (i.e., the window is already locked), simply (2.b) it does nothing.

Observe that no sequence of actions can achieve the goal from every possible initial situation. In other words, *there exists no conformant plan* achieving the goal. □

In this paper, we investigate the application of *answer set programming* (Baral 2003; Lifschitz 2002; Marek and Truszczyński 1999; Niemelä 1999) in conformant and conditional planning. To achieve our goal, we first define an approximation semantic for action theories with static causal laws and sensing actions based on the 0-approximation in Son and Baral (2001). It is an alternative to the possible world semantics for reasoning about effects of actions in the presence of incomplete information and sensing actions (Moore 1985). The basic idea of this approach is to *approximate* the set of possible world states by a set of fluent literals that is true in every possible world state. The main advantage of the approximation-based approach is its low complexity in reasoning and planning tasks (**NP**-complete) comparing to those based on the possible world semantics Σ_2**P**-complete (Baral et al. 2000a). The trade-off for this low complexity is incompleteness. As we will demonstrate in our experiments, this is not really an issue with the benchmarks in the literature.

We prove that the entailment relationship for action theories based on this approximation is sound with respect to the possible world semantics for action theories with incomplete initial situation. We then show that the planning problem with respect to the newly developed approximation is **NP**-complete. This facilitates the development of ASCP, an answer set programming based planner that is capable of generating both conditional and conformant plans. Given a planning problem instance with incomplete information about the initial situation and sensing actions, we translate it into a logic program whose answer sets (Gelfond and Lifschitz 1988) – which can be computed using existing answer set solvers (e.g. `cmodels` (Lierler and Maratea 2004), `smodels` (Simons et al. 2002), **dlv** (Citrigno et al. 1997), ASSAT (Lin and Zhao 2002), NoMore (Anger, et al. 2002), etc.) – correspond to conformant or conditional plans that satisfy the goal. We compare our planner against state-of-the-art planners. The results of our experiments show that conditional and conformant planning based on answer set programming can be competitive with other approaches. To the best of our knowledge, no answer set based conditional planner has been developed except a previous version of the planner presented in an earlier version of this paper (Son et al. 2004).

The paper is organized as follows. Section 2 presents the basics of an action language with sensing actions and static causal laws, including its syntax and the

https://doi.org/10.1017/S1471068406002948 Published online by Cambridge University Press

0-approximation, as well as the notions of conditional plans and queries. It also contains the complexity result of the conditional planning problem with respect to the 0-approximation. Section 3 describes a logic programming encoding of a conditional/conformant planner, called ASCP. Section 4 discusses several properties of ASCP. Section 5 experimentally compares ASCP with some other state-of-the-art conformant/conditional planners. Section 6 discusses some desirable extensions of the current work. The proofs of theorems and propositions are given in Appendices A and B. An example of encoding is given in Appendix C.

2 $\mathcal{A}_K^c$ – an action language with sensing actions and static causal laws

The representation language, $\mathcal{A}_K^c$, for our planner is an extension of the action language $\mathcal{A}_K$ in (Son and Baral 2001). While $\mathcal{A}_K$ extends the high-level action description language $\mathcal{A}$ from Gelfond and Lifschitz (1993) by introducing two new types of propositions called *knowledge producing proposition* and the *executability condition*, $\mathcal{A}_K^c$ extends $\mathcal{A}_K$ by adding *static causal laws* and allowing a sensing action to sense more than one fluent. Loosely speaking, $\mathcal{A}_K^c$ is a subset of the language $\mathcal{L}_{DS}$ in (Baral et al. 2000b). Nevertheless, like $\mathcal{A}_K$, $\mathcal{L}_{DS}$ considers sensing actions that sense only one fluent. The semantics given for $\mathcal{A}_K^c$ in this paper is an approximation of the semantics of $\mathcal{L}_{DS}$.

2.1 Action language $\mathcal{A}_K^c$ – syntax

The alphabet of an action theory in $\mathcal{A}_K^c$ consists of a set of actions $\mathbf{A}$ and a set of fluents $\mathbf{F}$. A *fluent literal* (or *literal* for short) is either a fluent $f \in \mathbf{F}$ or its negation $\neg f$. f and $\neg f$ are said to be complementary. For a literal l, by $\neg l$, we mean its complement. A *fluent formula* is a propositional formula constructed from the set of literals using operators $\wedge$, $\vee$, and/or $\neg$. To describe an action theory, propositions of the following forms are used:

$$\textbf{initially}(l) \tag{1}$$

$$\textbf{executable}(a, \psi) \tag{2}$$

$$\textbf{causes}(a, l, \phi) \tag{3}$$

$$\textbf{if}(l, \varphi) \tag{4}$$

$$\textbf{determines}(a, \theta) \tag{5}$$

where $a \in \mathbf{A}$ is an action, l is a literal, and $\psi, \phi, \varphi, \theta$ are sets of literals[3].

The initial situation is described by a set of propositions (1), called *v-propositions*. (1) says that l holds in the initial situation. A proposition of form (2) is called *executability condition*. It says that a is executable in any situation in which ψ holds (the precise meaning of *hold* will be given later). A proposition (3), called a *dynamic*

[3] A set of literals is interpreted as the conjunction of its members. The empty set $\emptyset$ denotes *true*.

https://doi.org/10.1017/S1471068406002948 Published online by Cambridge University Press

causal law, represents a conditional effect of an action. It says that performing a in a situation in which ϕ holds causes l to hold in the successor situation. A proposition (4), called a *static causal law*, states that l holds in any situation in which φ holds. A *knowledge proposition* (or *k-proposition* for short) (5) states that the values of literals in θ, sometimes referred to as *sensed-literals*, will be known after a is executed. Because the execution of a will determine the truth value of at least one fluent, without loss of generality, we assume that θ contains at least two literals. Furthermore, we require that if θ is not a set of two contrary literals f and $\neg f$ then the literals in θ are mutually exclusive, i.e.,

1. for every pair of literals g and g' in θ, $g \neq g'$, the theory contains the static causal law

$$\mathbf{if}(\neg g', \{g\})$$

and

2. for every literal g in θ, the theory contains the static causal law

$$\mathbf{if}(g, \{\neg g' \mid g' \in \theta \setminus \{g\}\}).$$

For convenience, we use the abbreviation

$$\mathbf{oneof}(\theta)$$

to denote the above set of static causal laws. Apart from this, we will sometime write

$$\mathbf{determines}(a, f)$$

to stand for

$$\mathbf{determines}(a, \{f, \neg f\}).$$

Actions appearing in (3) and (5) are called non-sensing actions and sensing actions, respectively. In this paper, we assume that they are disjoint from each other. In addition, we also assume that each sensing action appears in at most one k-proposition.

An *action theory* is given by a pair $(\mathscr{D}, \mathscr{I})$ where $\mathscr{D}$ is a set of propositions (2)–(5) and $\mathscr{I}$ is a set of propositions (1). $\mathscr{D}$ and $\mathscr{I}$ are called the *domain description* and *initial situation*, respectively. A *planning problem instance* is a 3-tuple $(\mathscr{D}, \mathscr{I}, \mathscr{G})$, where $(\mathscr{D}, \mathscr{I})$ is an action theory and $\mathscr{G}$ is a *conjunction* of fluent literals. It is worth mentioning that with a proper set of rules for checking the truth value of a fluent formula (see e.g. (Son et al. 2005a)), the framework and all results presented in this paper can be extended to allow $\mathscr{G}$ to be an arbitrary fluent formula as well.

https://doi.org/10.1017/S1471068406002948 Published online by Cambridge University Press

Example 2

The planning problem instance $\mathscr{P}_1 = (\mathscr{D}_1, \mathscr{I}_1, \mathscr{G}_1)$ in Example 1 can be represented as follows.

$$\mathscr{D}_1 = \left\{ \begin{array}{l} \textbf{executable}(check, \{\}) \\ \textbf{executable}(push\_up, \{closed\}) \\ \textbf{executable}(push\_down, \{open\}) \\ \textbf{executable}(flip\_lock, \{\neg open\}) \\ \\ \textbf{causes}(push\_down, closed, \{\}) \\ \textbf{causes}(push\_up, open, \{\}) \\ \textbf{causes}(flip\_lock, locked, \{closed\}) \\ \textbf{causes}(flip\_lock, closed, \{locked\}) \\ \\ \textbf{oneof}(\{open, locked, closed\}) \\ \\ \textbf{determines}(check, \{open, closed, locked\}) \end{array} \right\}$$

$$\mathscr{I}_1 = \{ \textbf{initially}(\neg open) \}$$

$$\mathscr{G}_1 = \{locked\}$$

□

Remark 1

For an action theory $(\mathscr{D}, \mathscr{I})$, $\textbf{if}(l, \emptyset) \in \mathscr{D}$ implies that literal l holds in every situation. Since l is always true, queries about the truth value of l (or $\neg l$) have a trivial answer and the theory can be simplified by removing all instances of l in other propositions. Furthermore, if the theory also contains a dynamic law of the form $\textbf{causes}(a, \neg l, \phi)$ then the execution of a in a state satisfying ϕ will result in an inconsistent state of the world. Thus, the introduction of l in the action theory is either redundant or erroneous. For this reason, without loss of generality, we will assume that action theories in this paper do not contain any static causal law (4) with $\varphi = \emptyset$.

Remark 2

Since an empty plan can always be used to achieve an empty goal, we will assume hereafter that planning problem instances considered in this paper have non-empty goals.

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** (Levesque 1996; Lobo et al. 1997; Son and Baral 2001). Notice that an if-then-else statement can be replaced by a case-endcase statement. Besides, if we are only interested in plans with bounded length then whatever can be represented by a while-do statement with a non-empty body can also be represented by a set of case-endcase statements as well. Therefore, in this paper, we limit ourselves to conditional plans with the case-endcase construct

https://doi.org/10.1017/S1471068406002948 Published online by Cambridge University Press

only. Formally, we consider conditional plans defined as follows. We note that our notion of conditional plans in this paper is fairly similar to the ones introduced elsewhere (Levesque 1996; Lobo et al. 1997; Son and Baral 2001).

Definition 1 (*Conditional Plan*)

1. [] is a conditional plan, denoting the empty plan, i.e., the plan containing no action.
2. if a is a non-sensing action and p is a conditional plan then $[a;p]$ is a conditional plan.
3. if a is a sensing action with proposition (5), where $\theta = \{g_1, \ldots, g_n\}$, and p_j's are conditional plans then $[a; \mathbf{cases}(\{g_j \rightarrow p_j\}_{j=1}^{n})]$ is a conditional plan.
4. Nothing else is a conditional plan.

By this definition, clearly a sequence of actions is also a conditional plan. The execution of a conditional plan of the form $[a;p]$, where a is a non-sensing action and p is another conditional plan, is done sequentially, i.e., a is executed first, followed by p. To execute a conditional plan of the form $[a; \mathbf{cases}(\{g_j \rightarrow p_j\}_{j=1}^{n})]$, we first execute a and then evaluate each g_j with respect to our current knowledge. If one of the g_j's, say g_k, holds, we execute the corresponding sub-plan p_k. Observe that because fluent literals in θ are mutually exclusive, such g_k uniquely exists.

Example 3

The following are conditional plans of the action theory in Example 2:

$$p_1 = [push\_down; flip\_lock]$$

$$p_2 = check;\ \mathbf{cases}\begin{pmatrix} open & \rightarrow & [] \\ closed & \rightarrow & [flip\_lock] \\ locked & \rightarrow & [] \end{pmatrix}$$

$$p_3 = check;\ \mathbf{cases}\begin{pmatrix} open & \rightarrow & [push\_down; flip\_lock] \\ closed & \rightarrow & [flip\_lock; flip\_lock; flip\_lock] \\ locked & \rightarrow & [] \end{pmatrix}$$

$$p_4 = check;\ \mathbf{cases}\begin{pmatrix} open & \rightarrow & [] \\ closed & \rightarrow & p_2 \\ locked & \rightarrow & [] \end{pmatrix}$$

Among those, p_2, p_3 and p_4 are conditional plans that achieve the goal $\mathcal{G}_1$[4]. □

In the rest of the paper, the terms "plan" and "conditional plan" will be used alternatively.

2.3 Queries

A query posed to an $\mathcal{A}_K^c$ action theory $(\mathcal{D}, \mathcal{I})$ is of the form

$$\mathbf{knows}\ \ \rho\ \mathbf{after}\ \ p \tag{6}$$

[4] Note that p_2 and p_4 can achieve the goal because the first case "the window is open" *cannot* happen.

https://doi.org/10.1017/S1471068406002948 Published online by Cambridge University Press

or

$$\textbf{whether } \rho \textbf{ after } p \tag{7}$$

where p is a conditional plan and ρ is a fluent formula. Intuitively, the first (resp. second) query asks whether ρ is true (resp. known) after the execution of p from the initial situation.

2.4 0-approximation semantics of $\mathcal{A}_K^c$

We now define an approximation semantics of $\mathcal{A}_K^c$, called 0-approximation, which extends the 0-approximation in Son and Baral (2001) to deal with static causal laws. It is defined by a transition function Φ that maps actions and a-states into sets of a-states (the meaning of a-states will follow). Before providing the formal definition of the transition function, we introduce some notations and terminology.

For a set of literals σ, $\neg\sigma$ denotes the set $\{\neg l \mid l \in \sigma\}$. σ is said to be *consistent* if it does not contain two complementary literals. A literal l (resp. set of literals γ) *holds* in a set of literals σ if $l \in \sigma$ (resp. $\gamma \subseteq \sigma$); l (resp. γ) *possibly holds* in σ if $\neg l \notin \sigma$ (resp. $\neg\gamma \cap \sigma = \emptyset$).

Given a consistent set of literals σ, the truth value of a formula ρ, denoted by $\sigma(\rho)$, is defined as follows. If $\rho \equiv l$ for some literal l then $\sigma(\rho) = \mathtt{T}$ if $l \in \sigma$; $\sigma(\rho) = \mathtt{F}$ if $\neg l \in \sigma$; $\sigma(\rho) = \mathtt{unknown}$ otherwise. If $\rho \equiv \rho_1 \wedge \rho_2$ then $\sigma(\rho) = \mathtt{T}$ if $\sigma(\rho_1) = \mathtt{T}$ and $\sigma(\rho_2) = \mathtt{T}$; $\sigma(\rho) = \mathtt{F}$ if $\sigma(\rho_1) = \mathtt{F}$ or $\sigma(\rho_2) = \mathtt{F}$; $\sigma(\rho) = \mathtt{unknown}$ otherwise. If $\rho \equiv \rho_1 \vee \rho_2$ then $\sigma(\rho) = \mathtt{T}$ if $\sigma(\rho_1) = \mathtt{T}$ or $\sigma(\rho_2) = \mathtt{T}$; $\sigma(\rho) = \mathtt{F}$ if $\sigma(\rho_1) = \mathtt{F}$ and $\sigma(\rho_2) = \mathtt{F}$; $\sigma(\rho) = \mathtt{unknown}$ otherwise. If $\rho \equiv \neg\rho_1$ then $\sigma(\rho) = \mathtt{T}$ if $\sigma(\rho_1) = \mathtt{F}$; $\sigma(\rho) = \mathtt{F}$ if $\sigma(\rho_1) = \mathtt{T}$; $\sigma(\rho) = \mathtt{unknown}$ otherwise.

We say that ρ is known to be true (resp. false) in σ and write $\sigma \models \rho$ (resp. $\sigma \models \neg\rho$) if $\sigma(\rho) = \mathtt{T}$ (resp. $\sigma(\rho) = \mathtt{F}$). When $\sigma \models \rho$ or $\sigma \models \neg\rho$ we say that ρ is *known* in σ; otherwise, ρ is *unknown* in σ. We will say that ρ holds in σ if it is known to be true in σ.

A set of literals σ satisfies a static causal law (4) if either (i) φ does not hold in σ; or (ii) l holds in σ (i.e., φ holds in σ implies that l holds in σ). By $Cl_{\mathscr{D}}(\sigma)$, we denote the smallest set of literals that includes σ and satisfies all static causal laws in $\mathscr{D}$. Note that $Cl_{\mathscr{D}}(\sigma)$ might be inconsistent but it is unique (*see* Lemma 1, Appendix A).

An *interpretation* I of a domain description $\mathscr{D}$ is a complete and consistent set of literals in $\mathscr{D}$, i.e., for every fluent $f \in \mathbf{F}$, (i) $f \in I$ or $\neg f \in I$; and (ii) $\{f, \neg f\} \nsubseteq I$.

A *state* s is an interpretation satisfying all static causal laws in $\mathscr{D}$. An action a is *executable* in s if there exists an executability condition (2) such that ψ holds in s. For a non-sensing action a executable in s, let

$$E(a,s) = \{l \mid \exists \text{ a dynamic causal law (3) such that } \phi \text{ holds in } s\} \tag{8}$$

The set $E(a,s)$ is often referred to as the *direct effects* of a. When the agent has complete information about the world, the set of possible next states after the execution of a in s, denoted by $Res^c_{\mathscr{D}}(a,s)$, is defined as follows.

https://doi.org/10.1017/S1471068406002948 Published online by Cambridge University Press

Definition 2 (*Possible Next States, (McCain and Turner 1995)*)
Let $\mathscr{D}$ be a domain description. For any state s and non-sensing action a executable in s, $Res^c_{\mathscr{D}}(a,s) = \{s' \mid s' \text{ is a state such that } s' = Cl_{\mathscr{D}}(E(a,s) \cup (s \cap s'))\}$.

The intuitive meaning of this definition is that a literal l holds in a possible next state s' of s after a is executed iff either (i) it is a direct effect of a, i.e., $l \in E(a,s)$ (ii) it holds by inertia, i.e., $l \in (s \cap s')$, or (iii) it is an indirect effect[5] of a, i.e., l holds because of the operator $Cl_{\mathscr{D}}$.

Note that the $Res^c_{\mathscr{D}}$-function can be *non-deterministic*, i.e., $Res^c_{\mathscr{D}}(a,s)$ might contain more than one element. The following example illustrates this point.

Example 4
Consider the following domain description

$$\mathscr{D}_2 = \left\{ \begin{array}{l} \textbf{executable}(a, \{\}) \\ \textbf{causes}(a, f, \{\}) \\ \textbf{if}(g, \{f, \neg h\}) \\ \textbf{if}(h, \{f, \neg g\}) \\ \textbf{if}(k, \{\neg f\}) \end{array} \right\}$$

Let $s = \{\neg f, \neg g, \neg h, k\}$. Clearly s is a state since it satisfies all static laws in $\mathscr{D}_2$. Executing a in s results in two possible next states

$$Res^c_{\mathscr{D}_2}(a,s) = \{\{f, \neg g, h, k\}, \{f, g, \neg h, k\}\}$$

In the first possible next state $s_1 = \{f, \neg g, h, k\}$, f holds because it is a direct effect of a, i.e., $f \in E(a,s)$; $\neg g$ and k hold because of inertia ($s \cap s_1 = \{\neg g, k\}$); and h holds because it is an indirect effect of a (in particular, h holds because of the static causal law $\textbf{if}(h, \{f, \neg g\})$).

Likewise, we can explain why each literal in the second possible next state holds. □

Definition 3 (*Consistent Domains*)
A domain description $\mathscr{D}$ is *consistent* if for every state s and action a executable in s, $Res^c_{\mathscr{D}}(a,s) \neq \emptyset$.

In the presence of incomplete information, an agent, however, does not always know exactly which state it is currently in. One possible way to deal with this problem is to represent the agent knowledge by a set of possible states (a.k.a. belief state) that are consistent with the agent's current knowledge and extend Definition 2 to define a mapping from pairs of actions and belief states into belief states as in Baral et al. (2000b). The main problem with this approach is its high complexity (Baral et al. 2000a), even for the computation of what is true/false after the execution of one action. We address this problem by defining an approximation of the set of states in Definition 2 as follows.

First, we relax the notion of a state in Definition 2 to be an approximate state defined as follows.

[5] Indirect effects are those caused by static causal laws.

https://doi.org/10.1017/S1471068406002948 Published online by Cambridge University Press

Definition 4 (*Approximate State*)
A consistent set of literals δ is called an approximate state (or *a-state*, for short) if δ satisfies all static causal laws in $\mathscr{D}$.

Intuitively, δ represents the (possibly incomplete) current knowledge of the agent, i.e., it contains all fluent literals that are known to be true to the agent. When δ is a subset of some state s, we say that it is *valid*. An action a is *executable* in δ if there exists an executability condition (2) in $\mathscr{D}$ such that ψ holds in δ.

Next, we define what are the possible next a-states after the execution of an action a in a given a-state δ, provided that a is executable in δ. Consider the case that a is a non-sensing action. Let

$$e(a,\delta) = Cl_{\mathscr{D}}(\{l \mid \exists \text{ a dynamic causal law (3) such that } \phi \text{ holds in } \delta\}) \tag{9}$$

and

$$pc(a,\delta) = \bigcup_{i=0}^{\infty} pc^i(a,\delta) \tag{10}$$

where

$$pc^0(a,\delta) = \{l \mid \exists \text{ a dynamic causal law (3) s.t. } l \notin \delta \text{ and } \phi \text{ possibly holds in } \delta\} \tag{11}$$

and for $i \geqslant 0$,

$$pc^{i+1}(a,\delta) = pc^i(a,\delta) \cup \{l \mid \exists \text{ a static causal law (4) s.t. } l \notin \delta, \varphi \cap pc^i(a,\delta) \neq \emptyset, \text{ and } \varphi \text{ possibly holds in } e(a,\delta)\} \tag{12}$$

Intuitively, $e(a,\delta)$ and $pc(a,\delta)$ denote what *definitely holds* and what *may change* in the next situation respectively[6]. Specifically, $l \in e(a,\delta)$ means that l holds in the next situation and $l \in pc(a,\delta)$ means that l is not in δ but possibly holds in the next situation. This implies that $\delta \setminus \neg pc(a,\delta)$ is an approximation of the set of literals that hold by inertia after the execution of a in δ. Taking into account the effects of the static causal laws, we have that the set of literals $\delta' = Cl_{\mathscr{D}}(e(a,\delta) \cup (\delta \setminus \neg pc(a,\delta)))$ must hold in the next situation. This leads us to the following definition of the possible next a-states after a non-sensing action gets executed.

Definition 5 (*0-Result Function*)
For every a-state δ and non-sensing action a executable in δ, let

$$\delta' = Cl_{\mathscr{D}}(e(a,\delta) \cup (\delta \setminus \neg pc(a,\delta))).$$

Define

1. $Res_{\mathscr{D}}(a,\delta) = \{\delta'\}$ if δ' is consistent.
2. $Res_{\mathscr{D}}(a,\delta) = \emptyset$ if δ' is inconsistent.

The next examples illustrate this definition.

[6] Note that the operator $Cl_{\mathscr{D}}$ is used in the definition of $e(a,\delta)$ to *maximize* what definitely holds in the next situation.

https://doi.org/10.1017/S1471068406002948 Published online by Cambridge University Press

Example 5

Consider the domain description $\mathscr{D}_1$ in Example 2. Let $\delta = \{\neg open, closed, \neg locked\}$. We can easily check that δ is an a-state of $\mathscr{D}_1$. We have

$$e(flip\_lock, \delta) = Cl_{\mathscr{D}_1}(\{locked\}) = \{\neg open, \neg closed, locked\}$$

and

$$pc^0(flip\_lock, \delta) = \{locked\}$$

Because **if**$(\neg open, \{locked\}) \in \mathscr{D}_1$, and **if**$(\neg closed, \{locked\}) \in \mathscr{D}_1$, by (12), we have

$$pc^1(flip\_lock, \delta) = \{locked, \neg closed\}$$

Note that $\neg open \notin pc^1(flip\_lock, \delta)$ because it is already in δ.

It is easy to see that $pc^i(flip\_lock, \delta) = pc^1(flip\_lock, \delta)$ for all $i > 1$. Hence, we have

$$pc(flip\_lock, \delta) = \bigcup_{i=0}^{\infty} pc^i(flip\_lock, \delta) = \{\neg closed, locked\}$$

Accordingly, we have

$$Res_{\mathscr{D}_1}(flip\_lock, \delta) = \{Cl_{\mathscr{D}_1}(e(flip\_lock, \delta) \cup (\delta \setminus \neg pc(flip\_lock, \delta)))\} =$$

$$\{Cl_{\mathscr{D}_1}(\{\neg open, \neg closed, locked\})\} = \{\{\neg open, \neg closed, locked\}\}$$

□

Example 6

For the domain description $\mathscr{D}_2$ in Example 4, we have

$$e(a, s) = Cl_{\mathscr{D}_2}(\{f\}) = \{f\}$$

$$pc^0(a, s) = \{f\}$$

As **if**$(g, \{f, \neg h\}) \in \mathscr{D}_2$ and **if**$(h, \{f, \neg g\}) \in \mathscr{D}_2$, we have

$$pc^1(a, s) = \{f, g, h\}$$

Note that $k \notin pc^1(a, s)$ since $\neg f$ does not hold in $e(a, s)$. We can check that $pc^i(a, s) = pc^1(a, s)$ for all $i > 1$. Hence, we have

$$pc(a, s) = \{f, g, h\}$$

As a result, we have

$$Res_{\mathscr{D}_2}(a, s) = \{Cl_{\mathscr{D}_2}(e(a, s) \cup (s \setminus \neg pc(a, s)))\} = \{Cl_{\mathscr{D}_2}(\{f, k\})\} = \{\{f, k\}\}$$

□

The following proposition shows that when a non-sensing action is executed, the *Res*-function is *deterministic* in the sense that it returns at most one possible next a-state; furthermore, it is "sound" with respect to the Res^c-function.

https://doi.org/10.1017/S1471068406002948 Published online by Cambridge University Press

Proposition 1
Let $\mathscr{D}$ be a consistent domain description. For any state s, a-state $\delta \subseteq s$, and non-sensing action a executable in δ, there exists an a-state δ' such that (i) $Res_{\mathscr{D}}(a, \delta) = \{\delta'\}$, and (ii) δ' is a subset of every state $s' \in Res^{c}_{\mathscr{D}}(a, s)$.

Proof
see Appendix A. □

We have specified what are the possible next a-states after a non-sensing action is performed. Let us move to the case when a sensing action is executed. Consider an a-state δ and a sensing action a with k-proposition (5) in $\mathscr{D}$. Intuitively, after a is executed, the agent will know the values of literals in θ. Thus, the set of possible next a-states can be defined as follows.

Definition 6 (*0-Result Function*)
For every a-state δ and sensing action a with proposition (5) such that a is executable in δ,

$$Res_{\mathscr{D}}(a, \delta) = \{Cl_{\mathscr{D}}(\delta \cup \{g\}) \mid g \in \theta \text{ and } Cl_{\mathscr{D}}(\delta \cup \{g\}) \text{ is consistent}\}$$

Roughly speaking, executing a will result in several possible next a-states, in each of which exactly one sensed-literal in θ holds. However, some of them might be inconsistent with what is currently known. For example, if the security robot in Example 1 knows that the window is not open then after it *checks* the window, it should not consider the case that the window is *open* because this is inconsistent with its current knowledge. Thus, in defining the set of possible next a-states resulting from the execution of a sensing action, we need to exclude such inconsistent a-states. The following example illustrates this.

Example 7
Consider again the domain description $\mathscr{D}_1$ in Example 2 and an a-state $\delta_1 = \{\neg open\}$.
We have

$$Cl_{\mathscr{D}_1}(\delta_1 \cup \{open\}) = \{open, \neg open, closed, \neg closed, locked, \neg locked\} = \delta_{1,1}$$

$$Cl_{\mathscr{D}_1}(\delta_1 \cup \{closed\}) = \{\neg open, closed, \neg locked\} = \delta_{1,2}$$

$$Cl_{\mathscr{D}_1}(\delta_1 \cup \{locked\}) = \{\neg open, \neg closed, locked\} = \delta_{1,3}$$

Among those, $\delta_{1,1}$ is inconsistent. Therefore, we have

$$Res_{\mathscr{D}_1}(check, \delta_1) = \{\delta_{1,2}, \delta_{1,3}\}$$

□

The next proposition shows that if a sensing action is performed in a valid a-state then the set of possible next a-states will contain at least one valid a-state. This corresponds to the fact that if the current knowledge of the world of the agent is consistent with the state of the world, it will remain consistent with the state of the world after the agent acquires additional knowledge through the execution of a sensing action.

https://doi.org/10.1017/S1471068406002948 Published online by Cambridge University Press

Proposition 2
Let $\mathscr{D}$ be a consistent domain description. For any a-state δ, and a sensing action a executable in δ, if δ is valid then $Res_{\mathscr{D}}(a,\delta)$ contains at least one valid a-state.

Proof
see Appendix A. □

The transition function Φ that maps actions and a-states into sets of a-states is defined as follows.

Definition 7 (*Transition Function*)
Given a domain description $\mathscr{D}$, for any action a and a-state δ,

1. if a is not executable in δ then
$$\Phi(a,\delta) = \perp$$
2. otherwise,
$$\Phi(a,\delta) = Res_{\mathscr{D}}(a,\delta)$$

The transition function Φ returns the set of possible next a-states after performing a single action in a given a-state. We now extend it to define the set of possible next a-states after the execution of a plan. The extended transition function, called $\hat{\Phi}$, is given in the following definition.

Definition 8 (*Extended Transition Function*)
Given a domain description $\mathscr{D}$, for any plan p and a-state δ,

1. if $p = []$ then
$$\hat{\Phi}(p,\delta) = \{\delta\}$$
2. if $p = [a;q]$, where a is a non-sensing action and q is a sub-plan, then
$$\hat{\Phi}(p,\delta) = \begin{cases} \perp & \text{if } \Phi(a,\delta) = \perp \\ \bigcup_{\delta' \in \Phi(a,\delta)} \hat{\Phi}(q,\delta') & \text{otherwise} \end{cases}$$
3. if $p = [a;\mathbf{cases}(\{g_j \rightarrow p_j\}_{j=1}^{n})]$, where a is a sensing action and p_j's are sub-plans, then
$$\hat{\Phi}(p,\delta) = \begin{cases} \perp & \text{if } \Phi(a,\delta) = \perp \\ \bigcup_{1 \leqslant j \leqslant n, \delta' \in \Phi(a,\delta), g_j \text{ holds in } \delta'} \hat{\Phi}(p_j,\delta') & \text{otherwise} \end{cases}$$

where, by convention, $\ldots \cup \perp \cup \ldots = \perp$.

Items (2) and (3) of the above definition deserve some elaboration.

Remark 3
During the execution of a plan p, when a non-sensing action a is encountered (Item 2), by Definitions 5 and 7, there are three possibilities: $\Phi(a,\delta) = \perp$, $\Phi(a,\delta) = \emptyset$, or $\Phi(a,\delta) = \{\delta'\}$ for some a-state δ'. If the first case occurs then the result of execution of p in δ by the definition is also $\perp$. In this case, we say that p is not executable in δ; otherwise, p is *executable* in δ. If the second case occurs then by the definition, $\hat{\Phi}(p,\delta) = \emptyset$. One may notice that, by Proposition 1, this case takes place only if there

https://doi.org/10.1017/S1471068406002948 Published online by Cambridge University Press

exists no state s such that $\delta \subseteq s$ (i.e., δ is invalid), or the domain is inconsistent. When $\Phi(a, \delta) = \{\delta'\}$, then the result of the execution of p in δ is exactly as the result of the execution of the rest of p in δ'.

Remark 4

If $p = [a; \mathbf{cases}(\{g_j \rightarrow p_j\}_{j=1}^{n})]$, where a is a sensing action and p_j's are sub-plans (Item 3), and $\Phi(a, \delta) \neq \perp$ then by Definitions 6 and 7, we know that $\Phi(a, \delta)$ may contain several a-states δ_j's. Each δ_j corresponds to an a-state in which literal g_j holds. Therefore, we define $\hat{\Phi}(p, \delta)$ to be the union of the sets of possible a-states that are the results of the execution of p_j in δ_j. Note that when we add g_j to the current state δ to generate δ_j, we *assume* that g_j holds. However, if later on, during the execution of the rest of p, which is p_j, we discover that $\hat{\Phi}(p_j, \delta_j) = \emptyset$, then our assumption about g_j is not correct. Therefore, such a δ_j contributes nothing to the set of possible a-states of $\hat{\Phi}(a, \delta)$. To see how this can happen, consider the following domain description:

$$\mathscr{D}_3 = \left\{ \begin{array}{l} \mathbf{executable}(a, \{\}) \\ \mathbf{executable}(b, \{\}) \\ \\ \mathbf{causes}(b, h, \{\}) \\ \mathbf{if}(f, \{g, h\}) \\ \mathbf{if}(f, \{g, \neg h\}) \\ \mathbf{determines}(a, f) \end{array} \right\}$$

and suppose that the set of fluents is $\{f, g, h\}$. Let us see what are the final possible a-states after the execution of plan $p = [a; \mathbf{cases}(\{f \rightarrow b; \neg f \rightarrow b\})]$ in a-state $\delta = \{g\}$ as defined by the extended transition function.

When a is performed, we generate two possible next a-states $\delta_1 = \{g, f\}$, and $\delta_2 = \{g, \neg f\}$. Executing b in δ_2 results in no possible next a-state because $Cl_{\mathscr{D}_3}(\{g, \neg f, h\}) = \{g, \neg f, h, f\}$ is not consistent. This means that $\Phi(b, \delta_2)$, and thus $\hat{\Phi}([b], \delta_2)$, become $\emptyset$. Therefore, the set of possible final a-states is $\hat{\Phi}(p, \delta) = \hat{\Phi}([b], \delta_1) = \{\{f, g, h\}\}$.

Note that in this example, we did not notice that δ_2 is inconsistent at the time the action a was performed. Rather, its inconsistency was only realized after the execution of b. In other words, our assumption that $\neg f$ holds was not correct.

Similarly to the execution of a non-sensing action, when a sensing action a is performed, by Proposition 2, $\Phi(a, \delta) = \emptyset$ only if the domain is inconsistent or δ is invalid.

The above remarks imply that in some cases, for a plan p and an a-state δ, $\hat{\Phi}(p, \delta)$ may be empty. Intuitively, this is because either δ is invalid or the domain is inconsistent. We will show that under reasonable assumptions about δ and the domain, this cannot happen.

Definition 9 (*Consistent Action Theories*)

An action theory $(\mathscr{D}, \mathscr{I})$ is *consistent* if $\mathscr{D}$ is consistent and its initial a-state, defined by $Cl_{\mathscr{D}}(\{l \mid \mathbf{initially}(l) \in \mathscr{I}\})$, is valid.

https://doi.org/10.1017/S1471068406002948 Published online by Cambridge University Press

The next proposition says that the execution of an executable plan from a valid a-state of a consistent action theory will result in at least one valid a-state.

Proposition 3
Let $(\mathscr{D}, \mathscr{I})$ be a consistent action theory and let δ be its initial a-state. For every conditional plan p, if $\hat{\Phi}(p, \delta) \neq \perp$ then $\hat{\Phi}(p, \delta)$ contains at least one valid a-state.

Proof
See Appendix A. □

The above proposition implies that if the action theory $(\mathscr{D}, \mathscr{I})$ is consistent and δ is its initial a-state then the execution of p in δ will yield at least a valid trajectory[7], provided that p is executable in δ. This is consistent with the fact that if the initial a-state is complete (i.e., if we have complete information) then the execution of an executable plan in the initial a-state would return a valid trajectory. *From now on, we only consider consistent action theories.*

We next define the entailment relationship between action theories and queries.

Definition 10 (*Entailment*)
Let $(\mathscr{D}, \mathscr{I})$ be an action theory and δ be its initial a-state. For a plan p and a fluent formula ρ, we say that

- $(\mathscr{D}, \mathscr{I})$ entails the query **knows** ρ **after** p and write

$$\mathscr{D} \models_{\mathscr{I}} \textbf{knows} \ \rho \ \textbf{after} \ p$$

  if $\hat{\Phi}(p, \delta) \neq \perp$ and ρ is true in every a-state in $\hat{\Phi}(p, \delta)$; and
- $(\mathscr{D}, \mathscr{I})$ entails the query **whether** ρ **after** p and write

$$\mathscr{D} \models_{\mathscr{I}} \textbf{whether} \ \rho \ \textbf{after} \ p$$

  if $\hat{\Phi}(p, \delta) \neq \perp$ and ρ is known in every a-state in $\hat{\Phi}(p, \delta)$.

Example 8
For the action theory $(\mathscr{D}_1, \mathscr{I}_1)$ in Example 2, we will show that

$$\mathscr{D}_1 \models_{\mathscr{I}_1} \textbf{knows} \ \ locked \ \textbf{after} \ \ p_2 \tag{13}$$

where p_2 is given in Example 3.

Let $p_{2,1} = []$, $p_{2,2} = [flip\_lock]$ and $p_{2,3} = []$. It is easy to see that the initial a-state of $(\mathscr{D}_1, \mathscr{I}_1)$ is $\delta_1 = \{\neg open\}$.

It follows from Example 7 that

$$\Phi(check, \delta_1) = \{\delta_{1,2}, \delta_{1,3}\}$$

On the other hand, we have

$$\hat{\Phi}(p_{2,2}, \delta_{1,2}) = \{\{locked, \neg open, \neg closed\}\}$$

and

$$\hat{\Phi}(p_{2,3}, \delta_{1,3}) = \{\{locked, \neg open, \neg closed\}\}$$

[7] A trajectory is an alternate sequence of a-states and actions, $\delta_0 a_1 \delta_1 a_2 \ldots a_n \delta_n$, such that $\delta_i \in \Phi(a_i, \delta_{i-1})$ for $i = 1, \ldots, n$; A trajectory is valid if δ_i's are valid a-states.

https://doi.org/10.1017/S1471068406002948 Published online by Cambridge University Press

Therefore, we have

$$\hat{\Phi}(p_2, \delta_1) = \hat{\Phi}(p_{2,2}, \delta_{1,2}) \cup \hat{\Phi}(p_{2,3}, \delta_{1,3}) = \{\{locked, \neg open, \neg closed\}\}$$

Since *locked* is true in $\{locked, \neg open, \neg closed\}$, we have (13) holds. On the other hand, because *closed* is false in $\{locked, \neg open, \neg closed\}$, we have

$$\mathscr{D}_1 \not\models_{\mathscr{I}_1} \textbf{knows}\ \ closed\ \textbf{after}\ \ p_2 \qquad \text{but} \qquad \mathscr{D}_1 \models_{\mathscr{I}_1} \textbf{knows}\ \ \neg closed\ \textbf{after}\ \ p_2.$$

Likewise, we can prove that

$$\mathscr{D}_1 \models_{\mathscr{I}_1} \textbf{knows}\ \ locked\ \textbf{after}\ \ p_3 \qquad \text{and} \qquad \mathscr{D}_1 \models_{\mathscr{I}_1} \textbf{knows}\ \ locked\ \textbf{after}\ \ p_4.$$

Definition 11 (*Solutions*)
A plan p is called a *solution* to a planning problem instance $\mathscr{P} = (\mathscr{D}, \mathscr{I}, \mathscr{G})$ iff

$$\mathscr{D} \models_{\mathscr{I}} \textbf{knows}\ \ \mathscr{G}\ \textbf{after}\ \ p$$

When p is a solution to $\mathscr{P}$, we say that p is a plan that *achieves* the goal $\mathscr{G}$.

According to this definition, it is easy to see that plans p_2, p_3, and p_4 in Example 3 are solutions to $\mathscr{P}_1 = (\mathscr{D}_1, \mathscr{I}_1, \mathscr{G}_1)$ in Example 2.

2.5 Properties of the 0-approximation

We will now discuss some properties of the 0-approximation. For a domain description $\mathscr{D}$, we define the size of $\mathscr{D}$ to be the sum of *(1)* the number of fluents; *(2)* the number of actions; and *(3)* the number of propositions in $\mathscr{D}$. The size of a planning problem instance $\mathscr{P} = (\mathscr{D}, \mathscr{I}, \mathscr{G})$ is defined as the size of $\mathscr{D}$. The size of a plan p, denoted by $size(p)$, is defined as follows:

1. $size([]) = 0$;
2. $size([a; p]) = 1 + size(p)$ if a is a non-sensing action and p is a plan; and
3. $size([a; \textbf{cases}(\{g_j \rightarrow p_j\}_{j=1}^n)]) = 1 + \Sigma_{j=1}^n (1 + size(p_j))$ if a is a sensing action and p_j's are plans.

Then, we have the following proposition.

Proposition 4
For a domain description $\mathscr{D}$, an action a, and an a-state δ, computing $\Phi(a, \delta)$ can be done in polynomial time in the size of $\mathscr{D}$.

Proof
See Appendix A. □

From this proposition, we have the following corollary.

Corollary 2.1
Determining whether or not a plan p is a solution of the planning problem instance $\mathscr{P} = (\mathscr{D}, \mathscr{I}, \mathscr{G})$ from an a-state δ can be done in polynomial time in the size of p and $\mathscr{P}$.

https://doi.org/10.1017/S1471068406002948 Published online by Cambridge University Press

Definition 12
The *conditional planning problem* is defined as follows.

- *Given:* A planning problem instance $\mathscr{P} = (\mathscr{D}, \mathscr{I}, \mathscr{G})$ of size n and a polynomial $Q(n) \geqslant n$;
- *Determine:* whether there exists a conditional plan, whose size is bounded by $Q(n)$, that achieves $\mathscr{G}$ from $\mathscr{I}$ (with respect to Definition 11).

Theorem 1
The conditional planning problem is **NP**-complete.

Proof
See Appendix A. □

The above theorem shows that planning using the 0-approximation has lower complexity than planning with respect to the full semantics. Here, by the full semantics we mean the possible world semantics extended to domains with sensing actions. Yet, the price one has to pay is the incompleteness of this approximation, i.e., there are planning instances which have solutions with respect to the full semantics but do not have solutions with respect to the approximation. This can be seen in the following example.

Example 9
Consider the planning problem instance $\mathscr{P}=(\mathscr{D}, \mathscr{I}, \mathscr{G})$ with

$$\mathscr{D} = \{\textbf{causes}(a, f, \{g\}), \textbf{causes}(a, f, \{\neg g\})\},\ \mathscr{I} = \emptyset,\ \text{and}\ \mathscr{G} = \{f\}.$$

We can easily check that $p = [a]$ is a plan achieves f from every initial situation (with respect to the possible world semantics developed for $\mathscr{A}_K^c$ in (Baral et al. 2000b)). However, p is not a solution with respect to Definition 11, because $\mathscr{D} \not\models_{\mathscr{I}}$ **knows** f **after** a.

The above example highlights the main weakness of this approximation in that it does not allow for reasoning by cases for non-sensing actions or in the presence of disjunctive initial situation. In our experiments with the benchmarks, we observe that most of the benchmarks that our planner could not solve fall into the second category, i.e., they require the capability of reasoning with disjunctive information about the initial state. Given that we do not consider action theories with disjunctive initial state, this should not come as a surprise.

3 A logic programming based conditional planner

This section describes an answer set programming based conditional planner, called ASCP. Given a planning problem instance $\mathscr{P} = (\mathscr{D}, \mathscr{I}, \mathscr{G})$, we translate it into a logic program $\pi_{h,w}(\mathscr{P})$, where h and w are two input parameters whose meanings will become clear shortly, and then use an answer set solver (e.g., `smodels` or `cmodels`) to compute its answer sets. The answer sets of $\pi_{h,w}(\mathscr{P})$ represent solutions to $\mathscr{P}$. Our intuition behind this task rests on the observation that each plan p (Definition 1)

https://doi.org/10.1017/S1471068406002948 Published online by Cambridge University Press

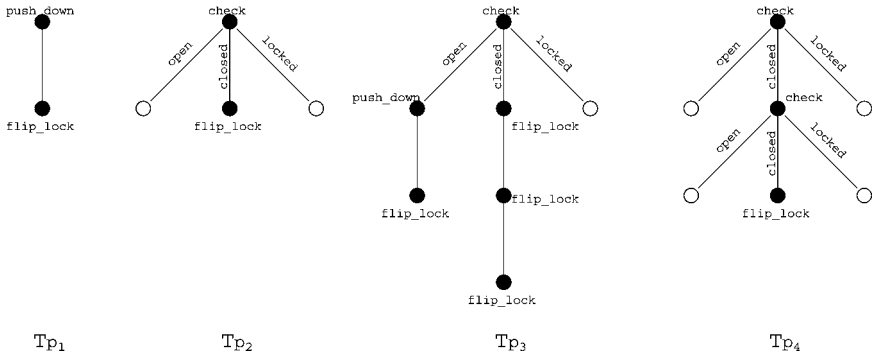

Fig. 1. Sample plan trees

corresponds to a labeled plan tree T_p defined as below:

- If $p = []$ then T_p is a tree with a single node.
- If $p = [a]$, where a is a non-sensing action, then T_p is a tree with a single node and this node is labeled with a.
- If $p = [a; q]$, where a is a non-sensing action and q is a non-empty plan, then T_p is a tree whose root is labeled with a and has only one subtree which is T_q. Furthermore, the link between a and T_q's root is labeled with an empty string.
- If $p = [a; \textbf{cases}(\{g_j \rightarrow p_j\}_{j=1}^{n})]$, where a is a sensing action that determines g_j's, then T_p is a tree whose root is labeled with a and has n subtrees $\{T_{p_j} \mid j \in \{1, \ldots, n\}\}$. For each j, the link from a to the root of T_{p_j} is labeled with g_j.

Observe that each trajectory of the plan p corresponds to a path from the root to a leave of T_p. As an example, Figure 1 depicts the labeled trees for plans p_1, p_2, p_3 and p_4 in Example 3 (black nodes indicate that there exists an action occurring at those nodes, while white nodes indicate that there is no action occurring at those nodes).

For a plan p, let α be the number of leaves of T_p and β be the number of nodes along the longest path from the root to the leaves of T_p. α and β will be called the *width* and *height* of T_p respectively. Suppose w and h are two integers that such that $\alpha \leqslant w$ and $\beta \leqslant h$.

Let us denote the leaves of T_p by $x_1, \ldots, x_\alpha$. We map each node y of T_p to a pair of integers $n_y = (t_y, p_y)$, where t_y is the number of nodes along the path from the root to y, and p_y is defined in the following way.

- For each leaf x_i of T_p, p_{x_i} is an arbitrary integer between 1 and w. Furthermore, there exists a leaf x with p-value of 1, i.e., $p_x = 1$, and there exist no $i \neq j$ such that $p_{x_i} = p_{x_j}$.
- For each interior node y of T_p with children $y_1, \ldots, y_r$, $p_y = \texttt{min}\{p_{y_1}, \ldots, p_{y_r}\}$.

For instance, Figure 2 shows some possible mappings with $h = 4$ and $w = 5$ for the trees in Figure 1. It is easy to see that if $\alpha \leqslant w$ and $\beta \leqslant h$ then such a mapping always exists. Furthermore, from the construction of T_p, independently of how the leaves of T_p are numbered, we have the following properties.

https://doi.org/10.1017/S1471068406002948 Published online by Cambridge University Press

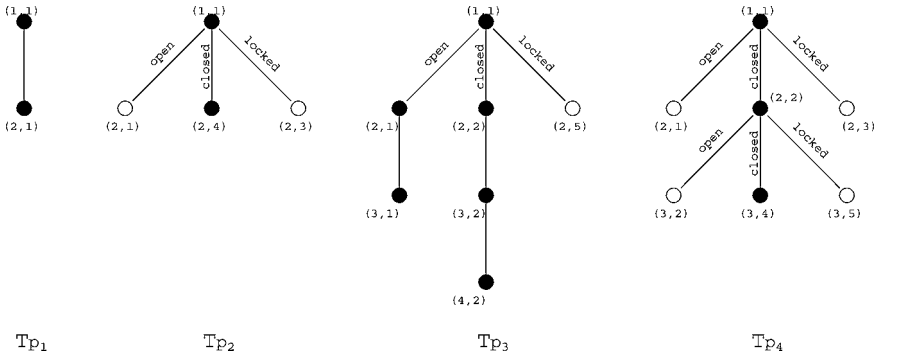

Fig. 2. Possible mappings for the trees in Figure 1

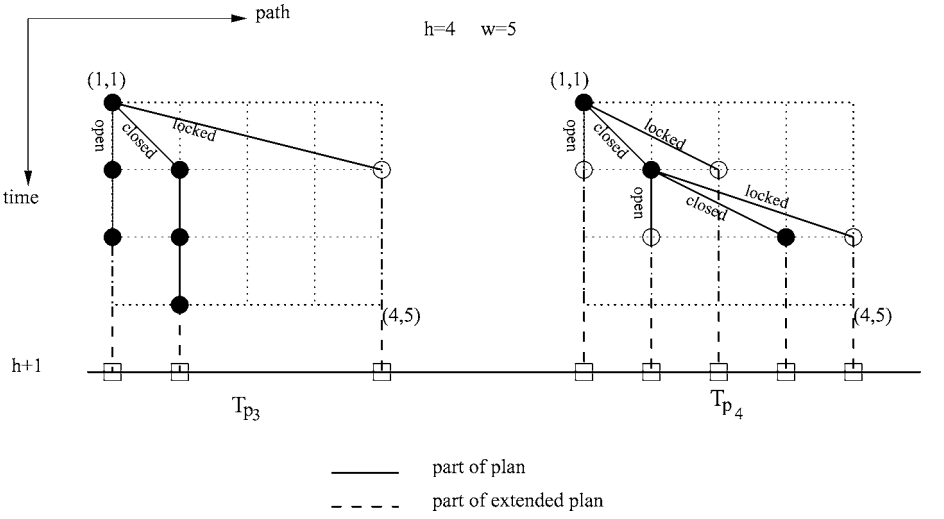

Fig. 3. Grid representation of conditional plans

1. For every node y, $t_y \leqslant h$ and $p_y \leqslant w$.
2. For a node y, all of its children have the same t-value. That is, if y has r children $y_1, \ldots, y_r$ then $t_{y_i} = t_{y_j}$ for every $1 \leqslant i, j \leqslant r$. Furthermore, the p-value of y is the smallest one among the p-values of its children.
3. The root of T_p is always mapped to the pair $(1, 1)$.

Our encoding is based on the above mapping. We observe that a conditional plan p can be represented on a grid $h \times w$ where each node y of T_p is placed at the position (t_y, p_y) relative to the leftmost top corner of the grid. This way, it is guaranteed that the root of T_p is always placed at the leftmost top corner. Figure 3 depicts the 4×5 grid representation of conditional plans T_{p_3} and T_{p_4} in Figure 2. As it can be seen in Figure 3, each path (trajectory) of the plan can end at an arbitrary time point. For example, the leftmost and rightmost trajectories of T_{P_4} end at 2, whereas the others end at 3. On the other hand, to check if the plan is indeed a solution, we need to check the satisfaction of the goal at every leaf node of the plan, that is, at the end of each trajectory. In our encoding, this task is simplified by

https://doi.org/10.1017/S1471068406002948 Published online by Cambridge University Press

extending all the trajectories of the plan so that they have the same height $h+1$ and then checking the goal at the end of each extended trajectory (see Figure 3). Note that an a-state associated with each node on the extended part of each trajectory in our encoding will be guaranteed to be the same as the one associated with the end node of the original trajectory.

We now describe the program $\pi_{h,w}(\mathscr{P})$ in the syntax of `smodels` (for a concrete example, see Appendix C). In $\pi_{h,w}(\mathscr{P})$, variables of sorts *time* and *path* correspond to rows and columns of the grid. Instead of using the predicate $holds(L, T)$ (Dimopoulos et al. 1997; Lifschitz 1999) to denote that a literal L holds at the time T, we use the predicate $holds(L, T, P)$ to represent the fact that L holds at node (T, P) (the time moment T, the path number P on the grid).

The program $\pi_{h,w}(\mathscr{P})$ contains the following elements:

1. **Constants.** There are two constants used in the program h and w which serve as the input parameters of the program. In addition, we have constants to denote fluents, literals and actions in the domain. Due to the fact that `smodels` does not allow symbol $\neg$, to represent a literal constant $\neg f$, we will use $neg(f)$.
2. **Predicates.** The program uses the following predicates.
   - $time(T)$ is true if $1 \leqslant T \leqslant h$.
   - $time1(T_1)$ is true if $1 \leqslant T_1 \leqslant h+1$.
   - $path(P)$ is true if $1 \leqslant P \leqslant w$.
   - $fluent(F)$ is true if F is a fluent.
   - $literal(L)$ is true if L is a literal.
   - $contrary(L, L_1)$ is true if L and L_1 are two complementary literals.
   - $sense(L)$ is true if L is a sensed literal.
   - $action(A)$ is true if A is an action
   - $holds(L, T, P)$ is true if literal L holds at (T, P).
   - $poss(A, T, P)$ is true if action A is executable at (T, P).
   - $occ(A, T, P)$ is true if action A occurs at (T, P). That means the node (T, P) in T_p is labeled with action A.
   - $e(L, T, P)$ is true if literal L is an effect of a non-sensing action occurring at (T, P).
   - $pc(L, T, P)$ is true if literal L may change at $(T+1, P)$.
   - $goal(T, P)$ is true if the goal is satisfied at (T, P).
   - $br(G, T, P, P_1)$ is true if there exists a branch from (T, P) to $(T+1, P_1)$ labeled with G in T_p. For example, in the grid representation of T_{p_3} (Figure 3), we have $br(open, 1, 1, 1)$, $br(closed, 1, 1, 2)$, and $br(locked, 1, 1, 5)$.
   - $used(T, P)$ is true if (T, P) belongs to some extended trajectory of the plan. This allows us to know which paths are used in the construction of the plan and thus to be able to check if the plan satisfies the goal. As an example, for T_{p_3} in Figure 3, we have $used(t, 1)$ for $1 \leqslant t \leqslant 5$, and $used(t, 2)$ and $used(t, 5)$ for $2 \leqslant t \leqslant 5$. The goal satisfaction, hence, will be checked at nodes $used(5, 1)$, $used(5, 2)$, and $used(5, 5)$.
3. **Variables.** The following variables are used in the program:
   - F: a *fluent* variable.

https://doi.org/10.1017/S1471068406002948 Published online by Cambridge University Press

- L and L_1: *literal* variables.
- T and T_1: *time* variables, in ranges $1..h$ and $1..h+1$ respectively,
- G, G_1 and G_2: *sensed—literal* variables.
- A: an *action* variable.
- P, P_1, and P_2: *path* variables, in range $1..w$.

The domains of these variables are declared in `smodels` using the keyword `#domain` (see Appendix C for more details). Observe that the type of each variable has to be declared accordingly if this feature of `smodels` is not used.

4. **Rules.** The program has the following facts to define variables of sort *time* and *path*:

$$\begin{aligned} time(1..h) &\leftarrow \\ time1(1..h+1) &\leftarrow \\ path(1..w) &\leftarrow \end{aligned}$$

For each action a, fluent f, or sensed-literal g in the domain, $\pi_{h,w}(\mathscr{P})$ contains the following facts respectively

$$\begin{aligned} action(a) &\leftarrow \\ fluent(f) &\leftarrow \\ sense(g) &\leftarrow \end{aligned}$$

The remaining rules of $\pi_{h,w}(\mathscr{P})$ are divided into three groups: (i) domain dependent rules; (ii) goal representation and (iii) domain independent rules, which are given next. Note that they are shown in a shortened form in which the following shortening conventions are used.

- Two contrary literal variables are written as L and $\neg L$.
- For a predicate symbol p, and a set γ of literals or actions, we will write $p(\gamma,\ldots)$ to denote the set of atoms $\{p(x,\ldots) \mid x \in \gamma\}$.
- For a literal constant l, $\neg l$ stands for $neg(f)$ (resp. f) if $l = f$ (resp. $l = \neg f$) for some fluent f.

For example, the rule (28) stands for the following rule

$$holds(L, T+1, P) \leftarrow holds(L, T, P), contrary(L, L_1), not\ pc(L_1, T, P)$$

3.1 Domain dependent rules

- **Rules encoding the initial situation.** For each v-proposition (1) in $\mathscr{I}$, $\pi_{h,w}(\mathscr{P})$ contains the fact

$$holds(l, 1, 1) \leftarrow \tag{14}$$

- **Rules encoding actions' executability conditions.** For each executability condition (2) in $\mathscr{D}$, $\pi_{h,w}(\mathscr{P})$ contains the rule

$$poss(a, T, P) \leftarrow holds(\psi, T, P) \tag{15}$$

https://doi.org/10.1017/S1471068406002948 Published online by Cambridge University Press

- **Rules for reasoning about the effect of non-sensing actions.** For each dynamic causal law (3) in D, we add to $\pi_{h,w}(\mathscr{P})$ the following rules:

$$e(l,T,P) \leftarrow occ(a,T,P), holds(\phi,T,P) \quad (16)$$

$$pc(l,T,P) \leftarrow occ(a,T,P), not\ holds(l,T,P), not\ holds(\neg\phi,T,P) \quad (17)$$

  Here, a is a non-sensing action. Its execution changes the world according to the *Res*-function. The first rule, when used along with (22), encodes what definitely holds as the effect of a in the next a-state. The second rule, when used along with (21), describes what would potentially be changed by a (see the definitions of $e(a,\delta)$ and $pc(a,\delta)$ in Subsection 2.4). Note that in the second rule, $not\ holds(\neg\phi,T,P)$ stands for $\{not\ holds(\neg l) \mid l \in \phi\}$, meaning that ϕ possibly holds at (T,P). These rules will be used in cooperation with (23), (27), and (28) to define the next a-state after the execution of a non-sensing action.

- **Rules for reasoning about the effect of sensing actions.** For each k-proposition (5) in $\mathscr{D}$, $\pi_{h,w}(\mathscr{P})$ contains the following rules:

$$\leftarrow occ(a,T,P), not\ br(\theta,T,P,P) \quad (18)$$

$$1\{br(g,T,P,X):new\_br(P,X)\}1 \leftarrow occ(a,T,P),\ (g \in \theta) \quad (19)$$

$$\leftarrow occ(a,T,P), holds(g,T,P),\ (g \in \theta) \quad (20)$$

  The first rule assures that if a sensing action a occurs at (T,P) then there must be a branch from (T,P) to $(T+1,P)$. The second rule ensures that a new branch, corresponding to a new successor a-state, will be created for each literal sensed by the action. The last rule is a constraint that prevents a from taking place if one of the literals sensed by the action is already known. With this rule, the returned plan is guaranteed to be optimal in the sense that a sensing action should not occur if one of the literals sensed by the action already holds. Observe that the semantics of ${\mathscr{A}^c}_K$ does not prevent a sensing action to execute when some of its sensed-fluents is known. For this reason, some solutions to a planning problem instance might not be found using this encoding. However, as we will see later, the program will generate an "equivalent" plan to those solutions. Section 4.2 will elaborate more on this issue.

- **Rules for reasoning about static causal laws.** For each static causal law (4) in $\mathscr{D}$, $\pi_{h,w}(\mathscr{P})$ contains the rules

$$pc(l,T,P) \leftarrow not\ holds(l,T,P), pc(l',T,P), not\ e(\neg\varphi,T,P),\ (l' \in \varphi) \quad (21)$$

$$e(l,T,P) \leftarrow e(\varphi,T,P) \quad (22)$$

$$holds(l,T_1,P) \leftarrow holds(\varphi,T_1,P) \quad (23)$$

  Rules in this group encode the equations (10)-(12) and the operator $Cl_{\mathscr{D}}$.

https://doi.org/10.1017/S1471068406002948 Published online by Cambridge University Press

3.2 Goal representation

The following rules encode the goal and make sure that it is always achieved at the end of every possible branch created by the execution of the plan.

$$goal(T_1, P) \leftarrow holds(\mathcal{G}, T_1, P) \quad (24)$$

$$goal(T_1, P) \leftarrow holds(L, T_1, P), holds(\neg L, T_1, P) \quad (25)$$

$$\leftarrow used(h{+}1, P), not\ goal(h{+}1, P) \quad (26)$$

The first rule says that the goal is satisfied at a node if all of its subgoals are satisfied at that node. The last rule guarantees that if a path P is used in the construction of a plan then the goal must be satisfied at the end of this path, that is, at node $(h+1, P)$.

Rule (25) deserves some explanation. Intuitively, the presence of $holds(L, T, P)$ and $holds(\neg L, T, P)$ indicates that the a-state at the node (T, P) is inconsistent. This means that no action should be generated at this node as inconsistent a-states will be removed by the extended transition function (Definition 8). To achieve this effect[8], we say that the "goal" has been achieved at (T, P). The inclusion of this rule might raise the question: is it possible for the program to generate a plan whose execution yields inconsistent a-states only. Fortunately, due to Proposition 3, this will not be the case for consistent action theories.

3.3 Domain independent rules

- **Rules encoding the effect of non-sensing actions.** Rules (16) – (17) specify what definitely holds and what could potentially be changed in the next a-state as the effect of a non-sensing action. The following rules encode the effect and frame axioms for non-sensing actions.

$$holds(L, T{+}1, P) \leftarrow e(L, T, P) \quad (27)$$

$$holds(L, T{+}1, P) \leftarrow holds(L, T, P), not\ pc(\neg L, T, P) \quad (28)$$

  When used in conjunction with (16) – (17), they define the *Res* function.

- **Inertial rules for sensing actions.** This group of rules encodes the fact that the execution of a sensing action does not change the world. However, there is one-to-one correspondence between the set of sensed literals and the set of possible next a-states after the execution of a sensing action.

$$\leftarrow P_1 < P_2, P_2 < P, br(G_1, T, P_1, P), br(G_2, T, P_2, P) \quad (29)$$

$$\leftarrow P_1 \leqslant P, G_1 \neq G_2, br(G_1, T, P_1, P), br(G_2, T, P_1, P) \quad (30)$$

$$\leftarrow P_1 < P, br(G, T, P_1, P), used(T, P) \quad (31)$$

[8] The same effect can be achieved by (i) introducing a new predicate, say $stop(T, P)$, to represent that the a-state at (T, P) is inconsistent; (ii) adding $not\ stop(T, P)$ in the body of rule (35) to prevent action to occur at (T, P); and (iii) modifying the rule (26) accordingly.

https://doi.org/10.1017/S1471068406002948 Published online by Cambridge University Press

$$used(T+1,P) \leftarrow P_1 < P, br(G,T,P_1,P) \quad (32)$$

$$holds(G,T+1,P) \leftarrow P_1 \leqslant P, br(G,T,P_1,P) \quad (33)$$

$$holds(L,T+1,P) \leftarrow P_1 < P, br(G,T,P_1,P), holds(L,T,P_1) \quad (34)$$

The first three rules make sure that there is no cycle in the plan that we are encoding. The next rule is to mark a node as used if there exists a branch in the plan that coming to that node. This allows us to know which paths on the grid are used in the construction of the plan and thus to be able to check if the plan satisfies the goal (see rule (26)).

The last two rules, along with rule (23), encode the possible next a-state corresponding to the branch denoted by literal G after a sensing action is performed in a state δ. They say that such a-state should contain G (rule (33)) and literals that hold in δ (rule (34)).

Note that because for each literal G sensed by a sensing action a, we create a corresponding branch (rules (18) and (19)), the rules of this group guarantee that all possible next a-states after a is performed are generated.

- **Rules for generating action occurrences.**

$$1\{occ(X,T,P) : action(X)\}1 \leftarrow used(T,P), not\ goal(T,P) \quad (35)$$

$$\leftarrow occ(A,T,P), not\ poss(A,T,P) \quad (36)$$

The first rule enforces exactly one action to take place at a node that was used but the goal has not been achieved. The second one guarantees that only executable actions can occur.

- **Auxiliary Rules.**

$$literal(F) \leftarrow \quad (37)$$

$$literal(\neg F) \leftarrow \quad (38)$$

$$contrary(F,\neg F) \leftarrow \quad (39)$$

$$contrary(\neg F,F) \leftarrow \quad (40)$$

$$new\_br(P,P_1) \leftarrow P \leqslant P_1 \quad (41)$$

$$used(1,1) \leftarrow \quad (42)$$

$$used(T+1,P) \leftarrow used(T,P) \quad (43)$$

The first four rules define literals and contrary literals. Rule (41) says that a newly created branch should outgo to a path number greater than the current path. The last two rules mark nodes that have been used.

4 Properties of ASCP

This section discusses some important properties of ASCP. We begin with how to extract a solution from an answer set returned by ASCP. Then, we argue that ASCP is sound and complete with respect to the 0-approximation semantics. We also show that ASCP can be used as a conformant planner. Finally, we present how to modify ASCP to act as a reasoner.

https://doi.org/10.1017/S1471068406002948 Published online by Cambridge University Press

4.1 Solution extraction

In some previous answer set based planners (Dimopoulos et al. 1997; Eiter et al. 2003; Lifschitz 1999), reconstructing a plan from an answer set for a logic program encoding the planning problem instance is quite simple: we only need to collect the action occurrences in the model and then order them by the time they occur. In other words, if the answer set contains $occ(a_1, 1)$, …, $occ(a_m, m)$ then the plan is $a_1, \ldots, a_m$. For $\pi_{h,w}(\mathscr{P})$, the reconstruction process is not that simple because each answer set for $\pi_{h,w}(\mathscr{P})$ represents a conditional plan which may contain conditionals in the form $br(l, t, p, p_1)$. The following procedure describes how to extract such a plan from an answer set.

Let $\mathscr{P} = (\mathscr{D}, \mathscr{I}, \mathscr{G})$ be a planning problem instance and S be an answer set for $\pi_{h,w}(\mathscr{P})$. For any pair of integers, $1 \leqslant i \leqslant h+1, 1 \leqslant k \leqslant w$, we define $p_i^k(S)$ as follows:

$$p_i^k(S) = \begin{cases} [] & \text{if } i = h+1 \text{ or } occ(a,i,k) \notin S \text{ for all } a \\ a; p_{i+1}^k(S) & \text{if } occ(a,i,k) \in S \text{ and} \\ & a \text{ is a non-sensing action} \\ a; cases(\{g_j \rightarrow p_{i+1}^{k_j}(S)\}_{j=1}^n) & \text{if } occ(a,i,k) \in S, \\ & a \text{ is a sensing action, and} \\ & br(g_j, i, k, k_j) \in S \text{ for } 1 \leqslant j \leqslant n \end{cases}$$

Intuitively, $p_i^k(S)$ is the conditional plan whose corresponding tree is rooted at node (i,k) on the grid $h \times w$. $p_1^1(S)$ is, therefore, a solution to $\mathscr{P}$. This is stated in Theorem 2 in the next subsection.

4.2 Soundness and completeness

Theorem 2
Let $(\mathscr{D}, \mathscr{I})$ be a consistent action theory, $\mathscr{P} = (\mathscr{D}, \mathscr{I}, \mathscr{G})$ be a planning problem instance and $h \geqslant 1$ and $w \geqslant 1$ be integers. If $\pi_{h,w}(\mathscr{P})$ returns an answer set S then $p_1^1(S)$ is a solution to $\mathscr{P}$.

Proof
see Appendix B. □

Theorem 2 shows the soundness of $\pi_{h,w}(\mathscr{P})$. We will now turn our attention to the completeness of $\pi_{h,w}(\mathscr{P})$. Observe that solutions generated by $\pi_{h,w}(\mathscr{P})$ are optimal in the following sense

1. actions do not occur once the goal is achieved or a possible next a-state does not exist; and
2. sensing actions do not occur if one of its sensed literals holds.

The first property holds because of rule (35) and the second property holds because of constraint (20). Since the definition of a conditional plan in general does not rule out *non-optimal* plans, obviously $\pi_{h,w}(\mathscr{P})$ will not generate all possible solutions to $\mathscr{P}$.

https://doi.org/10.1017/S1471068406002948 Published online by Cambridge University Press

For example, consider the planning problem instance $\mathscr{P}_1$ in Example 2. We have seen that plans p_2, p_3, and p_4 in Example 3 are all solutions to $\mathscr{P}_1$. However, p_3 and p_4 are not optimal because they do not satisfy the above two properties.

The above example shows that $\pi_{h,w}(\mathscr{P})$ is not complete w.r.t. the 0-approximation in the sense that no one-to-one correspondence between its answer sets and solutions to $\mathscr{P}$ exists. However, we will show next that it is complete in the sense that for each solution p to $\mathscr{P}$, there exist two integers h and w such that $\pi_{h,w}(\mathscr{P})$ will generate an answer set S whose corresponding plan, $p_1^1(S)$, can be obtained from p by applying the following transformation (called the `reduct` operation).

Definition 13 (*Reduct of a plan*)
Let $\mathscr{P} = (\mathscr{D}, \mathscr{I}, \mathscr{G})$ be a planning problem instance, p be a plan and δ be an a-state such that $\hat{\Phi}(p, \delta) \neq \bot$. A reduct of p with respect to δ, denoted by $\texttt{reduct}_\delta(p)$, is defined as follows.

1. if $p = []$ or $\delta \models \mathscr{G}$ then
$$\texttt{reduct}_\delta(p) = []$$
2. if $p = [a; q]$, where a is a non-sensing action and q is a plan, then
$$\texttt{reduct}_\delta(p) = \begin{cases} a; \texttt{reduct}_{\delta'}(q) & \text{if } \Phi(a, \delta) = \{\delta'\} \\ a & \text{otherwise} \end{cases}$$
3. if $p = [a; \textbf{cases}(\{g_j \rightarrow p_j\}_{j=1}^n)]$, where a is a sensing action that senses $g_1, \ldots, g_n$, then
$$\texttt{reduct}_\delta(p) = \begin{cases} \texttt{reduct}_\delta(p_k) & \text{if } g_k \text{ holds in } \delta \text{ for some } k \\ a; \textbf{cases}(\{g_j \rightarrow q_j\}_{j=1}^n) & \text{otherwise} \end{cases}$$
where
$$q_j = \begin{cases} [] & \text{if } Cl_{\mathscr{D}}(\delta \cup \{g_j\}) \text{ is inconsistent} \\ \texttt{reduct}_{Cl_{\mathscr{D}}(\delta \cup \{g_j\})}(p_j) & \text{otherwise} \end{cases}$$

Example 10
Consider the planning problem instance $\mathscr{P}_1$ in Example 2 and plans p_2, p_3, and p_4 in Example 3. Let $\delta = \{\neg open\}$. We will show that
$$\texttt{reduct}_\delta(p_3) = p_2 \tag{44}$$
and
$$\texttt{reduct}_\delta(p_4) = p_2 \tag{45}$$
Because *open*, *closed*, and *locked* do not hold in δ, we have
$$\texttt{reduct}_\delta(p_3) = check; \textbf{cases}(\{open \rightarrow q_1, closed \rightarrow q_2, locked \rightarrow q_3\})$$
where q_j's are defined as in Definition 13.

Let
$$\begin{aligned} \delta_1 &= Cl_{\mathscr{D}_1}(\delta \cup \{open\}) = \{open, \neg open, closed, \neg closed, locked, \neg locked\} \\ \delta_2 &= Cl_{\mathscr{D}_1}(\delta \cup \{closed\}) = \{\neg open, closed, \neg locked\} \\ \delta_3 &= Cl_{\mathscr{D}_1}(\delta \cup \{locked\}) = \{\neg open, \neg closed, locked\} \end{aligned}$$

https://doi.org/10.1017/S1471068406002948 Published online by Cambridge University Press

It is easy to see that $q_1 = []$ (because δ_1 is inconsistent) and $q_3 = []$ (because the sub-plan corresponding to the branch "locked" in p_3 is empty).

Let us compute q_2. We have

$$q_2 \quad = \quad \texttt{reduct}_{\delta_2}(flip\_lock; flip\_lock; flip\_lock)$$

Because δ_2 does not satisfy $\mathcal{G}$ and $\Phi(flip\_lock, \delta_2) = \{\delta_{2,1}\} \neq \emptyset$, where

$$\delta_{2,1} = \{\neg open, \neg closed, locked\},$$

we have

$$q_2 \quad = \quad flip\_lock; \texttt{reduct}_{\delta_{2,1}}(flip\_lock; flip\_lock)$$

As $\delta_{2,1}$ satisfies $\mathcal{G}$, we have $\texttt{reduct}_{\delta_{2,1}}(flip\_lock; flip\_lock) = []$. Hence, $q_2 = flip\_lock$. Accordingly, we have

$$\texttt{reduct}_{\delta}(p_3) = check; \textbf{cases}(\{open \rightarrow [], closed \rightarrow [flip\_lock], locked \rightarrow []\}) = p_2$$

That is, (44) holds.

We now show that (45) holds. It is easy to see that

$$\texttt{reduct}_{\delta}(p_4) = check; \textbf{cases}(\{open \rightarrow [], closed \rightarrow \texttt{reduct}_{\delta_2}(p_2), locked \rightarrow []\})$$

Because *closed* holds in δ_2, we have

$$\texttt{reduct}_{\delta_2}(p_2) = \texttt{reduct}_{\delta_2}(flip\_lock) = flip\_lock$$

Thus,

$$\texttt{reduct}_{\delta}(p_4) = check; \textbf{cases}(\{open \rightarrow [], closed \rightarrow flip\_lock, locked \rightarrow []\}) = p_2$$

As a result, we have (45) holds.

We have the following proposition.

Proposition 5
Let $\mathcal{P} = (\mathcal{D}, \mathcal{I}, \mathcal{G})$ be a planning problem instance and δ be its initial a-state. Then, for every solution p to $\mathcal{P}$, $\texttt{reduct}_{\delta}(p)$ is unique and also a solution to $\mathcal{P}$.

Proof
see Appendix B. □

The following theorem shows the completeness of our planner with respect to the 0-approximation semantics.

Theorem 3
Let $\mathcal{P} = (\mathcal{D}, \mathcal{I}, \mathcal{G})$ be a planning problem instance, and p be a solution to $\mathcal{P}$. Then, there exist two integers h and w such that $\pi_{h,w}(\mathcal{P})$ has an answer set S and $p_1^1(S) = \texttt{reduct}_{\delta}(p)$, where δ is the initial a-state of $(\mathcal{D}, \mathcal{I})$.

Proof
see Appendix B. □

https://doi.org/10.1017/S1471068406002948 Published online by Cambridge University Press

4.3 Special case: ASCP as a conformant planner

Since conformant planning deals only with incomplete information, it is easy to see that $\pi_{h,1}(\mathscr{P})$ can be used to generate conformant plans for $\mathscr{P}$.

Let S be an answer set for $\pi_{h,1}(\mathscr{P})$. Recall that we assume that each sensing action senses at least two literals. Hence, $w = 1$ implies S does not contain $occ(a,\ldots)$ where a is a sensing action because if otherwise rules (19) and (30) cannot be satisfied. Thus, $p_1^1(S)$ is a sequence of non-sensing actions. By Theorem 2, we know that $p_1^1(S)$ achieves the goal of $\mathscr{P}$ from every possible initial a-state of the domain, which implies that $p_1^1(S)$ is a conformant plan. In Section 5, we compare the performance of $\pi_{h,1}(\mathscr{P})$ against some of the state-of-the-art conformant planners.

4.4 Special case: ASCP as a reasoner

It is easy to see that with minor changes, ASCP can be used to compute the consequences of a plan. This can be done as follows. Given an action theory $(\mathscr{D},\mathscr{I})$, for any integers h, w, let $\pi_{h,w}(\mathscr{D},\mathscr{I})$ be the set of rules: $\pi_{h,w}(\mathscr{P}) \setminus \{(18)-(20),(24)-(26),(29)-(31),(35),(41)\}$. Intuitively, $\pi_{h,w}(\mathscr{D},\mathscr{I})$ is the program obtained from $\pi_{h,w}(\mathscr{P})$ by removing the rules for *(i)* generating the branches when sensing actions are executed; *(ii)* checking the satisfaction of the goal; *(iii)* representing the constraints on branches; and *(iv)* generating action occurrences. For a plan p, let T_p be the corresponding tree for p that is numbered according to the principles described in the previous section. We define $\epsilon(p)$ to be the following set of atoms:

$$\{occ(a,t,p) \mid \exists \text{ a node } x \text{ in } T_p \text{ labeled with action } a \text{ and numbered with } (t,p)\} \cup$$
$$\{br(g,t,p,p') \mid \exists \text{a link labeled with } g \text{ that connects the node numbered with } (t,p) \text{ to the node numbered with } (t+1,p') \text{ in } T_p\}.$$

It is easy to see that the program $\pi_{h,w}(\mathscr{D},\mathscr{I}) \cup \epsilon(p)$ has a unique answer set which corresponds to $\hat{\Phi}(p,s_0)$. This is detailed in the following proposition.

Proposition 6
Let $(\mathscr{D},\mathscr{I})$ be an action theory, p be a plan, ρ be a fluent formula, T_p be the plan tree for p with a given numbering, and h and w be the height and width of T_p respectively. Let

$$\Pi = \pi_{h,w}(\mathscr{D},\mathscr{I}) \cup \epsilon(p).$$

We have that

- Π has a unique answer set S;
- $\mathscr{D} \models_{\mathscr{I}}$ **knows** ρ **after** p if and only if
  - — there exists some j, $1 \leqslant j \leqslant w$, $\delta_{h+1,j}(S) \neq \bot$; and
  - — for every j, $1 \leqslant j \leqslant w$ and $\delta_{h+1,j}(S) \neq \bot$, ρ is known to be true in $\delta_{h+1,j}(S)$.
- $\mathscr{D} \models_{\mathscr{I}}$ **whether** ρ **after** p if and only if
  - — there exists some j, $1 \leqslant j \leqslant w$, $\delta_{h+1,j}(S) \neq \bot$; and
  - — for every j, $1 \leqslant j \leqslant w$ and $\delta_{h+1,j}(S) \neq \bot$, ρ is known in $\delta_{h+1,j}(S)$.

https://doi.org/10.1017/S1471068406002948 Published online by Cambridge University Press

where

$$\delta_{t,j}(S) = \begin{cases} \{l \mid holds(l,t,j) \in S\} & \text{if } used(t,j) \in S \text{ and} \\ & \{l \mid holds(l,t,j) \in S\} \text{ is consistent} \\ \bot & \text{otherwise} \end{cases}$$

Proof
The proof of this theorem is very similar to the proof of Theorem 2 so we omit it for brevity. □

5 Evaluation

In this section, we evaluate ASCP against other planners using planning benchmarks from the literature. We first briefly summarize the features of the systems that are used in our experiments. We then describe the benchmarks. Finally, we present the experimental results.

5.1 Planning systems

The planning systems that we compared with are the following:

- DLV$^{\mathrm{K}}$: DLV$^{\mathrm{K}}$ is a declarative, logic-based planning system built on top of the DLV system (`http://www.dbai.tuwien.ac.at/proj/dlv/`). The input language $\mathscr{K}$ is a logic-based planning language described in (Eiter et al. 2003). The version we used for testing is available at `http://www.dbai.tuwien.ac.at/proj/dlv/K/` . DLV$^{\mathrm{K}}$ is capable of generating both concurrent and conformant plans. It, however, does not support sensing actions and cannot generate conditional plans.
- CMBP (Conformant Model Based Planner) (Cimatti and Roveri 1999, 2000): CMBP is a conformant planner developed by Cimatti and Roveri. A planning domain in CMBP is represented as a finite state automaton. BDD (Binary Decision Diagram) techniques are employed to represent and search the automaton. CMBP allows nondeterministic domains with uncertainty in both the initial state and action effects. Nevertheless, it does not have the capability of generating concurrent and conditional plans. The input language to CMBP is $\mathscr{AR}$ described in Giunchiglia et al. (1997). The version used for testing was downloaded from `http://www.cs.washington.edu/research/jair/contents/v13.html` .
- KACMBP (Cimatti et al. 2004): Similarly to CMBP, KACMBP uses techniques from symbolic model checking to search in the belief space. However, in KACMBP, the search is guided by a heuristic function which is derived based on knowledge associated with a belief state.
  KACMBP is designated for sequential and conformant setting. It, however, does not support concurrent planning and conditional planning. The input language of KACMBP is SMV. The system was downloaded from `http://sra.itc.it/tools/mbp/AIJ04/`.

https://doi.org/10.1017/S1471068406002948 Published online by Cambridge University Press

Table 1. *Features of planning systems*

	ASCP	DLV^K	MBP	CMBP	SGP	POND	CFF	KACMBP
Input Language	$\mathscr{A}^c_K$	K	$\mathscr{AR}$	$\mathscr{AR}$	PDDL	PDDL	PDDL	SMV
Sequential planning	yes	yes	yes	yes	no	yes	yes	yes
Concurrent planning	no	yes	no	no	yes	no	no	no
Conformant planning	yes	yes	yes	yes	yes	yes	yes	yes
Conditional planning	yes	no	yes	no	yes	yes	no	no

- Conformant-FF (CFF) (Brafman and Hoffmann 2004): CFF[9], to our best knowledge, is one of the current fastest conformant planners in most of the benchmark domains in the literature. It extends the classical FF planner (Hoffmann and Nebel 2001) to deal with uncertainty in the initial state. The basic idea is to represent a belief state s just by the initial belief state (which is described as a CNF formula) together with the action sequence that leads to s. In addition, the reasoning is done by checking the satisfiability of CNF formulae.
  The input language of CFF is a subset of PDDL with a minor change that allows the users to specify the initial state as a CNF formula. Both sequential and and conformant planning are supported in CFF. However, it does not support concurrent and conditional planning.
- MBP (Bertoli et al. 2001): MBP is a previous version of CMBP. Unlike CMBP which only deals with conformant planning, MBP supports conditional planning as well. The version used for testing was downloaded from `http://sra.itc.it/tools/mbp/`.
- SGP (Sensory Graph Plan) (Weld et al. 1998; Anderson et al. 1998): SGP is a planner based on the planning graph algorithm proposed by Blum and Furst (95). SGP supports conditional effects, universal and existential quantification. It also handles uncertainty and sensing actions. SGP has the capability of generating both conformant and conditional plans, as well as concurrent plans. Nevertheless, static laws are not allowed in SGP. The input syntax is PDDL (Planning Domain Definition Language). The version used for testing is 1.0h (dated January 14th, 2000), written in Lisp, available at `http://www.cs.washington.edu/ai/sgp.html` .
- POND (Bryce et al. 2004): POND extends the planning graph algorithm (Blum and Furst 95) to deal with sensing actions. Conformant planning is also supported as a feature of POND. The input language is a subset of PDDL. POND was downloaded from `http://rakaposhi.eas.asu.edu/belief-search/`.

Table 1 summarizes the features of these planning systems.

[9] We would like to thank Jörg Hoffmann for providing us with an executable version of the system for testing.

https://doi.org/10.1017/S1471068406002948 Published online by Cambridge University Press

5.2 Benchmarks

To test the performance of the planners, we prepared two test suites for conformant and conditional planning, separately. In our preparation, we attempt to encode the planning problem instances given to the systems in a uniform way (in terms of the number of actions, fluents, and effects of actions). Due to the differences in the representation languages of these systems, there are situations in which the encoding of the problems might be different for each system.

5.2.1 Conformant planning

We tested the systems on the following domains[10]:

- **Bomb in the Toilet (BT):** This set of problems was introduced in (McDermott 1987): "It has been alarmed that there is a bomb in a lavatory. There are m suspicious packages, one of which contains the bomb. The bomb can be defused if we dunk the package that contains the bomb into a toilet." Experiments were made with $m = 2$, 4, 6, 8, and 10.
- **Bomb in the Toilet with Multiple Toilets (BMT):** This set of problems is similar to the **BT** problem but we have multiple toilets. There are five problems in this set, namely $BMT(2,2)$, $BMT(4,2)$, $BMT(6,2)$, $BMT(8,4)$, and $BMT(10,4)$, where the first parameter is the number of suspicious packages and the second parameter is the number of toilets.
- **Bomb in the Toilet with Clogging (BTC):** This set of problems is similar to BTs but we assume that dunking a package clogs the toilet and flushing the toilet unclogs it. We know that in the beginning, the toilet is unclogged. We did experiments with $m = 2, 4, 6, 8$, and 10, where m is the number of suspicious packages.
- **Bomb in the Toilet with Multiple Toilets and Clogging (BMTC):** This set of problems is similar to BTC but we have multiple toilets. We did experiments with five problems $BMTC(2,2)$, $BMTC(4,2)$, $BMTC(6,2)$, $BMTC(8,4)$, and $BMTC(10,4)$, where the first parameter is the number of suspicious packages and the second parameter is the number of toilets.
- **Bomb in the Toilet with Clogging and Uncertainty in Clogging (BTUC):** This set of problems is similar to BTC except that we do not know whether the toilet is clogged or not in the beginning.
- **Bomb in the Toilet with Multiple Toilets and Uncertainty in Clogging (BMTUC):** This set of problems is similar to BMTC except that we do not know whether or not each toilet is clogged in the beginning.
- **Ring:** This set of problems is from Cimatti et al. (2004). In this domain, one can move in a cyclic fashion (either forward or backward) around a n-room building to lock windows. Each room has a window and the window can be locked only if it is closed. Initially, the robot is in the first room and it does not know the state (open, closed or locked) of the windows. The goal

[10] The system is available at `http://www.cs.nmsu.edu/~tson/ASPlan/Sensing`.

https://doi.org/10.1017/S1471068406002948 Published online by Cambridge University Press

is to have all windows locked. A possible conformant plan is to perform a sequence of actions *forward, close, lock* repeatedly. In this domain, we tested with n =2,4,6,8, and 10.

- **Domino (DOM):** This domain is very simple. We have n dominos standing on a line in such a way that if one of them falls then the domino on its right also falls. There is a ball hanging close to the leftmost one. Touching the ball causes the first domino to fall. Initially, the states of dominos are unknown. The goal is to have the rightmost one to fall. The solution is obviously to touch the ball. In this domain, we tested with n =10,20,50,100, 1000, and 10000.

5.2.2 Conditional planning

The set of problems for testing includes:

- **Bomb in the Toilet with Sensing Actions (BTS):** This set of examples is taken from (Weld et al. 1998). They are variations of the BTC problem that allow sensing actions to be used to determine the existence of a bomb in a specific package. There are m packages and only one toilet. We can use one of the following methods to detect a bomb in a package: (1) use a metal detector (action *detect_metal*); (2) use a trained dog to sniff the bomb (action *sniff*); (3) use an x-ray machine (action *xray*); and, finally, (4) listen for the ticking of the bomb (action *listen_for_ticking*). This set of examples contains four subsets of problems, namely $BTS1(m)$, $BTS2(m)$, $BTS3(m)$, and $BTS4(m)$ respectively, where m is the number of suspicious packages. These subsets differ from each other in which ones of the above methods are allowed to use. The first subset allows only one sensing action (1); the second one allows sensing actions (1)-(2); and so on.
- **Medical Problem (MED):** This set of problems is from Weld et al. (1998). A patient is sick and we want to find the right medication for her. Using a wrong medication may be fatal. Performing a throat culture will return either *red*, *blue*, or *white*, which determines the group of illness the patient is infected with. Inspecting the color (that can be performed only after the throat culture is done) allows us to observe the color returned by a throat culture, depending on the illness of the patient. Analyzing a blood sample tells us whether or not the patient has a high white cell count. This can be done only after a blood sample is taken. In addition, we know that in the beginning, the patient is not dead but infected. In addition, none of the tests have been done. There are five problems in this set, namely, $MED1$, …, $MED5$. These problems are different from each other in how much we know about the illness of the patient in the beginning.
- **Sick Domain (SICK):** This set of problems is similar to MED. A patient is sick and we need to find a proper medication for her. There are n kinds of illness that she may be infected with and each requiring a particular medication. Performing throat culture can return a particular color. Inspecting that color determine what kind of illness the patient has. Initially, we do not know the exact illness that the patient is infected with. The characteristic of this domain

https://doi.org/10.1017/S1471068406002948 Published online by Cambridge University Press

is that the length of the plan is fixed (only 3) but the width of the plan may be large, depending on the number of illnesses. We did experiments with five problems in the domain, namely, $SICK(2)$, $SICK(4)$, ..., $SICK(10)$. They differ from each other in the number of illnesses that the patient may have.

- **Ring (RINGS)**: This domain is a modification of the $RING$ domain. In this modified version, the agent can close a window only if it is open. It can lock a window only if it is closed. The agent can determine the status of a window by observing it (sensing action *observe_window*).
- **Domino (DOMS)**: This is a variant of the DOM domain in which some dominos may be glued to the table. Unlike the original version of the DOM domain, in this variant, when a domino falls, the next one falls only if it is not glued. The agent can do an action to unglue a glued domino. We introduce a new sensing action *observe_domino*(X) to determine whether a domino X is glued or not.

5.3 Performance

We ran our experiments on a 2.4 GHz CPU, 768MB RAM, DELL machine, running Slackware 10.0 operating system. We compared ASCP with DLV^K, CMBP, SGP, CFF and KACMBP on the conformant benchmarks and with SGP, POND, and MBP on the conditional benchmarks. Time limit was set to 30 minutes. The CMU Common Lisp version 19a was used to run SGP examples. We ran ASCP examples on both `cmodels` and `smodels`. By convention, in what follows, we will use ASCP^c and ASCP^s to refer to the planner ASCP when it was run on `cmodels` and `smodels` respectively. Sometimes, if the distinction between the two is not important, by ASCP we mean both.

The experimental results for conformant and conditional planning are shown in Tables 2 and 3 respectively. Times are in seconds. "TO/AB" indicates that the corresponding planner does not return a solution within the time limit or stopped abnormally due to some reasons, for example, out of memory or segmentation fault.

In conformant setting (Table 2), it is noticeable that ASCP^c behaves better than ASCP^s in all the conformant benchmark domains, especially in large problems. Furthermore, CFF and KACMBP are superior to all the other planners on most of the testing problems. Especially, both of them scale up to larger instances very well, compared with the others. Yet, it is interesting to observe that ASCP^c does not lose out a whole lot against these two planners in many problems. In the following, we will discuss the performance ASCP in comparison with CMBP, DLV^K, and SGP.

It can be seen that ASCP^c is competitive with CMBP and outperforms DLV^K and SGP in most of problems. Specifically, in the BT domain, ASCP^c took only 0.12 seconds to solve the last problem, while DLV^K, CMBP, and SGP took 11.37, 0.5 and 2.13 seconds respectively. ASCP^s however is slower than CMBP and SGP in this domain.

In the BMT domain, ASCP^s is the worst. ASCP^s took more than two minutes to solve the largest problem in this domain, while CMBP took only 0.53 seconds. ASCP^c, however, is competitive with CMBP and outperforms both DLV^K and SGP.

https://doi.org/10.1017/S1471068406002948 Published online by Cambridge University Press

Table 2. *Conformant planning performance*

Problem	Min. PL	ASCP cmodels	ASCP smodels	DLV^K	CMBP	SGP	CFF	KA-CMBP
BT(2)	2	0.06	0.03	0.01	0.03	0.04	0.02	0.12
BT(4)	4	0.04	0.06	0.03	0.03	0.27	0.04	0.12
BT(6)	6	0.05	0.12	0.18	0.04	0.42	0.09	0.1
BT(8)	8	0.10	0.33	1.47	0.10	1.04	0.10	0.11
BT(10)	10	0.12	2.54	11.37	0.50	2.13	0.13	0.11
BMT(2,2)	2	0.04	0.04	0.01	0.03	0.07	0.02	0.07
BMT(4,2)	4	0.05	0.09	0.03	0.04	0.28	0.03	0.12
BMT(6,2)	6	0.11	0.23	0.19	0.05	0.29	0.07	0.10
BMT(8,4)	8	0.41	4.70	1.70	0.11	3.14	0.09	0.11
BMT(10,4)	10	0.51	152.45	12.18	0.53	5.90	0.12	0.14
BTC(2)	2	0.04	0.04	0.01	0.03	0.44	0.05	0.12
BTC(4)	7	0.04	0.12	0.33	0.04	21.62	0.06	0.10
BTC(6)	11	0.06	0.33	TO	0.1	TO	0.07	0.11
BTC(8)	15	0.11	0.53	TO	0.79	TO	0.07	0.13
BTC(10)	19	0.12	468.04	TO	9.76	TO	0.13	0.14
BMTC(2,2)	2	0.06	0.06	0.01	0.03	0.18	0.05	0.12
BMTC(4,2)	6	0.10	0.19	0.17	0.05	2.03	0.04	0.09
BMTC(6,2)	10	0.14	0.63	20.02	0.24	TO	0.07	0.12
BMTC(8,4)	12	0.56	60.56	TO	TO	TO	0.10	0.12
BMTC(10,4)	16	1.44	TO	TO	TO	TO	0.13	0.17
BTUC(2)	4	0.05	0.04	0.02	0.02	0.59	0.03	0.09
BTUC(4)	8	0.04	0.11	0.94	0.04	TO	0.04	0.11
BTUC(6)	12	0.06	0.22	524.3	0.11	TO	0.06	0.11
BTUC(8)	16	0.11	4.7	TO	0.96	TO	0.08	0.12
BTUC(10)	20	0.12	TO	TO	11.58	TO	0.13	0.16
BMTUC(2,2)	4	0.06	0.07	0.03	0.03	16.11	0.06	0.11
BMTUC(4,2)	8	0.10	0.23	0.24	0.07	TO	0.09	0.14
BMTUC(6,2)	12	0.14	19.88	1368.28	0.43	TO	0.08	0.14
BMTUC(8,4)	16	0.56	TO	TO	TO	TO	0.13	0.18
BMTUC(10,4)	20	0.63	TO	TO	TO	TO	0.16	0.16
RING(2)	5	0.12	0.47	0.201	0.04	0.14	0.05	0.00
RING(4)	11	0.21	6.76	0.638	0.05	2.28	0.09	0.12
RING(6)	17	31.73	TO	TO	0.40	77.10	0.20	0.13
RING(8)	23	1246.58	TO	TO	832.73	TO	0.74	0.18
RING(10)	29	TO	TO	TO	TO	TO	2.46	0.18
DOM(10)	1	0.11	0.08	0.03	0.04	2.24	0.05	0.13
DOM(20)	1	0.14	0.07	0.24	0.05	33.4	0.29	0.14
DOM(50)	1	0.47	0.40	1368.28	0.06	1315.98	4.44	1.34
DOM(100)	1	1.70	1.64	TO	0.11	TO	TO	2.56
DOM(500)	1	31.28	32.52	TO	2.16	TO	TO	29.10
DOM(1000)	1	121.91	129.96	TO	9.83	TO	TO	TO

In the *BTC* domain, although ASCP^s is better than DLV^K and SGP, its performance is far from that of CMBP. The time for ASCP^s to solve the largest problem is nearly 8 minutes, while that for CMBP is just 9.76 seconds. Again, ASCP^c is the best. It took only 0.12 seconds to solve the same problem.

https://doi.org/10.1017/S1471068406002948 Published online by Cambridge University Press

Table 3. *Conditional planning performance*

Problem	Min. Plan	ASCP		SGP	POND	MBP
	Length & Width	cmodels	smodels			
BTS1(2)	2x2	0.166	0.088	0.11	0.188	0.047
BTS1(4)	4x4	0.808	1.697	0.22	0.189	0.048
BTS1(6)	6x6	5.959	83.245	2.44	0.233	0.055
BTS1(8)	8x8	25.284	TO	24.24	0.346	0.076
BTS1(10)	10x10	85.476	TO	TO	0.918	0.384
BTS2(2)	2x2	0.39	0.102	0.19	0.186	0.038
BTS2(4)	4x4	1.143	3.858	0.32	0.198	0.067
BTS2(6)	6x6	19.478	1515.288	3.23	0.253	2.163
BTS2(8)	8x8	245.902	TO	25.5	0.452	109.867
BTS2(10)	10x10	345.498	TO	TO	1.627	178.823
BTS3(2)	2x2	0.357	0.13	0.22	0.185	0.082
BTS3(4)	4x4	1.099	5.329	0.44	0.195	1.93
BTS3(6)	6x6	7.055	TO	3.89	0.258	147.76
BTS3(8)	8x8	56.246	TO	28.41	0.549	AB
BTS3(10)	10x10	248.171	TO	TO	2.675	AB
BTS4(2)	2x2	0.236	0.149	0.26	0.194	0.098
BTS4(4)	4x4	1.696	3.556	0.64	0.191	AB
BTS4(6)	6x6	13.966	149.723	4.92	0.264	AB
BTS4(8)	8x8	115.28	TO	30.34	0.708	AB
BTS4(10)	10x10	126.439	TO	TO	4.051	AB
MED(1)	1x1	1.444	1.434	0.09	0.187	0.048
MED(2)	5x5	35.989	9.981	0.59	0.193	0.047
MED(3)	5x5	42.791	9.752	1.39	0.2	0.049
MED(4)	5x5	39.501	10.118	7.18	0.205	0.049
MED(5)	5x5	35.963	9.909	44.64	AB	0.05
SICK(2)	3x2	0.234	0.121	0.21	0.189	0.045
SICK(4)	3x4	0.901	0.797	10.29	0.19	0.048
SICK(6)	3x6	5.394	3.9	TO	0.201	0.059
SICK(8)	3x8	17.18	14.025	TO	0.221	0.129
SICK(10)	3x10	82.179	43.709	TO	0.261	0.778
RINGS(1)	3x3	0.768	0.14	0.67	0.198	0.045
RINGS(2)	7x9	1386.299	TO	TO	0.206	0.057
RINGS(3)	11x27	TO	TO	TO	0.391	0.207
RINGS(4)	15x64	TO	TO	TO	3.054	3.168
DOMS(1)	3x1	0.117	0.203	0.11	0.08	0.043
DOMS(2)	5x4	0.306	0.325	48.82	0.183	0.048
DOMS(3)	7x8	3.646	53.91	TO	0.19	0.057
DOMS(4)	9x16	87.639	TO	TO	0.248	0.101
DOMS(5)	11x32	TO	TO	TO	0.687	0.486

The $BMTC$ domain turns out to be hard for DLV^K, CMBP, and SGP. None of them were able to solve the $BMTC(8,4)$ within the time limit. Although ASCP^s was able to solve this instance, it could not solve the last instance. ASCP^c on the contrary can solve these instances very quickly, less than two seconds for each problem.

In the $BTUC$ and $BMTUC$ domains, although not competitive with ASCP^c, CMBP outperforms both DLV^K and SGP. For example, CMBP took less than 12

https://doi.org/10.1017/S1471068406002948 Published online by Cambridge University Press

seconds to solve the largest instance in the $BTUC$ domain, while ASCP^s, DLV^K, and SGP indicated a timeout. ASCP^s is competitive with DLV^K and much better than SGP. Its performance is worse than CMBP in these domains however.

The $RING$ domain is really hard for the planners except CFF and KACMBP. CFF and KACMBP took just a few minutes to solve the largest problem; however, KACMBP seems to scale up better than CFF on this domain. None of the other planners could solve the last problem. Among the others, CMBP is the best, followed by ASCP^c. CMBP took around 14 minutes to solve $RING(8)$ while ASCP^c took more than 20 minutes. ASCP^s is outperformed by both DLV^K and SGP.

In the last domain, DOM, again, CMBP outperforms ASCP, DLV^K, and SGP. The solving time of ASCP for the last problem is around 2 minutes, while that for CMBP is just less than 10 seconds. DLV^K and SGP were able to solve the first three instances of this domain only. It is worth noting here that the not-very-good performance of CFF and KACMBP on this domain is because that this domain is in nature very rich in static causal laws, a feature that is not supported by CFF and KACMBP. Therefore, to encode the domain in CFF and KACMBP, we had to compile away static causal laws.

The performance of ASCP in the conditional benchmarks is not as good as in the conformant benchmarks, compared with other testing planners. As can be seen in Table 3, it was outperformed by both POND and MBP in the benchmarks, except in the last two problems of the $BTS3$ domain or in the last three of the $BTS4$, where MBP had a problem with segmentation fault or memory excess, or in $MED(5)$ problem where POND stopped abnormally. Both POND and MBP did very good at testing domains. POND took just a few seconds to solve each instance in the testing domains. ASCP is also not competitive with SGP in small instances of the first five domains ($BTS1$-MED). However, when scaling up to larger problems, ASCP^c seems to be better than SGP. In the last three domains ($SICK$, $RINGS$, and $DOMS$), SGP is outperformed by both ASCP^c and ASCP^s.

6 Conclusion and future work

In this paper, we define an approximation for action theories with static causal laws and sensing actions. We prove that the newly developed approximation is sound with respect to the possible world semantics and is deterministic when non-sensing actions are executed. We also show that the approximation reduces the complexity of the conditional planning problem.

We use the approximation to develop an answer set programming based conditional planner, called ASCP. ASCP differs from previously developed model-based planners for domains with incomplete initial state (Bonet and Geffner 2000; Cimatti and Roveri 1999; Eiter et al. 2003; Smith and Weld 1998), in that it is capable of dealing with sensing actions and generating both conditional and conformant plans. We prove the correctness of ASCP by showing that plans generated by ASCP are solutions of the encoded planning problem instances. Furthermore, we prove that ASCP will generate a solution to $\mathscr{P}$ if it has a solution with respect to the

https://doi.org/10.1017/S1471068406002948 Published online by Cambridge University Press

given approximation. We also discuss the use of ASCP in reasoning about effects of conditional plans.

We compare ASCP with several planners. These results provide evidence for the usefulness of answer set planning in dealing with sensing actions and incomplete information. Our experiments also show that there are situations in which ASCP does not work as well as other state-of-the-art planners. In the future, we would like to investigate methods such as the use of domain knowledge to speed up the planning process (Son et al. 2005a).

Acknowledgments

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 this paper and an extended abstract of this paper, which appeared in (Son et al. 2004), for their constructive comments and suggestions. The first two authors were partially supported by NSF grant EIA-0220590.

Appendix A: Proofs related to the 0-approximation

This appendix contains the proofs for the propositions and theorems given in the paper. As stated, we assume that the body of each static law (4) is not an empty set and $\mathscr{G} \neq \emptyset$ for every planning problem $(\mathscr{D}, \mathscr{I}, \mathscr{G})$.

We begin with a lemma about the operator $Cl_{\mathscr{D}}$ that will be used in these proofs. We need the following definition. Given a domain description $\mathscr{D}$, for a set of literals σ, let

$$\Gamma(\sigma) = \sigma \cup \{l \mid \exists \mathbf{if}(l, \varphi) \in \mathscr{D} \text{ such that } \varphi \subseteq \sigma\}.$$

Let $\Gamma^0(\sigma) = \Gamma(\sigma)$ and $\Gamma^{i+1}(\sigma) = \Gamma(\Gamma^i(\sigma))$ for $i \geqslant 0$. Since, by the definition of Γ, for any set of literals σ' we have $\sigma' \subseteq \Gamma(\sigma')$, the sequence $\langle \Gamma^i(\sigma) \rangle_{i=0}^{\infty}$ is monotonic with respect to the set inclusion operation. In addition, $\langle \Gamma^i(\sigma) \rangle_{i=0}^{\infty}$ is bounded by the set of fluent literals. Thus, there exists $\sigma^{\mathtt{limit}}$ such that $\sigma_{\mathscr{D}}^{\mathtt{limit}} = \bigcup_{i=0}^{\infty} \Gamma^i(\sigma)$. Furthermore, $\sigma_{\mathscr{D}}^{\mathtt{limit}}$ is unique and satisfies all static causal laws in $\mathscr{D}$.

Lemma 1
For any set of literals σ, we have $\sigma_{\mathscr{D}}^{\mathtt{limit}} = Cl_{\mathscr{D}}(\sigma)$.

Proof
By induction we can easily show that $\Gamma^i(\sigma) \subseteq Cl_{\mathscr{D}}(\sigma)$ for all $i \geqslant 0$. Hence, we have

$$\sigma_{\mathscr{D}}^{\mathtt{limit}} \subseteq Cl_{\mathscr{D}}(\sigma)$$

Furthermore, from the construction of $\Gamma^i(\sigma)$, it follows that $\sigma^{\mathtt{limit}}$ satisfies all static causal laws in $\mathscr{D}$. Because of the minimality property of $Cl_{\mathscr{D}}(\sigma)$, we have

$$Cl_{\mathscr{D}}(\sigma) \subseteq \sigma_{\mathscr{D}}^{\mathtt{limit}}$$

Accordingly, we have

$$\sigma_{\mathscr{D}}^{\mathtt{limit}} = Cl_{\mathscr{D}}(\sigma)$$

□

https://doi.org/10.1017/S1471068406002948 Published online by Cambridge University Press

The following corollary follows immediately from the above lemma.

Corollary 6.1
For two sets of literals $\sigma \subseteq \sigma'$, $Cl_{\mathscr{D}}(\sigma) \subseteq Cl_{\mathscr{D}}(\sigma')$.

For an action a and a state s, let $e(a,s) = Cl_{\mathscr{D}}(E(a,s))$. We have the following lemma:

Lemma 2
Let a be an action and s, s' be states. Then, we have

$$Cl_{\mathscr{D}}(E(a,s) \cup (s \cap s')) = Cl_{\mathscr{D}}(e(a,s) \cup (s \cap s'))$$

Proof
Let $\gamma = E(a,s) \cup (s \cap s')$ and $\gamma' = e(a,s) \cup (s \cap s')$. As $\gamma \subseteq \gamma'$, it follows from Corollary 6.1 that to prove this lemma, it suffices to prove that

$$Cl_{\mathscr{D}}(\gamma') \subseteq Cl_{\mathscr{D}}(\gamma)$$

It is easy to see that

$$\gamma' = Cl_{\mathscr{D}}(E(a,s)) \cup (s \cap s') \subseteq Cl_{\mathscr{D}}(E(a,s) \cup (s \cap s')) = Cl_{\mathscr{D}}(\gamma)$$

Therefore, by Corollary 6.1, we have

$$Cl_{\mathscr{D}}(\gamma') \subseteq Cl_{\mathscr{D}}(Cl_{\mathscr{D}}(\gamma)) = Cl_{\mathscr{D}}(\gamma)$$

Proof done. □

Proof of Proposition 1

Lemma 3
For every state $s' \in Res^{c}_{\mathscr{D}}(a,s)$, we have

$$s' \setminus (e(a,s) \cup (s \cap s')) \subseteq pc(a,\delta)$$

Proof
Let σ denote $e(a,s) \cup (s \cap s')$. By Corollary 6.1, since $e(a,\delta) \subseteq e(a,s) \subseteq \sigma$, we have

$$Cl_{\mathscr{D}}(e(a,\delta)) \subseteq Cl_{\mathscr{D}}(\sigma) = s' \tag{46}$$

We now show that, for every $i \geqslant 1$,

$$\Gamma^{i}(\sigma) \setminus \Gamma^{i-1}(\sigma) \subseteq pc^{i}(a,\delta) \tag{47}$$

by induction on i.

1. **Base case:** $i = 1$. Let l be a literal in $\Gamma^{1}(\sigma) \setminus \Gamma^{0}(\sigma)$. We need to prove that $l \in pc^{1}(a,\delta)$.
   By the definition of Γ, it follows that

$$l \notin \Gamma^{0}(\sigma) = \sigma \tag{48}$$

$$l \in \Gamma^{1}(\sigma) \subseteq s' \tag{49}$$

https://doi.org/10.1017/S1471068406002948 Published online by Cambridge University Press

and, in addition, there exists a static causal law

$$\mathbf{if}(l, \varphi)$$

in $\mathscr{D}$ such that

$$\varphi \subseteq \Gamma^0(\sigma) = \sigma \tag{50}$$

By (48), we have $l \notin (s \cap s')$. By (49), we have $l \in s'$. Accordingly, we have $l \notin s$. On the other hand, because $\delta \subseteq s$, we have

$$l \notin \delta \tag{51}$$

It follows from (50) that $\varphi \subseteq s'$ since $\sigma \subseteq s'$. Because of the completeness of s', we have $\neg\varphi \cap s' = \emptyset$. On the other hand, by (46), we have $Cl_{\mathscr{D}}(e(a, \delta)) \subseteq s'$. As a result, we have

$$\neg\varphi \cap Cl_{\mathscr{D}}(e(a, \delta)) = \emptyset \tag{52}$$

We now show that $\varphi \nsubseteq s$. Suppose otherwise, that is, $\varphi \subseteq s$. This implies that $l \in s$. By (49), it follows that $l \in (s \cap s') \subseteq \sigma$ and this is a contradiction to (48). Thus, $\varphi \nsubseteq s$.

On the other hand, we know that $\varphi \subseteq \sigma = e(a, s) \cup (s \cap s')$ and thus we have $\varphi \cap (e(a, s) \setminus s) \neq \emptyset$. In addition, it is easy to see that $e(a, s) \setminus s \subseteq e(a, s) \setminus \delta \subseteq pc^0(a, \delta)$. Therefore, we have

$$\varphi \cap pc^0(a, \delta) \neq \emptyset \tag{53}$$

From (51) – (53), and by the definition of $pc^1(a, \delta)$, we can conclude that $l \in pc^1(a, \delta)$. The base case is thus true.

2. **Inductive Step:** Assume that (47) is true for all $i \leqslant k$. We need to prove that it is true for $i = k + 1$. Let l be a literal in $\Gamma^{k+1}(\sigma) \setminus \Gamma^k(\sigma)$. We will show that $l \in pc^{k+1}(a, \delta)$.

   By the definition of Γ, there exists a static causal law

$$\mathbf{if}(l, \varphi)$$

   in $\mathscr{D}$ such that

$$\varphi \subseteq \Gamma^k(\sigma) \subseteq s' \tag{54}$$

   Because $\varphi \subset s'$, we have $\neg\varphi \cap s' = \emptyset$. In addition, by (46), $Cl_{\mathscr{D}}(e(a, \delta))$ is a subset of s'. A a result, we have

$$\neg\varphi \cap Cl_{\mathscr{D}}(e(a, \delta)) = \emptyset \tag{55}$$

   It is easy to see that $\varphi \nsubseteq \Gamma^{k-1}(\sigma)$ for if otherwise then, by the definition of Γ, l must be in $\Gamma^k(\sigma)$, which is impossible. In other words, there exists $l' \in \varphi$ such that $l' \notin \Gamma^{k-1}(\sigma)$ but $l' \in \Gamma^k(\sigma)$. By the inductive hypothesis, we have $l' \in pc^k(a, \delta)$, which implies that

$$\varphi \cap pc^k(a, \delta) \neq \emptyset \tag{56}$$

https://doi.org/10.1017/S1471068406002948 Published online by Cambridge University Press

Because $l \notin \Gamma^k(\sigma)$, we have $l \notin \sigma$. As a result, $l \notin (s \cap s')$. On the other hand, since $l \in \Gamma^{k+1}(\sigma) \subseteq s'$, it follows that $l \notin s$. Thus, we have

$$l \notin \delta \tag{57}$$

From (55) – (57), and by the definition of $pc^{k+1}(a, \delta)$, it follows that $l \in pc^{k+1}(a, \delta)$. So the inductive step is proven.

As a result, it is always the case that (47) holds. Hence, we have

$$\Gamma^i(\sigma) \setminus \sigma \subseteq \bigcup_{j=0}^{i}(pc^j(a, \delta)) = pc^i(a, \delta)$$

and thus,

$$\bigcup_{i=0}^{\infty}(\Gamma^i(\sigma) \setminus \sigma) \subseteq \bigcup_{i=0}^{\infty} pc^i(a, \delta)$$

Accordingly, by Lemma (1) and by the definition of $pc(a, \delta)$, we have

$$(s' \setminus \sigma) \subseteq pc(a, \delta).$$

The lemma is thus true. □

We now prove Proposition 1. Let

$$\gamma = e(a, \delta) \cup (\delta \setminus \neg pc(a, \delta)) \qquad \delta' = Cl_{\mathscr{D}}(\gamma)$$

Let s' be some state in $Res^c_{\mathscr{D}}(a, s)$. Such an s' exists because $\mathscr{D}$ is consistent. By Lemma 2 and by Definition 2, we have

$$s' = Cl_{\mathscr{D}}(\sigma) \tag{58}$$

where

$$\sigma = e(a, s) \cup (s \cap s')$$

To prove Proposition 1, it suffices to prove that $\delta' \subseteq s'$. But first of all, let us prove, by induction, the following

$$\Gamma^i(\gamma) \subseteq s' \tag{59}$$

for every integer $i \geqslant 0$.

1. **Base Case:** $i = 0$. Assume that $l \in \Gamma^0(\gamma) = \gamma$. We need to show that $l \in s'$. There are two possibilities for $l \in \gamma$.
   a) $l \in e(a, \delta)$. It is easy to see that $l \in s'$ because
   $$e(a, \delta) \subseteq e(a, s) \subseteq \sigma \subseteq Cl_{\mathscr{D}}(\sigma) = s'.$$
   b) $l \notin e(a, \delta)$, $l \in \delta$, *and* $\neg l \notin pc(a, \delta)$. Since $\delta \subseteq s$, we have $l \in s$. Because of the completeness of s, it follows that $\neg l \notin s$. Accordingly, we have
   $$\neg l \notin (s \cap s') \tag{60}$$
   On the other hand, because $\neg l \notin pc(a, \delta)$, $\neg l \notin s$, and $(e(a, s) \setminus s) \subseteq pc^0(a, \delta) \subseteq pc(a, \delta)$, we have
   $$\neg l \notin e(a, s) \tag{61}$$

https://doi.org/10.1017/S1471068406002948 Published online by Cambridge University Press

From (60) and (61), it follows that $\neg l \notin \sigma$. In addition, since $\neg l \notin pc(a, \delta)$, by Lemma 3, we have $\neg l \notin s' \setminus \sigma$. Accordingly, we have $\neg l \notin s'$. Because s' is complete, we can conclude that $l \in s'$.

2. **Inductive Step:** Assume that (59) is true for all $i \leqslant k$. We need to show that $\Gamma^{k+1}(\gamma) \subseteq s'$. Let l be a literal in $\Gamma^{k+1}(\gamma)$. By the definition of $\Gamma^{k+1}(\gamma)$, there are two possibilities for l:

   a) $l \in \Gamma^k(\gamma)$. Clearly, in this case, we have $l \in s'$.

   b) *there exists a static causal law*

   $$\mathbf{if}(l, \varphi)$$

   in $\mathscr{D}$ *such that* $\varphi \subseteq \Gamma^k(\gamma)$.

   By the inductive hypothesis, we have $\varphi \subseteq s'$. Hence, l must hold in s'.

   Therefore, in both cases, we have $l \in s'$. This implies that $\Gamma^{k+1}(\gamma) \subseteq s'$.

As a result, (59) always holds. By Lemma 1, we have

$$\delta' = \bigcup_{i=0}^{\infty} \Gamma^i(\gamma) \subseteq s'$$

Since s' is a state, δ' is consistent. Thus, by the definition of the *Res*-function, we have

$$Res_{\mathscr{D}}(a, \delta) = \{\delta'\}$$

Furthermore, $\delta' \subseteq s'$ for every $s' \in Res^c_{\mathscr{D}}(a, s)$.

The proposition is proven.

Proof of Proposition 2

Since δ is valid, there exists a state s such that $\delta \subseteq s$.

On the other hand, we assume that in every state of the world, exactly one literal in θ holds, there exists a literal $g \in \theta$ such that g holds in s and for all $g' \in \theta \setminus \{g\}$, g' does not hold in s.

Accordingly, we have $\delta \cup \{g\} \subseteq s$. By Corollary 6.1, we have $\delta' = Cl_{\mathscr{D}}(\delta \cup \{g\}) \subseteq Cl_{\mathscr{D}}(s) = s$. Hence, δ' is consistent. By the definition of the *Res*—function, we have $\delta' \in Res_{\mathscr{D}}(a, \delta)$. Since $\delta' \subseteq s$, δ' is a valid a-state.

The proposition is thus true.

Proof of Proposition 3

Let us prove this proposition by using structural induction on p.

1. $p = []$. Trivial.
2. $p = [a; q]$, *where* q *is a conditional plan and* a *is a non-sensing action.*
   Assume that Proposition 3 is true for q. We need to prove that it is also true for p.
   Suppose $\hat{\Phi}(p, \delta) \neq \perp$. Clearly we have $\Phi(a, \delta) \neq \perp$.

https://doi.org/10.1017/S1471068406002948 Published online by Cambridge University Press

Therefore, we have $\Phi(a,\delta) = Res_{\mathscr{D}}(a,\delta)$. On the other hand, since δ is a valid a-state, it follows from Proposition 1 that $Res_{\mathscr{D}}(a,\delta) = \{\delta'\}$ for some valid a-state δ'.

As a result, we have $\hat{\Phi}(q,\delta')$ contains at least one valid a-state. Hence, $\hat{\Phi}(p,\delta) \neq \perp$ contains at least one valid a-state.

3. $p = [a; \textbf{cases}(\{g_j \rightarrow p_j\}_{j=1}^{n})]$, *where a is a sensing action that senses* $g_1,\ldots,g_n$.
Assume that Proposition 3 is true for p_j's. We need to prove that it is also true for p.

Because $\hat{\Phi}(p,\delta) \neq \perp$, we have $\Phi(a,\delta) \neq \perp$. By the definition of the Φ-function, we have $\Phi(a,\delta) = Res_{\mathscr{D}}(a,\delta)$. As δ is valid, by Proposition 2, $Res_{\mathscr{D}}(a,\delta)$ contains at least one valid a-state δ'.

By the definition of the *Res*—function for sensing actions, we know that $\delta' = Cl_{\mathscr{D}}(\delta \cup \{g_k\})$ for some k. This implies that g_k holds in δ'.

By the inductive hypothesis, we have $\hat{\Phi}(p_k,\delta')$ contains at least one valid a-state.

By the definition of the $\hat{\Phi}$-function, we have $\hat{\Phi}(p_k,\delta') \subseteq \hat{\Phi}(p,\delta)$. Thus, $\Phi(p,\delta)$ contains at least one valid a-state.

Proof of Proposition 4

Let n denote the size of $\mathscr{D}$. Because of Lemma 1, we can conclude that for any set of literals σ, computing $Cl_{\mathscr{D}}(\sigma)$ can be done in polynomial time in n.

Observe that for a non-sensing action a and an a-state δ, computing $e(a,\delta)$ and $pc(a,\delta)$ can be done in polynomial time in n. Thus, computing $\Phi(a,\delta)$ can be done in polynomial time in n.

Likewise, computing $\Phi(a,\delta)$ for a sensing action a can also be done in polynomial time in n.

Hence, Proposition 4 holds.

Proof of Theorem 1

The proof is similar to the proof of Theorem 3 in Baral et al. (2000a) which states that the conditional planning problem with respect to the 0-approximation in Son and Baral (2001) is **NP**-complete. Membership follows from Corollary 2.1. Hardness follows from the fact that the approximation proposed in this paper coincides with the 0-approximation in Son and Baral (2001), i.e, the conditional planning problem considered in this paper coincides with the planning problem with limited-sensing in Baral et al. (2000a) which is **NP**-complete. By the restriction principle, we conclude that the problem considered in this paper is also **NP**-complete.

Appendix B: Proofs related to π

This section contain proofs related to the correctness of π. Before we present the proofs, let us introduce some notations that will be used throughout the rest of the appendix. Given a program Π, by $lit(\Pi)$ we mean the set of atoms in Π. If Z is a

https://doi.org/10.1017/S1471068406002948 Published online by Cambridge University Press

splitting set for Π and Σ is a set of atoms then by $b_Z(\Pi)$ and $e_Z(\Pi \setminus b_Z(\Pi), \Sigma)$, we mean the bottom part of Π w.r.t. Z and the evaluation of the top part w.r.t. (Z, Σ) (see Lifschitz and Turner (1994) for more information about these notions).

Lemma 4

1. Let Π be a logic program. Suppose Π can be divided into two disjoint subprograms Π_1 and Π_2, i.e., $\Pi = \Pi_1 \cup \Pi_2$ and $lit(\Pi_1) \cap lit(\Pi_2) = \emptyset$. Then S is an answer set for Π if and only if there exist two sets S_1 and S_2 of atoms such that $S = S_1 \cup S_2$ and S_1 and S_2 are answer sets for Π_1 and Π_2 respectively.
2. The result in Item 1 can be generalized to n disjoint subprograms, where n is an arbitrary integer.

Proof
The first item can easily proved by using the splitting set $Z = lit(\Pi_1)$. The second item immediately follows from this result. □

Proof of Theorem 2

Suppose we are given a planning problem instance $\mathscr{P} = (\mathscr{D}, \mathscr{I}, \mathscr{G})$ and $\pi_{h,w}(\mathscr{P})$, where $h \geqslant 1$ and $w \geqslant 1$ are some integers, returns an answer set S. The proof is primarily based on the splitting set and splitting sequence theorems described in (Lifschitz and Turner 1994). It is organized as follows. We first prove a lemma related to the closure of a set of literals (Lemma 5). Together with Lemma 4, this lemma is used to prove some properties of $\pi_{h,w}(\mathscr{P})$ (Lemmas 6, 7 & 8). Based on these results, we prove the correctness of $\pi_{h,w}(\mathscr{P})$ in implementing the Φ and $\hat{\Phi}$ functions (Lemma 9 & Lemma 10). Theorem 2 can be derived directly from Lemma 10.

Recall that we have made certain assumptions for action theories given to ASCP: *(a)* for every k-proposition **determines**(a, η), η contains at least two elements; and *(b)* for every static causal law **if**(f, ϕ), ϕ is not an empty set.

The following lemma shows a code fragment that correctly encodes the closure of a set of literals.

Lemma 5
Let i and k be two integers greater than 0, and x be a 3-ary predicate. For any set σ of literals, the following program

$$\begin{array}{ll} x(l,i,k) \leftarrow & (l \in \sigma) \\ x(l,i,k) \leftarrow x(\varphi,i,k) & (\mathbf{if}(l,\varphi) \in \mathscr{D}) \end{array}$$

has the unique answer set $\{x(l,i,k) \mid l \in Cl_{\mathscr{D}}(\sigma)\}$.

Proof
By the definition of a model of a positive program, it is easy to see that the above program has the unique answer set $\{x(l,i,k) \mid l \in \sigma_{\mathscr{D}}^{\mathtt{limit}}\} = \{x(l,i,k) \mid l \in Cl_{\mathscr{D}}(\sigma)\}$ (see Lemma 1). □

Before showing some lemmas about the properties of $\pi_{h,w}(\mathscr{P})$, let us introduce some notions and definitions that will be used throughout the rest of this section.

https://doi.org/10.1017/S1471068406002948 Published online by Cambridge University Press

We first define some sets of atoms which will frequently be used in the proofs of Lemmas 6 – 10 and Theorem 2. Then we divide the program $\pi_{h,w}(\mathscr{P})$ into small parts to simplify the proofs. In particular, $\pi_{h,w}(\mathscr{P})$ is divided into two programs $\pi^*_{h,w}(\mathscr{P})$ and $\pi^c_{h,w}(\mathscr{P})$. The former consists of normal logic program rules while the latter consists of constraints in $\pi_{h,w}(\mathscr{P})$. Then we use the splitting set theorem to remove from $\pi^*_{h,w}(\mathscr{P})$ auxiliary atoms such as $fluent(\ldots)$, $literal(\ldots)$, $time(\ldots)$, $path(\ldots)$, etc. The resulting program, denoted by π_0, consists of "main" atoms only. We then use the splitting sequence theorem to further split π_0 into a set of programs π_i's. Intuitively, each π_i corresponds to a "cut" of π_0 at time point i. Finally, each π_i is divided into disjoint subprograms π^k_i's, each of which, intuitively, is a "cut" of π_i at a specific path.

For $1 \leqslant i \leqslant h+1$ and $1 \leqslant k \leqslant w$, let $A_{i,k}$ be the set of all the atoms of the form $occ(a,i,k)$, $poss(a,i,k)$, $used(i,k)$, $goal(i,k)$, $holds(l,i,k)$, $br(g,i,k,k')$ $(k' \geqslant k)$, $e(l,i,k)$, $pc(l,i,k)$, i.e.,

$$\begin{aligned} A_{i,k} \quad = \quad & \{occ(a,i,k), poss(a,i,k) \mid a \in \mathbf{A}\} \cup \\ & \{holds(l,i,k), e(l,i,k), pc(l,i,k) \mid l \text{ is a literal}\} \cup \\ & \{br(g,i,k,k') \mid g \text{ is a sensed-literal}, k \leqslant k' \leqslant w\} \cup \\ & \{used(i,k), goal(i,k)\} \end{aligned} \tag{62}$$

and let

$$A_i = \bigcup_{k=1}^{w} A_{i,k}, \qquad A = \bigcup_{i=1}^{h+1} A_i \tag{63}$$

For a set of atoms $\Sigma \subseteq A$ and a set of predicate symbols X, by Σ^X we denote the set of atoms in Σ whose predicate symbols are in X and by $\delta_{i,k}(\Sigma)$, we mean $\{l \mid holds(l,i,k) \in \Sigma\}$.

Observe that $\pi_{h,w}(\mathscr{P})$ can be divided into two parts (1) $\pi^*_{h,w}(\mathscr{P})$ consisting of normal logic program rules, and (2) $\pi^c_{h,w}(\mathscr{P})$ consisting of constraints. Since S is an answer set for $\pi_{h,w}(\mathscr{P})$[11], S is also an answer set for $\pi^*_{h,w}(\mathscr{P})$ and does not violate any constraint in $\pi^c_{h,w}(\mathscr{P})$.

Let V be the set of atoms in $\pi_{h,w}(\mathscr{P})$ whose parameter list does not contain either the time or path variable. Specifically, V is the following set of atoms

$$\begin{aligned} & \{fluent(f), literal(f), literal(\neg f), contrary(f,\neg f), contrary(\neg f, f) \mid f \in \mathbf{F}\} \cup \\ & \{sensed(g) \mid \exists \textbf{determines}(a,\theta) \in \mathscr{D}.g \in \theta\} \cup \{action(a) \mid a \in \mathbf{A}\} \cup \\ & \{time(t) \mid t \in \{1..h\}\} \cup \{time1(t) \mid t \in \{1..h+1\}\} \cup \{path(p) \mid p \in \{1..w\}\} \end{aligned} \tag{64}$$

It is easy to see that V is a splitting set for $\pi^*_{h,w}(\mathscr{P})$. Furthermore, the bottom part $b_V(\pi^*_{h,w}(\mathscr{P}))$ is a positive program and has only one answer set $X_0 = V$. The partial evaluation of the top part of $\pi^*_{h,w}(\mathscr{P})$ with respect to X_0,

$$\pi_0 = e_V(\pi^*_{h,w}(\mathscr{P}) \setminus b_V(\pi^*_{h,w}(\mathscr{P})), X_0),$$

[11] Recall that at at the beginning of this section, we state that $\pi_{h,w}(\mathscr{P})$ returns S as an answer set.

https://doi.org/10.1017/S1471068406002948 Published online by Cambridge University Press

is the following set of rules (the condition for each rule follows that rule; and, by default t and p are in ranges $1 \dots h$ and $1 \dots w$ unless otherwise specified):

$$holds(l,1,1) \leftarrow \quad (\mathbf{initially}(l) \in \mathscr{I}) \tag{65}$$

$$poss(a,t,p) \leftarrow holds(\psi,t,p) \quad (\mathbf{executable}(a,\psi) \in \mathscr{D}) \tag{66}$$

$$e(l,t,p) \leftarrow occ(a,t,p), holds(\phi,t,p) \quad (\mathbf{causes}(a,l,\phi) \in \mathscr{D}) \tag{67}$$

$$pc(l,t,p) \leftarrow occ(a,t,p), not\ holds(l,t,p), not\ holds(\phi,t,p) \quad (\mathbf{causes}(a,l,\phi) \in \mathscr{D}) \tag{68}$$

$$br(g,t,p,p) \mid \dots$$

$$\mid br(g,t,p,w) \leftarrow occ(a,t,p) \quad (\mathbf{determines}(a,\theta) \in \mathscr{D}, g \in \theta) \tag{69}$$

$$pc(l,t,p) \leftarrow not\ holds(l,t,p), pc(l',t,p), not\ e(\neg\varphi,t,p) \quad (\mathbf{if}(l,\varphi) \in \mathscr{D}, l' \in \varphi) \tag{70}$$

$$e(l,t,p) \leftarrow e(\varphi,t,p) \quad (\mathbf{if}(l,\varphi) \in \mathscr{D}) \tag{71}$$

$$holds(l,t,p) \leftarrow holds(\varphi,t,p) \quad (\mathbf{if}(l,\varphi) \in \mathscr{D}, 1 \leqslant t \leqslant h+1) \tag{72}$$

$$goal(t,p) \leftarrow holds(\mathscr{G},t,p) \quad (1 \leqslant t \leqslant h+1) \tag{73}$$

$$goal(t,p) \leftarrow holds(f,t,p), holds(\neg f,t,p) \quad (1 \leqslant t \leqslant h+1) \tag{74}$$

$$holds(l,t{+}1,p) \leftarrow e(l,t,p) \tag{75}$$

$$holds(l,t{+}1,p) \leftarrow h(l,t,p), not\ pc(\neg l,t,p) \tag{76}$$

$$used(t{+}1,p) \leftarrow br(g,t,p_1,p) \quad (p_1 < p) \tag{77}$$

$$holds(g,t{+}1,p) \leftarrow br(g,t,p_1,p) \quad (p_1 \leqslant p) \tag{78}$$

$$holds(l,t{+}1,p) \leftarrow br(g,t,p_1,p), holds(l,t,p_1) \quad (p_1 < p) \tag{79}$$

$$occ(a_1,t,p) \mid \dots$$

$$occ(a_m,t,p) \leftarrow used(t,p), not\ goal(t,p) \tag{80}$$

$$used(1,1) \leftarrow \tag{81}$$

$$used(t{+}1,p) \leftarrow used(t,p) \tag{82}$$

https://doi.org/10.1017/S1471068406002948 Published online by Cambridge University Press

And $\pi^c_{h,w}(\mathscr{P})$ is the following collection of constraints

$$\leftarrow \quad occ(a,t,p), not\ br(\theta,t,p,p) \tag{83}$$
$$(\textbf{determines}(a,\theta) \in \mathscr{D})$$
$$\leftarrow \quad occ(a,t,p), br(g,t,p,p_1), br(g,t,p,p_2) \tag{84}$$
$$(\textbf{determines}(a,\theta) \in \mathscr{D}, g \in \theta, p \leqslant p_1 < p_2)$$
$$\leftarrow \quad occ(a,t,p), holds(g,t,p) \tag{85}$$
$$(\textbf{determines}(a,\theta) \in \mathscr{D}, g \in \theta)$$
$$\leftarrow \quad used(h{+}1,p), not\ goal(h{+}1,p) \tag{86}$$
$$\leftarrow \quad br(g_1,t,p_1,p), br(g_2,t,p_2,p) \tag{87}$$
$$(p_1 < p_2 < p)$$
$$\leftarrow \quad br(g_1,t,p_1,p), br(g_2,t,p_1,p) \tag{88}$$
$$(g_1 \neq g_2, p_1 \leqslant p)$$
$$\leftarrow \quad br(g,t,p_1,p), used(t,p) \tag{89}$$
$$(p_1 < p)$$
$$\leftarrow \quad used(t,p), not\ goal(t,p), occ(a_i,t,p), occ(a_j,t,p) \tag{90}$$
$$(1 \leqslant i < j \leqslant m)$$
$$\leftarrow \quad occ(a,t,p), not\ poss(a,t,p) \tag{91}$$

Note that choice rules of the form

$$1\{L_1,\ldots,L_n\}1 \leftarrow Body$$

have been translated into

$$L_1 \mid \ldots \mid L_n \leftarrow Body$$

and

$$\leftarrow Body, L_i, L_j \quad (1 \leqslant i < j \leqslant n)$$

By the splitting set theorem, there exists an answer set S_0 for π_0 such that $S = S_0 \cup X_0$. Let U_i be the set of atoms in π_0 whose time parameter is less than or equal to i, i.e.,

$$U_i = \bigcup_{j=1}^{i} A_j \tag{92}$$

It is easy to see that the sequence $\langle U_i \rangle_{i=1}^{h+1}$ is a splitting sequence for π_0. By the splitting sequence theorem, since S_0 is an answer set for π_0, there must be a sequence of sets of literals $\langle X_i \rangle_{i=1}^{h+1}$ such that $X_i \subseteq U_i \setminus U_{i-1}$, and

- $S_0 = \bigcup_{i=1}^{h+1} X_i$
- X_1 is an answer set for

$$\pi_1 = b_{U_1}(\pi_0) \tag{93}$$

- for every $1 < i \leqslant h+1$, X_i is an answer set for

$$\pi_i = e_{U_i}(b_{U_i}(\pi_0) \setminus b_{U_{i-1}}(\pi_0), \bigcup_{1 \leqslant t \leqslant i-1} X_t) \tag{94}$$

https://doi.org/10.1017/S1471068406002948 Published online by Cambridge University Press

Given a set of atoms Σ, consider rules of the following forms:

$$holds(l,1,1) \leftarrow \quad (\mathbf{initially}(l) \in \mathscr{I}) \tag{95}$$

$$poss(a,t,p) \leftarrow holds(\psi,t,p) \quad (\mathbf{executable}(a,\psi) \in \mathscr{D}) \tag{96}$$

$$e(l,t,p) \leftarrow occ(a,t,p), holds(\phi,t,p) \quad (\mathbf{causes}(a,l,\phi) \in \mathscr{D}) \tag{97}$$

$$pc(l,t,p) \leftarrow occ(a,t,p), not\ holds(l,t,p), not\ holds(\neg\phi,t,p) \quad (\mathbf{causes}(a,l,\phi) \in \mathscr{D}) \tag{98}$$

$$br(g,t,k,p) \mid \ldots$$

$$br(g,t,k,w) \leftarrow occ(a,t,p) \quad (\mathbf{determines}(a,\theta) \in \mathscr{D}, g \in \theta) \tag{99}$$

$$pc(l,t,p) \leftarrow not\ holds(l,t,p), pc(l',t,p), not\ e(\neg\varphi,t,p) \quad (\mathbf{if}(l,\varphi) \in \mathscr{D}, l' \in \varphi) \tag{100}$$

$$e(l,t,p) \leftarrow e(\varphi,t,p) \quad (\mathbf{if}(l,\varphi) \in \mathscr{D}) \tag{101}$$

$$holds(l,t,p) \leftarrow holds(\varphi,t,p) \quad (\mathbf{if}(l,\varphi) \in \mathscr{D}) \tag{102}$$

$$goal(t,p) \leftarrow holds(\mathscr{G},t,p) \tag{103}$$

$$goal(t,p) \leftarrow holds(f,t,p), holds(\neg f,t,p) \tag{104}$$

$$holds(l,t,p) \leftarrow \quad (e(l,t{-}1,p) \in \Sigma) \tag{105}$$

$$holds(l,t,p) \leftarrow \quad (holds(l,t{-}1,p) \in \Sigma, pc(\neg l,t{-}1,p) \notin \Sigma) \tag{106}$$

$$used(t,p) \leftarrow \quad (\exists\langle g,p'\rangle.p' < p \wedge br(g,t{-}1,p',p) \in \Sigma) \tag{107}$$

$$holds(g,t,p) \leftarrow \quad (\exists\langle g,p'\rangle.p' \leqslant p \wedge br(g,t{-}1,p',p) \in \Sigma) \tag{108}$$

$$holds(l,t,p) \leftarrow \quad \exists\langle g,p'\rangle.p' < p \wedge br(g,t{-}1,p',p) \in \Sigma \wedge holds(l,t{-}1,p') \in \Sigma) \tag{109}$$

$$occ(a_1,t,p) \mid \ldots \mid occ(a_m,t,p) \leftarrow used(t,p), not\ goal(t,p) \tag{110}$$

$$used(1,1) \leftarrow \tag{111}$$

$$used(t,p) \leftarrow \quad (used(t{-}1,p) \in \Sigma) \tag{112}$$

https://doi.org/10.1017/S1471068406002948 Published online by Cambridge University Press

Then for each $i \in \{1,\ldots,h+1\}$, π_i can be divided into w disjoint subprograms π_i^k, $1 \leqslant k \leqslant w$, where π_i^k is defined as follows

$$\pi_i^k = \begin{cases} \{(95)-(104),(110)-(111) \mid t=1,p=1\} & \text{if } i=1,k=1 \\ \{(96)-(104),(110) \mid t=1,p=k\} & \text{if } i=1,k>1 \\ \{(96)-(110),(112) \mid t=i,p=k,\Sigma=X_{i-1}\} & \text{if } 1<i\leqslant h \\ \{(102)-(109),(112) \mid t=h+1,p=k,\Sigma=X_h\} & \text{otherwise} \end{cases} \quad (113)$$

Let $X_{i,k}$ denote $X_i \cap A_{i,k}$. From Lemma 4, it follows that $X_{i,k}$ is an answer set for π_i^k. Hence, we have

$$\delta_{i,k}(S) = \delta_{i,k}(S_0) = \delta_{i,k}(X_i) = \delta_{i,k}(X_{i,k})$$

Due to this fact, from now on, we will use $\delta_{i,k}$ to refer to either $\delta_{i,k}(S)$, $\delta_{i,k}(S_0)$, $\delta_{i,k}(X_i)$, or $\delta_{i,k}(X_{i,k})$.

We have the following lemma

Lemma 6

For $1 \leqslant i \leqslant h+1$ and $1 \leqslant k \leqslant w$,

1. if $used(i,k) \notin S$ then S does not contain any atoms of the forms $holds(l,i,k)$, $e(l,i,k)$, $br(g,i,k,k)$;
2. if $used(h+1,k) \in S$ and $\delta_{h+1,k}$ is consistent then

$$\delta_{h+1,k} \models \mathscr{G}.$$

Proof

1. We will use induction on i to prove this item.
   a. **Base case:** $i = 1$. Let k be an integer such that $used(1,k) \notin S$. Clearly we have $k > 1$. On the other hand, it is easy to see that (by using the splitting set $Z = A_{1,k}^{\{holds,e,br,occ,goal,used\}}$) if $k > 1$ then S does not contain atoms of the forms $holds(l,1,k)$, $e(l,1,k)$, and $br(g,1,k,k)$. Thus, the base case is true.
   b. **Inductive step:** Assume that Item 1 is true for $i \leqslant j-1$, where $j > 1$. We will prove that it is also true for $i = j$. Let k be an integer such that $used(j,k) \notin S$.
   Clearly, to prove Item 1 we only need to prove that atoms of the forms $e(l,j,k)$, $holds(l,j,k)$, $br(g,j,k,k)$ do not belong to $X_{j,k}$. Consider the program π_j^k (see (113)). We know that $X_{j,k}$ is an answer set for π_j^k.
   Because of rule (112), we have $used(j-1,k) \notin X_{j-1}$. From (107), it follows that $br(g,j-1,k',k) \notin X_{j-1}$ for every pair $\langle g,k'\rangle$ such that $k' < k$. In addition, by the inductive hypothesis, we have that for any l and g, $e(l,j-1,k)$, $holds(l,j-1,k)$, and $br(g,j-1,k,k)$ are not in X_{j-1}. As a result, rules (105)-(109) do not exist in π_j^k. If we split π_j^k by the set $Z = A_{j,k}^{\{holds,e,br,occ,used,goal\}}$ then $b_Z(\pi_j^k)$ is the set of rules of the forms
      i. (97), (99), (101)–(103), (110) if $i \leqslant h$
      ii. (102)–(103) if $i = h+1$

https://doi.org/10.1017/S1471068406002948 Published online by Cambridge University Press

It is not difficult to show that this program has the empty set as its only answer set (recall that $\mathscr{G} \neq \emptyset$). From this, we can conclude the inductive step.

2. It is obvious because of the rules (73), (74) and the constraint (86).

□

Lemma 7
For $1 \leqslant i \leqslant h$ and $1 \leqslant k \leqslant w$, if $occ(a,i,k) \in S$ then a is executable in $\delta_{i,k}$ and there is no $b \neq a$ such that $occ(b,i,k) \in S$.

Proof
From constraint (91), it follows that $poss(a,i,k) \in S$. Notice only rules of the form (66) may have $poss(a,i,k)$ as its head. Hence, there must be a proposition (2) in $\mathscr{D}$ such that ψ holds in $\delta_{i,k}$. This means a is executable in $\delta_{i,k}$.

If there exists $b \neq a$ such that $occ(b,i,k) \in S$ then constraint (90) could not be satisfied. □

Lemma 8
for $1 \leqslant i \leqslant h$ and $1 \leqslant k \leqslant w$

1. if $occ(a,i,k) \in S$ and a is a non-sensing action then
   a. $e(l,i,k) \in S$ iff $l \in e(a,\delta_{i,k})$
   b. $pc(l,i,k) \in S$ iff $l \in pc(a,\delta_{i,k})$
   c. $\neg\exists\langle g,k'\rangle.br(g,i,k',k) \in S$
2. if $occ(a,i,k) \in S$ and a is a sensing action a with occurring in a k-proposition of the form (5) in $\mathscr{D}$ and $\theta = \{g_1,\ldots,g_n\}$ then there exist n distinct integers $k_1,\ldots,k_n$ greater than or equal to k such that
   a. $X_{i,k}^{\{br\}} = \{br(g_j,i,k,k_j) \mid j \in \{1,\ldots,n\}\}$
   b. g_j does not hold in $\delta_{i,k}$,
   c. if $k_j > k$ then S does not contain any atoms of the form $holds(l,i,k_j)$
3. if $occ(a,i,k) \notin S$ for every action a then
   a. $\forall l.pc(l,i,k) \notin S \wedge e(l,i,k) \notin S$
   b. $\forall\langle g,k'\rangle.br(g,i,k,k') \notin S$

Proof
Let us split π_i^k by the set $Z_1 = A_{i,k}^{\{used,goal,occ,holds,poss\}}$. By the splitting set theorem, $X_{i,k} = M \cup N$ where M is an answer set for $b_{Z_1}(\pi_i^k)$ and N is an answer set for $\Pi_1 = e_{Z_1}(\pi_i^k \setminus b_{Z_1}(\pi_i^k), M)$, which consists of the following rules

$$e(l,i,k) \leftarrow (occ(a,i,k) \in M, \mathbf{causes}(a,l,\phi) \in \mathscr{D}, holds(\phi,i,k) \subseteq M) \tag{114}$$

$$pc(l,i,k) \leftarrow (occ(a,i,k) \in M, holds(l,i,k) \notin M, \mathbf{causes}(a,l,\phi) \in \mathscr{D}, holds(\neg\phi,i,k) \cap M = \emptyset) \tag{115}$$

https://doi.org/10.1017/S1471068406002948 Published online by Cambridge University Press

$$
\begin{aligned}
& br(g,i,k,k) \mid \ldots \\
br(g,i,k,w) \quad & \leftarrow \qquad (occ(a,i,k) \in M, \mathbf{determines}(a,\theta) \in \mathscr{D}, g \in \theta) \qquad (116)\\
pc(l,i,k) \quad & \leftarrow \quad pc(l',i,k), not\ e(\neg\varphi,i,k) \qquad (117)\\
& \qquad (\mathbf{if}(l,\varphi) \in \mathscr{D}, holds(l,i,k) \notin M, l' \in \varphi)\\
e(l,i,k) \quad & \leftarrow \quad e(\varphi,i,k) \qquad (118)\\
& \qquad (\mathbf{if}(l,\varphi) \in \mathscr{D})
\end{aligned}
$$

From the splitting set theorem, it follows that $\delta_{i,k}(M) = \delta_{i,k}$

1. Assume that $occ(a,i,k) \in S$ and a is a non-sensing action. By Lemma 7, we know that there exists no sensing action[12] b such that $occ(b,i,k) \in S$. This means that rules of form (116) does not exist. Therefore, Π_1 can be rewritten to

$$
\begin{aligned}
e(l,i,k) \quad & \leftarrow \\
& \qquad (\mathbf{causes}(a,l,\phi) \in \mathscr{D}, holds(\phi,i,k) \subseteq M)\\
pc(l,i,k) \quad & \leftarrow \\
& \qquad (holds(l,i,k) \notin M, \mathbf{causes}(a,l,\phi) \in \mathscr{D},\\
& \qquad holds(\neg\phi,i,k) \cap M = \emptyset)\\
pc(l,i,k) \quad & \leftarrow \quad pc(l',i,k), not\ e(\neg\varphi,i,k)\\
& \qquad (\mathbf{if}(l,\varphi) \in \mathscr{D}, holds(l,i,k) \notin M, l' \in \varphi)\\
e(l,i,k) \quad & \leftarrow \quad e(\varphi,i,k)\\
& \qquad (\mathbf{if}(l,\varphi) \in \mathscr{D})
\end{aligned}
$$

   If we continue splitting the above program using $Z_2 = A_{i,k}^{\{e\}}$ then by Lemma 5, the bottom part has the only answer set

$$\{e(l,i,k) \mid l \in e(a,\delta_{i,k})\}$$

   and the evaluation of the top part has the only answer set

$$\{pc(l,i,k) \mid l \in pc(a,\delta_{i,k})\}$$

   Due to the fact that M does not contain any atoms of the form $e(l,i,k)$ or $pc(l,i,k)$, we therefore can conclude Items (a) and (b).
   We now show that $\neg\exists\langle g,k'\rangle.br(g,i,k',k) \in S$. Suppose otherwise, i.e., there exists g and k' such that $br(g,i,k',k) \in S$. Notice that only rule (80) with $t = i$ and $p = k$ has $occ(a,i,k)$ in its head. Hence, its body must be satisfied by S. That implies $used(i,k) \in S$.
   On the other hand, since only rules of the form (69) with $p = k'$ may have $br(g,i,k',k)$ in its head, there exists a sensing action b such that $occ(b,i,k') \in S$ and in addition, $k' \leqslant k$. As the sets of non-sensing actions and sensing actions

[12] Recall that the sets of non-sensing actions and sensing actions are disjoint from each other. Hence, a itself is not a sensing action.

https://doi.org/10.1017/S1471068406002948 Published online by Cambridge University Press

are disjoint from each other, we have $b \neq a$. From Lemma 7, it follows that $k' < k$.

Accordingly, we have $used(i,k) \in S, br(g,i,k',k) \in S$ and $k' < k$. Constraint (89) with $t = i$, $p = k$, and $p_1 = k'$ is thus violated. Thus, Item (c) holds.

2. Assume that $occ(a,i,k) \in S$ and a is a sensing action occurring in a k-proposition of the form (5) in $\mathscr{D}$ with$\theta = \{g_1, \ldots, g_n\}$.

In this case, since rules of the forms (114) and (115) do not exist, Π_1 is the following set of rules

$$
\begin{array}{rcl}
br(g_1,i,k,k) \mid \ldots & & \\
\quad br(g_1,i,k,w) & \leftarrow & \\
\qquad \ldots \quad \ldots & \quad \ldots & \\
br(g_n,i,k,k) \mid \ldots & & \\
\quad br(g_n,i,k,w) & \leftarrow & \\
\qquad pc(l,i,k) & \leftarrow & pc(l',i,k), not\ e(\neg\varphi,i,k) \\
 & & (\mathbf{if}(l,\varphi) \in \mathscr{D}, holds(l,i,k) \notin M, l' \in \varphi) \\
\qquad e(l,i,k) & \leftarrow & e(\varphi,i,k) \\
 & & (\mathbf{if}(l,\varphi) \in \mathscr{D})
\end{array}
$$

By further splitting the above program using the set $A_{i,k}^{\{e,pc\}}$, we will see that the bottom part has the empty set as its only answer set (recall that we are assuming that the body of each static law of the form (4) is not empty). Therefore, the answer set for the above program is also the answer set for the following program and vice versa.

$$
\begin{array}{rcl}
br(g_1,i,k,k) \mid \ldots & & \\
\quad br(g_1,i,k,w) & \leftarrow & \\
\qquad \ldots \quad \ldots & \quad \ldots & \\
br(g_n,i,k,k) \mid \ldots & & \\
\quad br(g_n,i,k,w) & \leftarrow &
\end{array}
$$

Thus, there exist n integers $k_1, \ldots, k_n$ greater than or equal to k such that

$$N = \bigcup_{j=1}^{n} \{br(g_j,i,k,k_j)\}$$

It is easy to see that $X_{i,k}^{\{br\}} = N^{\{br\}}$. In addition, by constraints of the form (88), k_j's must be distinct. Thus, Items (a) is true.

Item (b) can be drawn from constraints of the form (85).

Assume $k_j > k$. Because of constraints of the form (89), we have $used(i,k_j) \notin S$. From Lemma 6, it follows that S does not contain any atoms of the form $holds(l,i,k_j)$. Item (c) is thus true.

3. $occ(a,i,k) \notin S$ for every action a. In this case, Π_1 is the following set of rules

$$
\begin{array}{rcl}
pc(l,i,k) & \leftarrow & pc(l',i,k), not\ e(\neg\varphi,i,k) \\
 & & (\mathbf{if}(l,\varphi) \in \mathscr{D}, holds(l,i,k) \notin M, l' \in \varphi)
\end{array}
$$

https://doi.org/10.1017/S1471068406002948 Published online by Cambridge University Press

$$e(l,i,k) \leftarrow e(\varphi,i,k) \quad (\mathbf{if}(l,\varphi) \in \mathscr{D})$$

which has an empty set as its only answer set. Items (a)–(b) follow from this.

□

The following lemma shows that $\pi_{h,w}(\mathscr{P})$ correctly implements the transition function Φ.

Lemma 9
For $1 \leqslant i \leqslant h$ and $1 \leqslant k \leqslant w$

1. if there exists a non-sensing action a such that $occ(a,i,k) \in S$ then
$$\Phi(a,\delta_{i,k}) = \begin{cases} \emptyset & \text{if } \delta_{i+1,k} \text{ is inconsistent} \\ \{\delta_{i+1,k}\} & \text{otherwise} \end{cases};$$
2. if there exists a sensing action a occurring in a k-proposition of the form (5) in $\mathscr{D}$ with $\theta = \{g_1,\ldots,g_n\}$ such that $occ(a,i,k) \in S$ then there exist n integers $\{k_1,\ldots,k_n\}$ such that
$$\Phi(a,\delta_{i,k}) = \{\delta_{i+1,k_j} \mid 1 \leqslant j \leqslant n, \delta_{i+1,k_j} \text{ is consistent}\},$$
and for each j, g_j holds in δ_{i+1,k_j};
3. if $occ(a,i,k) \notin S$ for every action a,
$$\delta_{i+1,k} = \delta_{i,k}.$$

Proof

1. Assume that there exists a non-sensing action a such that $occ(a,i,k) \in X_i$.
Observe that $Z_1 = A_{i+1,k}^{\{holds\}}$ is a splitting set for π_{i+1}^k. Hence, by the splitting set theorem, $X_{i+1,k} = M \cup N$, where $M \subseteq Z_1$ is an answer set for $\Pi_1 = b_{Z_1}(\pi_{i+1}^k)$ and N is an answer set for $\Pi_2 = e_{Z_1}(\pi_{i+1}^k \setminus \Pi_1, M)$.
Notice that by Lemma 8, rules (108)–(109) for $t = i+1$, $p = k$ do not exist. Thus, Π_1 is the following set of rules:
$$\begin{aligned} holds(l,i{+}1,k) &\leftarrow holds(\varphi,i{+}1,k) \quad (\mathbf{if}(l,\varphi) \in \mathscr{D}) \\ holds(l,i{+}1,k) &\leftarrow \quad (e(l,i,k) \in X_i) \\ holds(l,i{+}1,k) &\leftarrow \quad (holds(l,i,k) \in X_i, pc(\neg l,i,k) \notin X_i) \end{aligned}$$
Also by Lemma 8, the conditions for the second and third rules can be written as $(l \in e(a,\delta_{i,k}))$ and $(l \in \delta_{i,k}, \neg l \notin pc(a,\delta_{i,k}))$ respectively. Thus, by Lemma 5, Π_1 has the unique answer set
$$M = \{holds(l,i{+}1,k) \mid l \in Cl_{\mathscr{D}}(a,\delta_{i,k})\}$$

https://doi.org/10.1017/S1471068406002948 Published online by Cambridge University Press

On the other hand, by Lemma 7, a is executable in $\delta_{i,k}$. From the definition of the $Res_{\mathscr{D}}$ and Φ functions, it follows that

$$\Phi(a, \delta_{i,k}) = \begin{cases} \emptyset & \text{if } \delta_{i+1,k} \text{ is inconsistent} \\ \{\delta_{i+1,k}\} & \text{otherwise} \end{cases}$$

2. Assume that there exists a sensing action a with a k-proposition of the form (5) and $\theta = \{g_1, \ldots, g_n\}$ such that $occ(a, i, k) \in S$.
By Lemma 8, for each $j \in \{1 \ldots n\}$, there exists $k_j \geqslant k$ such that $br(g_j, i, k, k_j) \in X_i$. It is easy to see that $Z_2 = A_{i+1,k_j}^{\{holds\}}$ is a splitting set for $\pi_{i+1}^{k_j}$. Considering cases $k_j = k$ and $k_j > k$ in turn and observe that $holds(l, i, k_j) \notin S$ if $k_j > k$, we will see that in both cases $b_{Z_2}(\pi_{i+1}^{k_j})$ is the following set of rules:

$$\begin{aligned} holds(l, i{+}1, k_j) \quad &\leftarrow \quad holds(\varphi, i{+}1, k_j) \\ &\qquad\quad (\mathbf{if}(l, \varphi) \in \mathscr{D}) \\ holds(l, i{+}1, k_j) \quad &\leftarrow \\ &\qquad\quad (holds(l, i, k) \in X_i) \\ holds(g_j, i{+}1, k_j) \quad &\leftarrow \end{aligned}$$

By Lemma 5, the only answer set for the above program is

$$M = \{holds(l, i+1, k_j) \mid l \in Cl_{\mathscr{D}}(\delta_{i,k} \cup \{g_j\})\}$$

On the other hand, by Lemma 7, a is executable in $\delta_{i,k}$ and by Lemma 8, g_j does not hold in $\delta_{i,k}$. Thus, according to the definition of the transition function, we have

$$\Phi(a, \delta_{i,k}) = \{Cl_{\mathscr{D}}(\delta_{i,k} \cup \{g_j\}) \mid 1 \leqslant j \leqslant n, Cl_{\mathscr{D}}(\delta_{i,k} \cup \{g_j\}) \text{ is consistent}\}$$

Hence, we have

$$\Phi(a, \delta_{i,k}) = \{\delta_{i+1,k_j}(M) \mid 1 \leqslant j \leqslant n, \delta_{i+1,k_j}(M) \text{ is consistent}\} =$$

$$\{\delta_{i+1,k_j} \mid 1 \leqslant j \leqslant n, \delta_{i+1,k_j} \text{ is consistent}\}$$

and obviously, g_j holds in δ_{i+1,k_j}.

3. Assume that $occ(a, i, k) \notin S$ for every action a.
Similar to the first case, Z_1 is a splitting set for π_{i+1}^{k}. $b_{Z_1}(\pi_{i+1}^{k})$ is the following set of rules:

$$\begin{aligned} holds(l, i{+}1, k) \quad &\leftarrow \quad holds(\varphi, i{+}1, k) \\ &\qquad\quad (\mathbf{if}(l, \varphi) \in \mathscr{D}) \\ holds(l, i{+}1, k) \quad &\leftarrow \\ &\qquad\quad (holds(l, i, k) \in X_i) \end{aligned}$$

Because that $\delta_{i,k}$ is an a-state (Lemma 6), by Lemma 5 the only answer set for this program is

$$M = \{holds(l, i+1, k) \mid l \in \delta_{i,k}\}$$

Thus, we have

$$\delta_{i+1,k} = \delta_{i+1,k}(M) = \delta_{i,k}$$

□

https://doi.org/10.1017/S1471068406002948 Published online by Cambridge University Press

The following lemma shows that $\pi_{h,w}(\mathscr{P})$ correctly implements the extended transition function.

Lemma 10
We have

1. $\delta_{1,1}$ is the initial a-state for $\mathscr{P}$.
2. For every pair of integers $1 \leqslant i \leqslant h{+}1$, $1 \leqslant k \leqslant w$, if $used(i,k) \in S$ then
   a) $p_i^k(S)$ is a conditional plan
   b) furthermore, if $\delta_{i,k}$ is consistent then for every $\delta \in \hat{\Phi}(p_i^k(S), \delta_{i,k})$, $\delta \models \mathscr{G}$.

Proof

1. $Z_1 = A_{1,1}^{\{holds\}}$ is a splitting set for π_1^1. The bottom part, $b_{Z_1}(\pi_1^1)$, consists of the following rules:

$$\begin{aligned} holds(l,1,1) \quad &\leftarrow \\ &\quad \{\textbf{initially}(l) \in \mathscr{I}\} \\ holds(l,1,1) \quad &\leftarrow \quad holds(\varphi,1,1) \\ &\quad \{\textbf{if}(l,\varphi) \in \mathscr{D}\} \end{aligned}$$

   By Lemma 5, the only answer set for the above program is

$$M = \{holds(l,1,1) \mid l \in \delta_1\}$$

   where δ_1 is the initial a-state of $\mathscr{P}$. Thus, $\delta_{1,1} = \delta_{1,1}(M)$ is the initial a-state of $\mathscr{P}$.

2. We now prove Item 2 by induction on parameter i.
   a. **Base case:** $i = h{+}1$. Let k be an arbitrary integer between 1 and w such that $used(i,k) \in S$. Clearly $p_i^k(S) = []$ is a conditional plan.
   Now suppose that $\delta_{i,k}$ is consistent. According to the definition of the extended transition function, we have

$$\hat{\Phi}(p_i^k(S), \delta_{i,k}) = \hat{\Phi}([], \delta_{i,k}) = \{\delta_{i,k}\}$$

   On the other hand, by Lemma 6, we have that $\delta_{i,k} \models \mathscr{G}$. Thus, Item 2 is true for $i = h{+}1$.
   b. **Inductive step:** Assume that Item 2 is true for all $h + 1 \geqslant i > t$. We will show that it is true for $i = t$. Let k be an integer between 1 and w such that $used(t,k) \in S$. Consider three possibilities:
      i. $occ(a,t,k) \in S$ for some non-sensing action a. By the definition of $p_t^k(S)$, we have $p_t^k(S) = [a; p_{t+1}^k(S)]$. In addition, by rule (82) we have $used(t{+}1,k) \in S$. Thus, according to the inductive hypothesis, $p_{t+1}^k(S)$ is a conditional plan. Accordingly, $p_t^k(S)$ is also a conditional plan.
      Now suppose that $\delta_{t,k}$ is consistent. Consider two cases
         – $\delta_{t+1,k}$ *is consistent*. We have

$$\hat{\Phi}(p_t^k(S), \delta_{t,k}) = \hat{\Phi}([a; p_{t+1}^k(S)], \delta_{t,k}) = \hat{\Phi}(p_{t+1}^k(S), \delta_{t+1,k})$$

         (by Lemma 9 and by the definition of the extended transition function).

https://doi.org/10.1017/S1471068406002948 Published online by Cambridge University Press

On the other hand, according to the inductive hypothesis, for every δ in $\hat{\Phi}(p^k_{t+1}(S), \delta_{t+1,k})$, $\delta \models \mathscr{G}$. Hence, the inductive step is proven.

– $\delta_{t+1,k}$ *is inconsistent*. By Lemma 9, we have $\hat{\Phi}(p^k_t(S), \delta_{t,k}) = \emptyset$. Thus, the inductive step is proven.

ii. $occ(a,t,k) \in S$ for some sensing action a with a k-proposition of the form (5) and $\theta = \{g_1, \ldots, g_n\}$. By Lemma 8 there exist exactly n integers $k_1, \ldots, k_n$ greater than k such that $br(g_j, t, k, k_j) \in S$ for $1 \leqslant j \leqslant n$. This implies that $used(t+1, k_j) \in S$ (see rules (77) and (82)). Thus, by the definition of $p^k_t(S)$, we have $p^k_t(S) = [a; \mathbf{cases}(\{g_j \rightarrow p^{k_j}_{i+1}(S)\}^n_{j=1})]$. On the other hand, we know by the inductive hypothesis that $p^{k_j}_{i+1}(S)$ is a conditional plan for $1 \leqslant j \leqslant n$. As a result, $p^k_t(S)$ is also a conditional plan.
Suppose $\delta_{i,k}$ is consistent. Let $J = \{j \mid \delta_{t+1,k_j} \text{ is consistent}\}$. By Lemma 9, we have

$$\Phi(a, \delta_{t,k}) = \{\delta_{t+1,k_j} \mid j \in J\}$$

and g_j holds in δ_{t+1,k_j} for every $1 \leqslant j \leqslant n$. Hence, by the definition of $\hat{\Phi}$, we have

$$\hat{\Phi}(p^k_t(S), \delta_{t,k}) = \bigcup_{j \in J} \hat{\Phi}(p^k_{t+1}(S), \delta_{t+1,k_j})$$

According to the inductive hypothesis, for every $\delta \in \hat{\Phi}(p^k_{t+1}(S), \delta_{t+1,k_j})$, where $j \in J$, we have $\delta \models \mathscr{G}$. This implies that for every $\delta \in \hat{\Phi}(p^k_t(S), \delta_{t,k})$, we have $\delta \models \mathscr{G}$.

iii. There is no action a such that $occ(a,t,k) \in S$. According to the definition of $p^k_t(S)$, $p^k_t(S) = []$. Hence, it is a conditional plan.
It is easy to see that $goal(t,k) \in S$, which means that either $\delta_{t,k}$ is inconsistent or $\delta_{t,k} \models \mathscr{G}$ (see rules (73), (74), and (80)). Now suppose that $\delta_{t,k}$ is consistent. This implies that $\delta_{t,k} \models \mathscr{G}$. We have

$$\hat{\Phi}(p^k_t(S), \delta_{t,k}) = \hat{\Phi}([], \delta_{t,k}) = \{\delta_{t,k}\}$$

Thus, the inductive step is proven.

□

Theorem 2 immediately follows from Lemma 10.

Proof of Proposition 5

First, we prove the following lemma.

Lemma 11

Let $\mathscr{P} = (\mathscr{D}, \mathscr{I}, \mathscr{G})$ be a planning problem instance, δ be an a-state and p be a plan. If $\hat{\Phi}(p, \delta) \models \mathscr{G}$ then $\hat{\Phi}(\mathtt{reduct}_\delta(p), \delta) \models \mathscr{G}$.

https://doi.org/10.1017/S1471068406002948 Published online by Cambridge University Press

Proof

Let us prove the lemma by structural induction on p.

1. $p = []$.
   The proof is trivial since $\texttt{reduct}_\delta(p) = p = []$.
2. Assume that $p = [a; q]$, *where* q *is a conditional plan and* a *is a non-sensing action and the lemma is true for* q.
   Suppose $\hat{\Phi}(p, \delta) \models \mathcal{G}$. We need to show that $\hat{\Phi}(\texttt{reduct}_\delta(p), \delta) \models \mathcal{G}$.
   If $\delta \models \mathcal{G}$ then
   $$\hat{\Phi}(\texttt{reduct}_\delta(p), \delta) = \hat{\Phi}([], \delta) = \{\delta\} \models \mathcal{G}$$
   Now consider the case that $\delta \not\models \mathcal{G}$.
   Clearly, we have $\Phi(a, \delta) \neq \bot$. Therefore, $\Phi(a, \delta) = \{\delta'\}$ for some δ'. Hence, by the definition of `reduct`, we have
   $$\texttt{reduct}_\delta(p) = a; \texttt{reduct}_{\delta'}(q)$$
   Thus,
   $$\hat{\Phi}(\texttt{reduct}_\delta(p), \delta) = \hat{\Phi}(\texttt{reduct}_{\delta'}(q), \delta')$$
   On the other hand, we have
   $$\hat{\Phi}(p, \delta) = \hat{\Phi}(q, \delta')$$
   Because $\hat{\Phi}(p, \delta) \models \mathcal{G}$, we have
   $$\hat{\Phi}(q, \delta') \models \mathcal{G}$$
   By inductive hypothesis, we have
   $$\hat{\Phi}(\texttt{reduct}_{\delta'}(q), \delta') \models \mathcal{G}$$
   Hence,
   $$\hat{\Phi}(\texttt{reduct}_\delta(p), \delta) \models \mathcal{G}$$
3. Assume that $p = [a; \textbf{cases}(\{g_j \rightarrow p_j\}_{j=1}^n)]$, *where* a *is a sensing action that senses* $g_1, \ldots, g_n$, *and the lemma for* p_j*'s.*
   Suppose $\hat{\Phi}(p, \delta) \models \mathcal{G}$. We need to show that $\hat{\Phi}(\texttt{reduct}_\delta(p), \delta) \models \mathcal{G}$.
   If $\delta \models \mathcal{G}$ then
   $$\hat{\Phi}(\texttt{reduct}_\delta(p), \delta) = \hat{\Phi}([], \delta) = \{\delta\} \models \mathcal{G}$$
   Now consider the case that $\delta \not\models \mathcal{G}$. There are two possibilities.
   a) *there exists* g_k *such that* $g_k \in \delta$. By the definition of `reduct`, we have
   $$\texttt{reduct}_\delta(p) = \texttt{reduct}_\delta(p_k)$$
   By the definition of the $\hat{\Phi}$-function, it is easy to see that
   $$\hat{\Phi}(p, \delta) = \hat{\Phi}(p_k, \delta)$$
   Since $\hat{\Phi}(p, \delta) \models \mathcal{G}$, we have $\hat{\Phi}(p_k, \delta) \models \mathcal{G}$. By the inductive hypothesis, we have
   $$\hat{\Phi}(\texttt{reduct}_\delta(p_k), \delta) \models \mathcal{G}$$

https://doi.org/10.1017/S1471068406002948 Published online by Cambridge University Press

Hence, we have

$$\hat{\Phi}(\texttt{reduct}_\delta(p), \delta) \models \mathscr{G}$$

a) *for every* $1 \leqslant j \leqslant n$, $g_j \notin \delta$. By the definition of `reduct`, we have

$$\texttt{reduct}_\delta(p) = a; \textbf{cases}(\{g_j \rightarrow q_j\}_{j=1}^n)$$

where

$$q_j = \begin{cases} [] & \text{if } Cl_{\mathscr{D}}(\delta \cup \{g_j\}) \text{ is inconsistent} \\ \texttt{reduct}_{Cl_{\mathscr{D}}(\delta \cup \{g_j\})}(p_j) & \text{otherwise} \end{cases}$$

For every $1 \leqslant j \leqslant n$, let $\delta_j = Cl_{\mathscr{D}}(\delta \cup \{g_j\})$. Let $J = \{j \mid \delta_j \text{ is consistent}\}$. It is easy to see that

$$\hat{\Phi}(p, \delta) = \bigcup_{j \in J} \Phi(p_j, \delta_j)$$

because g_j holds in δ_j but for every $k \neq j$, g_k does not hold in δ_j. Because $\hat{\Phi}(p, \delta) \models \mathscr{G}$, we have

$$\Phi(p_j, \delta_j) \models \mathscr{G}$$

for every $j \in J$.
On the other hand, we have

$$q_j = \begin{cases} [] & \text{if } j \notin J \\ \texttt{reduct}_{\delta_j}(p_j) & \text{otherwise} \end{cases}$$

Thus,

$$\hat{\Phi}(\texttt{reduct}_\delta(p), \delta) = \bigcup_{j \in J} \Phi(q_j, \delta_j) = \bigcup_{j \in J} \Phi(\texttt{reduct}_{\delta_j}(p_j), \delta_j)$$

By the inductive hypothesis, for every $j \in J$, as $\Phi(p_j, \delta_j) \models \mathscr{G}$, we have $\Phi(\texttt{reduct}_{\delta_j}(p_j), \delta_j) \models \mathscr{G}$. As a result, we have

$$\hat{\Phi}(\texttt{reduct}_\delta(p), \delta) \models \mathscr{G}$$

□

We now prove Proposition 5. Let p be a solution to $\mathscr{P}$. From the construction of `reduct`, it is easy to see that $\texttt{reduct}_\delta(p)$ is unique.

By Lemma 11, we have that $\hat{\Phi}(\texttt{reduct}_\delta(p), \delta) \models \mathscr{G}$. Thus, $\texttt{reduct}_\delta(p)$ is also a solution to $\mathscr{P}$.

So, we can conclude the proposition.

Proof of Theorem 3

The idea of the proof is as follows. Let q be $\texttt{reduct}_\delta(p)$, where δ is the initial a-state of $\mathscr{P}$, and let T_q be the labeled tree for q numbered according to the principles described in Section 3. Let h and w denote the height and width of T_q respectively. For $1 \leqslant i \leqslant h+1$, $1 \leqslant k \leqslant w$, we define $\delta_{i,k}$ to be the a-state at node (i,k)[13] of

[13] That is, the node numbered with (i,k) in T_q.

https://doi.org/10.1017/S1471068406002948 Published online by Cambridge University Press

T_q if such a node exists and $\perp$ otherwise. Based on T_q and $\delta_{i,k}$, we construct the set $Y_{i,k}$ of atoms that hold at node (i,k). Then we prove that the union of these sets, denoted by S'_0, is an answer set for π_0 (rules (65)-(82)) by showing that each $Y_{i,k}$ is an answer set for a part of π_0, denoted by π'^k_i. Furthermore, a set S' can be constructed from S'_0 in such a way that it is an answer set for $\pi^*_{h,w}(\mathscr{P})$. Moreover, S' does not violate any constraints in $\pi^c_{h,w}$ (rules (83)-(91)). As such, it is an answer set for $\pi_{h,w}(\mathscr{P})$. Moreover, $q = p^1_1(S')$.

Given the numbered tree T_q, by $\langle a,i,k\rangle$ we mean the node labeled with a and numbered with (i,k) in T_q; by $\langle g,i,k,k'\rangle \in T_q$ we mean the link, whose label is g, between the nodes (i,k) and $(i+1,k')$ in T_q.

For $1 \leqslant i \leqslant h+1$, $1 \leqslant k \leqslant w$, we define the a-state $\delta_{i,k}$ as follows.

i. if $i = 1$

$$\delta_{i,k} = \begin{cases} Cl_{\mathscr{D}}(\{l \mid \textbf{initially}(l) \in \mathscr{I}\}) & \text{if } k = 1 \\ \perp & \text{if } k > 1 \end{cases} \tag{119}$$

ii. if $i > 1$

$$\delta_{i,k} = \begin{cases} Cl_{\mathscr{D}}(e(a,\delta_{i-1,k}) \cup (\delta_{i-1,k} \setminus pc(a,\delta_{i-1,k}))) & \text{if } \langle a,i{-}1,k\rangle \in T_q \text{ for} \\ & \quad \text{a non-sensing action } a \\ Cl_{\mathscr{D}}(\delta_{i-1,k'} \cup \{g\}) & \text{if } \langle g,i{-}1,k',k\rangle \in T_q \\ \delta_{i-1,k} & \text{otherwise} \end{cases} \tag{120}$$

Note that given (i,k), there exists at most one action a such that $\langle a,i{-}1,k\rangle \in T_q$, and furthermore, at most one pair $\langle g,k'\rangle$ such that $\langle g,i{-}1,k',k\rangle \in T_q$. In addition, the conditions in Equation (120) do not overlap each other. Thus, $\delta_{i,k}$ is uniquely defined for $1 \leqslant i \leqslant h+1$ and $1 \leqslant k \leqslant w$. In what follows, the undefined situation $\perp$ can sometimes be thought of as $\emptyset$, depending the context in which it is used.

Let us construct the set $Y_{i,k}$ of atoms based on $\delta_{i,k}$ as follows.

1. $used(1,1) \in Y_{1,1}$
2. $holds(l,i,k) \in Y_{i,k}$ iif $l \in \delta_{i,k}$
3. $poss(a,i,k) \in Y_{i,k}$ iif there exists a proposition of the form (2) s.t. $\psi \subseteq \delta_{i,k}$
4. $occ(a,i,k) \in Y_{i,k}$ iif $\langle a,i,k\rangle \in T_q$
5. $br(g,i,k,k') \in Y_{i,k}$ iif $\langle g,i,k,k'\rangle \in T_q$ for some g,k'
6. $e(l,i,k) \in Y_{i,k}$ iif $\langle a,i,k\rangle \in T_q$ and $l \in e(a,\delta_{i,k})$ for some non-sensing action a
7. $pc(l,i,k) \in Y_{i,k}$ iif $\langle a,i,k\rangle \in T_q$ and $l \in pc(a,\delta_{i,k})$ for some non-sensing action a
8. For $i > 1$, $used(i,k) \in Y_{i,k}$ iif either
   (a) $used(i{-}1,k) \in Y_{i-1,k}$; or
   (b) there exists $\langle g,k'\rangle$ s.t. $\langle g,i{-}1,k',k\rangle \in Y_{i-1,k'}$
9. $goal(i,k) \in Y_{i,k}$ iff $\delta_{i,k} \models \mathscr{G}$ or $\delta_{i,k}$ is inconsistent
10. Nothing else in $Y_{i,k}$

Clearly, $Y_{i,k}$'s are uniquely defined. Furthermore, they are disjoint from each other. Let

$$Y_i = \bigcup_{k=1}^{w} Y_{i,k} \text{ and } S'_0 = \bigcup_{i=1}^{h+1} Y_i$$

https://doi.org/10.1017/S1471068406002948 Published online by Cambridge University Press

Lemma 12
For $1 \leqslant i \leqslant h$ and $1 \leqslant k \leqslant w$, let $M = Y_{i,k}^{\{holds,poss,goal,used,occ\}}$ and let Π be the following program:

$$
\begin{aligned}
e(l,i,k) \quad &\leftarrow \\
&\quad (occ(a,i,k) \in M, \mathbf{causes}(a,l,\phi) \in \mathscr{D}, holds(\phi,i,k) \subseteq M) \\
pc(l,i,k) \quad &\leftarrow \\
&\quad (occ(a,i,k) \in M, \mathbf{causes}(a,l,\phi) \in \mathscr{D}, \\
&\quad holds(l,i,k) \notin M, holds(\neg\phi,i,k) \cap M = \emptyset) \\
br(g,i,k,k) \mid \ldots & \\
br(g,i,k,w) \quad &\leftarrow \\
&\quad (occ(a,i,k) \in M, \mathbf{determines}(a,\theta) \in \mathscr{D}, g \in \theta) \\
pc(l,i,k) \quad &\leftarrow \quad pc(l',i,k), not\ e(\neg\varphi,i,k) \\
&\quad (\mathbf{if}(l,\varphi) \in \mathscr{D}, holds(l,i,k) \notin M, l' \in \varphi) \\
e(l,i,k) \quad &\leftarrow \quad e(\varphi,i,k) \\
&\quad (\mathbf{if}(l,\varphi) \in \mathscr{D})
\end{aligned}
$$

Then, $N = Y_{i,k}^{\{e,pc,br\}}$ is an answer set for Π.

Proof
Given (i,k), there are three cases that may happen at node (i,k).

- there exists a non-sensing action a such that $\langle a,i,k\rangle \in T_q$;
- there exists a sensing action a such that $\langle a,i,k\rangle \in T_q$;
- $\langle a,i,k\rangle \notin T_q$ for every action a

Let us consider each of those in turn.

1. *there exists a non-sensing action a such that* $\langle a,i,k\rangle \in T_q$.
   From the construction of $Y_{i,k}$, we know that $occ(a,i,k) \in M$ and there is no $b \neq a$ such that $occ(b,i,k) \in M$. Furthermore, due to the fact that N does not contain any atom of the form $holds(l,i,k)$, we have $holds(l,i,k) \in M$ iff $holds(l,i,k) \in Y_{i,k}$. That means $holds(l,i,k) \in M$ iff $l \in \delta_{i,k}$.
   Hence, Π can be rewritten to:

$$
\begin{aligned}
e(l,i,k) \quad &\leftarrow \\
&\quad (l \in e(a,\delta_{i,k})) \\
pc(l,i,k) \quad &\leftarrow \\
&\quad (l \in pc^0(a,\delta_{i,k})) \\
pc(l,i,k) \quad &\leftarrow \quad pc(l',i,k), not\ e(\neg\varphi,i,k) \\
&\quad (\mathbf{if}(l,\varphi) \in \mathscr{D}, l \notin \delta_{i,k}, l' \in \varphi) \\
e(l,i,k) \quad &\leftarrow \quad e(\varphi,i,k) \\
&\quad (\mathbf{if}(l,\varphi) \in \mathscr{D})
\end{aligned}
$$

https://doi.org/10.1017/S1471068406002948 Published online by Cambridge University Press

As have been seen in the proof of Theorem 2 (see the proof of Lemma 8, Item 1), the only answer set for this program is $\{e(l,i,k) \mid l \in e(a,\delta_{i,k})\} \cup \{pc(l,i,k) \mid l \in pc(a,\delta_{i,k})\} = N$.

2. *there exists a sensing action a such that* $\langle a,i,k\rangle \in T_q$.
We have $occ(a,i,k) \in M$ and there is no non-sensing action b such that $occ(b,i,k) \in M$. As a result, Π is

$$\begin{array}{rcl} br(g,i,k,k) \mid \ldots & & \\ \quad br(g,i,k,w) & \leftarrow & \\ & & (occ(a,i,k) \in M, \mathbf{determines}(a,\theta) \in \mathscr{D}, g \in \theta) \\ \quad pc(l,i,k) & \leftarrow & pc(l',i,k), not\ e(\neg\varphi,i,k) \\ & & (\mathbf{if}(l,\varphi) \in \mathscr{D}, holds(l,i,k) \notin M, l' \in \varphi) \\ \quad e(l,i,k) & \leftarrow & e(\varphi,i,k) \\ & & (\mathbf{if}(l,\varphi) \in \mathscr{D}) \end{array}$$

It is easy that an answer set for Π is also an answer set for

$$\begin{array}{rcl} br(g,i,k,k) \mid \ldots & & \\ \quad br(g,i,k,w) & \leftarrow & \\ & & (occ(a,i,k) \in M, \mathbf{determines}(a,\theta) \in \mathscr{D}, g \in \theta) \end{array}$$

and vice versa. On the other hand,

$$N = Y_{i,k}^{\{e,pc,br\}} = \{br(g,i,k,k') \mid \langle g,i,k,k'\rangle \in T_q\}$$

is an answer set for the latter program. As a result, N is also an answer set for Π.

3. $\langle a,i,k\rangle \notin T_q$ *for every action a.*
In this case, the first three rules of Π do not exist because $occ(a,i,k) \notin M$ for every a. Thus, Π consists of the last two rules only. It is easy to see that it has the empty set as its only answer set. On the other hand, from the construction of $Y_{i,k}$, we have $Y_{i,k}^{\{e,pc,br\}} = \emptyset$. Accordingly, $Y_{i,k}^{\{e,pc,br\}}$ is an answer set for Π.

The proof is done. □

Lemma 13
For $1 \leqslant i \leqslant h+1$, $1 \leqslant k \leqslant w$, $Y_{i,k}$ is an answer set for π'^k_i, where π'^k_i is defined in the same way as π^k_i except that we replace every occurrence of X in Equation (113) by Y.

Proof
Let us consider in turn two cases $i = 1$ and $i > 1$.

1. $i = 1$. It is easy to see that the only answer set for π'^k_1, where $k > 1$, is

$$Y_{1,k} = \{poss(a,1,k) \mid \mathbf{executable}(a,\emptyset) \in \mathscr{D}\}$$

by using the splitting set $A_{1,k}^{\{holds,occ,br,used,e,pc\}}$ (see (62) for the definition of $A_{i,k}$) and observe that the bottom part has the empty set as its only answer set and $Y_{1,k}$ is the only answer set for the evaluation of the top part.

https://doi.org/10.1017/S1471068406002948 Published online by Cambridge University Press

We now prove that $Y_{1,1}$ is an answer set for π'^1_1 which consists of the rules of the forms (95)-(104), (110)-(111) where $t = 1$ and $p = 1$. If we use the set $Z_1 = A^{\{holds,occ,poss,goal,used\}}_{1,1}$ to split π'^1_1 then $b_{Z_1}(\pi'^1_1)$ is

$$\{(95) - (96), (102) - (104), (110), (111) \mid t = 1, p = 1\}$$

From the definition of $Y_{1,1}$, we can easily show that $M = Y^{\{holds,occ,poss,goal,used\}}_{1,1}$ is an answer set for $b_{Z_1}(\pi'^1_1)$. Furthermore, we have

$$\delta_{1,1}(M) = \delta_{1,1}(Y_{1,1}) = \delta_{1,1}$$

The evaluation of the top part, $\Pi_1 = e_{Z_1}(\pi'^1_1 \setminus b_{Z_1}(\pi'^1_1), M)$, is the following set of rules

$$\begin{aligned}
e(l,1,1) \quad &\leftarrow \\
&\quad (occ(a,1,1) \in M, \mathbf{causes}(a,l,\phi) \in \mathscr{D}, \\
&\quad holds(\phi,1,1) \subseteq M) \\
pc(l,1,1) \quad &\leftarrow \\
&\quad (occ(a,1,1) \in M, \mathbf{causes}(a,l,\phi) \in \mathscr{D}, \\
&\quad holds(l,1,1) \notin M, holds(\neg\phi,1,1) \cap M = \emptyset) \\
br(g,1,1,k) \mid \ldots & \\
br(g,1,1,w) \quad &\leftarrow \\
&\quad (\mathbf{determines}(a,\theta) \in \mathscr{D}, g \in \theta, occ(a,1,1) \in M) \\
pc(l,1,1) \quad &\leftarrow \quad pc(l',1,1), not\ e(\neg\varphi,1,1) \\
&\quad (\mathbf{if}(l,\varphi) \in \mathscr{D}, l' \in \varphi, holds(l,1,1) \notin M) \\
e(l,1,1) \quad &\leftarrow \quad e(\varphi,1,1) \\
&\quad (\mathbf{if}(l,\varphi) \in \mathscr{D})
\end{aligned}$$

By Lemma 12, $N = Y^{\{e,pc,br\}}_{1,1}$ is an answer set for Π_1. As a result, $Y_{1,1} = M \cup N$ is an answer set for π'^1_1.

2. $1 < i \leqslant h + 1$.
Using the splitting set $Z_2 = A^{\{holds,occ,goal,used,poss\}}_{i,k}$ to split π'^k_i, we have that the bottom part $\Pi_2 = b_{Z_2}(\pi'^k_i)$ consists of rules of the forms
   - (96), (102)–(110), and (112) if $i \leqslant h$
   - (102)–(109), and (112) if $i = h + 1$

   We now prove that $M = Y^{\{holds,occ,goal,used,poss\}}_{i,k}$ is an answer set for Π_2. Let us further split Π_2 by the set $Z_3 = A^{\{holds\}}_{i,k}$. Then, the bottom part $b_{Z_3}(\Pi_2)$ consists of rules of the forms (102), (105)–(106), (108)–(109) only.
   Consider three cases
   a. *there exists a non-sensing action a such that* $occ(a, i{-}1, k) \in Y_{i-1}$.
      From the construction of $Y_{i,k}$'s, it is easy to see that there exists no $\langle g, k' \rangle$ such that $br(g, i{-}1, k', k) \in Y_{i-1}$. Thus, $b_{Z_3}(\Pi_2)$ contains rules of the forms (102), (105)–(106) only. On the other hand, we have

$$e(l, i{-}1, k) \in Y_{i-1} \text{ iff } l \in e(a, \delta_{i-1,k})$$

$$pc(\neg l, i{-}1, k) \notin Y_{i-1} \text{ iff } \neg l \notin pc(a, \delta_{i-1,k})$$

https://doi.org/10.1017/S1471068406002948 Published online by Cambridge University Press

Hence, $b_{Z_3}(\Pi_2)$ is the following collection of rules:

$$\begin{aligned} holds(l,i,k) &\leftarrow holds(\varphi,i,k) \\ &\quad (\mathbf{if}(l,\varphi) \in \mathscr{D}) \\ holds(l,i,k) &\leftarrow \\ &\quad (l \in e(a,\delta_{i-1,k})) \\ holds(l,i,k) &\leftarrow \\ &\quad (l \in \delta_{i-1,k}, \neg l \notin \delta_{i-1,k}) \end{aligned}$$

By Lemma 5, it has the only answer set

$$\{holds(l,i,k) \mid l \in Cl_{\mathscr{D}}(e(a,\delta_{i-1,k}) \cup (\delta_{i-1,k} \setminus pc(a,\delta_{i-1,k})))\} = Y_{i,k}^{\{holds\}}$$

b. $\exists\langle g,k'\rangle.br(g,i{-}1,k',k) \in Y_{i-1}$.
From the construction of $Y'_{i,k}s$, such $\langle g,k'\rangle$ is unique and in addition $k' \leqslant k$. Thus, $b_{Z_3}(\Pi_2)$ is

$$\begin{aligned} holds(l,i,k) &\leftarrow holds(\varphi,i,k) \\ &\quad (\mathbf{if}(l,\varphi) \in \mathscr{D}) \\ holds(l,i,k) &\leftarrow \\ &\quad ((l \in \delta_{i-1,k}) \vee (k' < k \wedge l \in \delta_{i-1,k'})) \\ holds(g,i,k) &\leftarrow \end{aligned}$$

or equivalently,

$$\begin{aligned} holds(l,i,k) &\leftarrow holds(\varphi,i,k) \\ &\quad (\mathbf{if}(l,\varphi) \in \mathscr{D}) \\ holds(l,i,k) &\leftarrow \\ &\quad (l \in \delta_{i-1,k'} \cup \{g\}) \end{aligned}$$

since if $k' < k$ then $\delta_{i-1,k} = \emptyset$. By Lemma 5, this program has the only answer set

$$\{holds(l,i,k) \mid l \in Cl_{\mathscr{D}}(\delta_{i-1,k'} \cup \{g\})\} = \{holds(l,i,k) \mid l \in \delta_{i,k}\}$$

Hence, $Y_{i,k}^{\{holds\}}$ is the only answer set for $b_{Z_3}(\Pi_2)$.

c. *$occ(a,i{-}1,k) \notin Y_{i-1}$ for every non-sensing action a and $\forall\langle g,k'\rangle.br(g,i{-}1,k',k) \notin Y_{i-1}$.*
From the construction of $Y_{i,k}$'s, it follows that $e(l,i{-}1,k) \notin Y_{i-1}$ and $pc(l,i{-}1,k) \notin Y_{i-1}$ for every l. Hence, $b_{Z_3}(\Pi_2)$ is the following set of rules

$$\begin{aligned} holds(l,i,k) &\leftarrow holds(\varphi,i,k) \\ &\quad (\mathbf{if}(l,\varphi) \in \mathscr{D}) \\ holds(l,i,k) &\leftarrow \\ &\quad (l \in \delta_{i-1,k}) \end{aligned}$$

https://doi.org/10.1017/S1471068406002948 Published online by Cambridge University Press

whose only answer set is

$$\{holds(l,i,k) \mid l \in \delta_{i-1,k}\} = \{holds(l,i,k) \mid l \in \delta_{i,k}\} = Y_{i,k}^{\{holds\}}$$

So, in all three cases, we have $Y_{i,k}^{\{holds\}}$ is an answer set for $b_{Z_3}(\Pi_2)$.
Hence, $\Pi_3 = e_{Z_3}(\Pi_2 \setminus b_{Z_3}(\Pi_2), Y_{i,k}^{\{holds\}})$ is the following set of rules:

$$\begin{aligned}
poss(a,i,k) &\leftarrow \\
&\quad (\mathbf{executable}(a,\psi) \in \mathscr{D}, \psi \subseteq \delta_{i,k}) \\
used(i,k) &\leftarrow \\
&\quad (\exists \langle g,k' \rangle . k' < k, br(g,i{-}1,k',k) \in Y_{i-1}) \\
goal(i,k) &\leftarrow \\
&\quad (\mathscr{G} \subseteq \delta_{i,k}) \\
goal(i,k) &\leftarrow \\
&\quad (\delta_{i,k} \text{ is inconsistent}) \\
occ(a_1,i,k) \mid \ldots & \\
\mid occ(a_m,i,k) &\leftarrow used(i,k), not\ goal(i,k) \\
used(i,k) &\leftarrow \\
&\quad (used(i{-}1,k) \in Y_{i-1})
\end{aligned}$$

It is easy to see that $Y_{i,k}^{\{poss,used,goal,occ\}}$ is an answer set for Π_3. Accordingly, we have $M = Y_{i,k}^{\{holds,poss,used,goal,occ\}}$ is an answer set for Π_2.
$\Pi_4 = e_{Z_2}(\pi_i'^k \setminus \Pi_2, M)$ is thus the following set of rules:

$$\begin{aligned}
e(l,i,k) &\leftarrow \\
&\quad (occ(a,i,k) \in M, \mathbf{causes}(a,l,\phi) \in \mathscr{D}, holds(\phi,i,k) \subseteq M) \\
pc(l,i,k) &\leftarrow \\
&\quad (occ(a,i,k) \in M, \mathbf{causes}(a,l,\phi) \in \mathscr{D}, \\
&\quad holds(l,i,k) \notin M, holds(\neg\phi,i,k) \cap M = \emptyset) \\
br(g,i,k,k) \mid \ldots & \\
br(g,i,k,w) &\leftarrow \\
&\quad (occ(a,i,k) \in M, \mathbf{determines}(a,\theta) \in \mathscr{D}, g \in \theta) \\
pc(l,i,k) &\leftarrow pc(l',i,k), not\ e(\neg\varphi,i,k) \\
&\quad (\mathbf{if}(l,\varphi) \in \mathscr{D}, holds(l,i,k) \notin M, l' \in \varphi) \\
e(l,i,k) &\leftarrow e(\varphi,i,k) \\
&\quad (\mathbf{if}(l,\varphi) \in \mathscr{D})
\end{aligned}$$

By Lemma 12, $N = Y_{i,k}^{\{e,pc,br\}}$ is an answer set for Π_4.
As a result, $Y_{i,k} = M \cup N$ is an answer set for $\pi_i'^k$.

□

https://doi.org/10.1017/S1471068406002948 Published online by Cambridge University Press

Lemma 14
We have

1. $S' = \bigcup_{i=1}^{h+1} Y_i \cup X_0$ is an answer set for $\pi_{h,w}(\mathscr{D})$, where $X_0 = V$ is defined in (64).
2. $p_1^1(S') = q$

Proof

1. Since $Y_{i,k}$ is an answer set for ${\pi'}_i^k$ and ${\pi'}_i^k$'s are disjoint from each other, we have Y_i is an answer set for ${\pi'}_i$, where ${\pi'}_i$ is defined in the same way as π_i except that every occurrence of X in Equations (93) and (94) is replaced with Y. From the splitting sequence theorem, it follows that $S'_0 = \bigcup_{i=1}^{h+1} Y_i$ is an answer set for π_0. Thus, S' is an answer set for $\pi^*_{h,w}(\mathscr{P})$.
   On the other hand, it is not difficult to show that S' satisfies all constraints in $\pi^*_{h,w}(\mathscr{P})$ based on the following observations.
   - If $occ(a,i,k) \in Y_{i,k}$ for some sensing action a which occurs in a k-proposition of the form (5) then there exists g in θ such that $br(g,i,k,k) \in Y_{i,k}$. Furthermore, for every $g' \in \theta$, g' does not in $\delta_{i,k}$. The latter property holds because that q does not contain an action that senses an already known-to-be-true literal.
   - If $used(h+1,k) \in Y_{h+1,k}$ then $\delta_{h+1,k} \models \mathscr{G}$.
   - $\delta_{i,k}$ is either $\perp$ or an a-state. This means that $Y_{i,k}^{\{holds\}}$ does not contain two atoms of the forms $holds(l,i,k)$ and $holds(l',i,k)$, where l and l' are contrary literals.
   - No two branches come to the same node (i,k).
   - If $used(i,k) \in Y_i$ then $br(g,i,k',k) \notin Y_i$ for any pair $\langle g,k'\rangle$, $k' \neq k$.
   - if $\langle a,i,k\rangle \in T_q$ then a must be executable in $\delta_{i,k}$.

   Accordingly, we have S is an answer set for $\pi_{h,w}(\mathscr{P})$.
2. Immediate from the construction of $Y_{i,k}$.

□

Theorem 3 follows directly from this lemma.

Appendix C: A sample encoding

This appendix contains the encoding of the planning problem $\mathscr{P}_1$ in Example 2. The first subsection describes the input planning problem. The next subsection presents the corresponding logic program $\pi_{h,w}(\mathscr{P}_1)$. The last two subsections are the outputs of `smodels` and `cmodels` when this logic program is run with the parameters $h = 2$ and $w = 3$.

Input Domain

```
% A possible plan is
%   check; cases(open-> [];closed->[flip_lock];locked->[])
% fluents
```

https://doi.org/10.1017/S1471068406002948 Published online by Cambridge University Press

```
fluent(open).
fluent(closed).
fluent(locked).

% actions
action(check).
action(push_up).
action(push_down).
action(flip_lock).

% executability conditions
executable(check,[]).
executable(push_up,[closed]).
executable(push_down,[open]).
executable(flip_lock,[neg(open)]).

% dynamic laws
causes(push_down,closed,[]).
causes(push_up,open,[]).
causes(flip_lock,locked,[closed]).
causes(flip_lock,closed,[locked]).

% knowledge laws
determines(check,[open,closed,locked]).

% static laws
oneof([open,closed,locked]).

% initial state
initially(neg(open)). % window is not open

% goal
goal(locked). % window is locked
```

Encoding

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Usage:
%     lparse -c h=<height> -c w=<width> | smodels
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
#domain fluent(F).
#domain literal(L;L1).
#domain sense(G;G1;G2).
#domain time(T).
#domain time1(T1).
#domain path(P;P1;P2).
#domain action(A).

% Input parameters
time(1..h).
time1(1..h+1).
path(1..w).
```

https://doi.org/10.1017/S1471068406002948 Published online by Cambridge University Press

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Action declarations
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
action(check).
action(push_up).
action(push_down).
action(flip_lock).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Fluent declarations
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
fluent(open).
fluent(closed).
fluent(locked).
sense(open).
sense(closed).
sense(locked).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% DOMAIN DEPENDENT RULES
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Initial situation
holds(neg(open),1,1).

% Executability conditions
poss(check,T,P).

poss(push_up,T,P) :-
        holds(closed,T,P).
poss(push_down,T,P) :-
        holds(open,T,P).
poss(flip_lock,T,P) :-
        holds(neg(open),T,P).

% Effects of non-sensing actions
e(closed,T+1,P) :-
        occ(push_down,T,P).
pc(closed,T+1,P) :-
        occ(push_down,T,P).
e(open,T+1,P) :-
        occ(push_up,T,P).
pc(open,T+1,P) :-
        occ(push_up,T,P).
e(locked,T+1,P) :-
        occ(flip_lock,T,P),
        holds(closed,T,P).
pc(locked,T+1,P) :-
        occ(flip_lock,T,P),
        not holds(neg(closed),T,P).
e(closed,T+1,P) :-
        occ(flip_lock,T,P),
        holds(locked,T,P).
```

https://doi.org/10.1017/S1471068406002948 Published online by Cambridge University Press

```
pc(closed,T+1,P) :-
        occ(flip_lock,T,P),
        not holds(neg(locked),T,P).

% Effects of sensing actions
:- occ(check,T,P),
        not br(open,T,P,P),
        not br(closed,T,P,P),
        not br(locked,T,P,P).
1{br(open,T,P,X):new_br(P,X)}1 :-
        occ(check,T,P).
1{br(closed,T,P,X):new_br(P,X)}1 :-
        occ(check,T,P).
1{br(locked,T,P,X):new_br(P,X)}1 :-
        occ(check,T,P).
:- occ(check,T,P),
        holds(open,T,P).
:- occ(check,T,P),
        holds(closed,T,P).
:- occ(check,T,P),
        holds(locked,T,P).


% Static laws
holds(neg(open),T1,P) :-
        holds(closed,T1,P).

e(neg(open),T+1,P) :-
        e(closed,T+1,P).

pc(neg(open),T+1,P) :-
        pc(closed,T+1,P),
        not holds(neg(open),T,P),
        not e(neg(closed),T+1,P).

holds(neg(open),T1,P) :-
        holds(locked,T1,P).

e(neg(open),T+1,P) :-
        e(locked,T+1,P).

pc(neg(open),T+1,P) :-
        pc(locked,T+1,P),
        not holds(neg(open),T,P),
        not e(neg(locked),T+1,P).

holds(open,T1,P) :-
        holds(neg(closed),T1,P),
        holds(neg(locked),T1,P).

e(open,T+1,P) :-
        e(neg(closed),T+1,P),
        e(neg(locked),T+1,P).
```

https://doi.org/10.1017/S1471068406002948 Published online by Cambridge University Press

```
pc(open,T+1,P) :-
        pc(neg(closed),T+1,P),
        not holds(open,T,P),
        not e(closed,T+1,P),
        not e(locked,T+1,P).
pc(open,T+1,P) :-
        pc(neg(locked),T+1,P),
        not holds(open,T,P),
        not e(closed,T+1,P),
        not e(locked,T+1,P).

holds(neg(closed),T1,P) :-
        holds(open,T1,P).

e(neg(closed),T+1,P) :-
        e(open,T+1,P).

pc(neg(closed),T+1,P) :-
        pc(open,T+1,P),
        not holds(neg(closed),T,P),
        not e(neg(open),T+1,P).

holds(neg(closed),T1,P) :-
        holds(locked,T1,P).

e(neg(closed),T+1,P) :-
        e(locked,T+1,P).

pc(neg(closed),T+1,P) :-
        pc(locked,T+1,P),
        not holds(neg(closed),T,P),
        not e(neg(locked),T+1,P).

holds(closed,T1,P) :-
        holds(neg(open),T1,P),
        holds(neg(locked),T1,P).

e(closed,T+1,P) :-
        e(neg(open),T+1,P),
        e(neg(locked),T+1,P).


pc(closed,T+1,P) :-
        pc(neg(open),T+1,P),
        not holds(closed,T,P),
        not e(open,T+1,P),
        not e(locked,T+1,P).
pc(closed,T+1,P) :-
        pc(neg(locked),T+1,P),
        not holds(closed,T,P),
        not e(open,T+1,P),
        not e(locked,T+1,P).
```

https://doi.org/10.1017/S1471068406002948 Published online by Cambridge University Press

```
holds(neg(locked),T1,P) :-
        holds(open,T1,P).

e(neg(locked),T+1,P) :-
        e(open,T+1,P).

pc(neg(locked),T+1,P) :-
        pc(open,T+1,P),
        not holds(neg(locked),T,P),
        not e(neg(open),T+1,P).

holds(neg(locked),T1,P) :-
        holds(closed,T1,P).

e(neg(locked),T+1,P) :-
        e(closed,T+1,P).

pc(neg(locked),T+1,P) :-
        pc(closed,T+1,P),
        not holds(neg(locked),T,P),
        not e(neg(closed),T+1,P).

holds(locked,T1,P) :-
        holds(neg(open),T1,P),
        holds(neg(closed),T1,P).

e(locked,T+1,P) :-
        e(neg(open),T+1,P),
        e(neg(closed),T+1,P).

pc(locked,T+1,P) :-
        pc(neg(open),T+1,P),
        not holds(locked,T,P),
        not e(open,T+1,P),
        not e(closed,T+1,P).
pc(locked,T+1,P) :-
        pc(neg(closed),T+1,P),
        not holds(locked,T,P),
        not e(open,T+1,P),
        not e(closed,T+1,P).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% GOAL REPRESENTATION
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

goal(T1,P) :-
        holds(locked,T1,P).

goal(T1,P) :-
        contrary(L,L1),
        holds(L,T1,P),
        holds(L1,T1,P).
```

https://doi.org/10.1017/S1471068406002948 Published online by Cambridge University Press

```
:- used(h+1,P),
        not goal(h+1,P).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% DOMAIN INDEPENDENT RULES
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Rules encoding the effects of non-sensing actions
holds(L,T+1,P) :-
        e(L,T+1,P).

holds(L,T+1,P) :-
        holds(L,T,P),
        contrary(L,L1),
        not pc(L1,T+1,P).

% Inertial rules for sensing actions
% Cannot branch to the same path
:- P1 < P2,
        P2 < P,
        br(G1,T,P1,P),
        br(G2,T,P2,P).

:- G1 != G2,
        P1 <= P,
        br(G1,T,P1,P),
        br(G2,T,P1,P).

:- P1 < P,
        br(G,T,P1,P),
        used(T,P).

used(T+1,P) :-
        P1 < P,
        br(G,T,P1,P).

holds(G,T+1,P) :-
        P1 <= P,
        br(G,T,P1,P).

holds(L,T+1,P) :-
        P1 < P,
        br(G,T,P1,P),
        holds(L,T,P1).

% Rules for generating action occurrences
1{occ(X,T,P):action(X)}1 :-
        used(T,P),
        not goal(T,P).

:- occ(A,T,P),
        not poss(A,T,P).

% Auxiliary Rules
```

https://doi.org/10.1017/S1471068406002948 Published online by Cambridge University Press

```
literal(F).
literal(neg(F)).

contrary(F,neg(F)).
contrary(neg(F),F).

new_br(P,P1) :-
        P <= P1.

used(1,1).
used(T+1,P) :-
        used(T,P).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% HIDE/SHOW ATOMS
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
hide.
show occ(A,T,P).
show br(G,T,P,P1).
```

Smodels Output

```
$ lparse -c h=2 -c w=3 examples/ex2.smo | smodels

smodels version 2.28. Reading...done
Answer: 1
Stable Model:
br(open,1,1,2) occ(check,1,1) br(closed,1,1,1)
br(locked,1,1,3) occ(flip_lock,2,1)
True
Duration: 0.020
Number of choice points: 2
Number of wrong choices: 0
Number of atoms: 313
Number of rules: 893
Number of picked atoms: 257
Number of forced atoms: 31
Number of truth assignments: 4052
Size of searchspace (removed): 12 (65)
```

Cmodels Output

```
$ lparse -c h=2 -c w=3 examples/ex2.smo | cmodels

cmodels
cmodels version 3.01 Reading...done
Program is not tight.
Calling SAT solver mChaff...
Answer: 1
Answer set: br(open,1,1,3) occ(check,1,1) br(closed,1,1,1)
br(locked,1,1,2) occ(flip_lock,2,1)
Number of Loop Formulas 6
```

https://doi.org/10.1017/S1471068406002948 Published online by Cambridge University Press

References

Anderson, C., Smith, D., and Weld, D. 1998. Conditional effects in Graphplan. In *Proceedings of the 4th International Conference on Artificial Intelligence Planning Systems (AIPS'98)*. AAAI Press, 44–53.

Anger, C., Konczak, K., and Linke, T. 2002. NoMoRe: Non-monotonic reasoning with logic programs. In *Proceedings of the 8th European Workshop on Logics in Artificial Intelligence 2002*, LNAI 2424. Springer Verlag.

Baral, C. 2003. *Knowledge Representation, reasoning, and declarative problem solving with Answer sets*. Cambridge University Press.

Baral, C., Kreinovich, V., and Trejo, R. 2000a. Computational complexity of planning and approximate planning in the presence of incompleteness. *Artificial Intelligence 122*, 241–267.

Baral, C., McIlraith, S., and Son, T. 2000b. Formulating diagnostic problem solving using an action language with narratives and sensing. In *Proceedings of the Seventh International Conference on Principles of Knowledge and Representation and Reasoning (KR'2000)*. 311–322.

Bertoli, P., Cimatti, A., and Roveri, M. 2001. Heuristic search + symbolic model checking = efficient conformant planning. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*. Morgan Kaufmann, 467–472.

Blum, A. and Furst, M. 95. Fast planning through planning graph analysis. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*. Morgan Kaufmann Publishers, San Francisco, CA, 1636–1642.

Bonet, B. and Geffner, H. 2000. Planning with incomplete information as heuristic search in belief space. In *Proceedings 6th International Conference on Artificial Intelligence Planning and Scheduling*, S. Chien, S. Kambhampati, and C. Knoblock, Eds. AAAI Press, 52–61.

Brafman, R. and Hoffmann, J. 2004. Conformant planning via heuristic forward search: A new approach. In *Proceedings of the 14th International Conference on Automated Planning and Scheduling (ICAPS-04)*, S. Koenig, S. Zilberstein, and J. Koehler, Eds. Morgan Kaufmann, Whistler, Canada, 355–364.

Bryce, D., Kambhampati, S., and Smith, D. 2004. Planning Graph Heuristics for Belief Space Search. Tech. rep., Arizona State University, Computer Science and Engineering. `http://www.public.asu.edu/~danbryce/papers/`.

Castellini, C., Giunchiglia, E., and Tacchella, A. 2003. Sat-based planning in complex domains: concurrency, constraints and nondeterminism. *Artificial Intelligence 147,* 1-2 (July), 85–117.

Cimatti, A. and Roveri, M. 1999. Conformant planning via model checking. In *European Conference on Planning*. Springer Verlag, LNAI 1809, 21–34.

Cimatti, A. and Roveri, M. 2000. Conformant Planning via Symbolic Model Checking. *Journal of Artificial Intelligence Research 13*, 305–338.

Cimatti, A., Roveri, M., and Bertoli, P. 2004. Conformant Planning via Symbolic Model Checking and Heuristic Search. *Artificial Intelligence Journal 159*, 127–206.

Citrigno, S., Eiter, T., Faber, W., Gottlob, G., Koch, C., Leone, N., Mateis, C., Pfeifer, G., and Scarcello, F. 1997. The dlv system: Model generator and application frontends. In *Proceedings of the 12th Workshop on Logic Programming*. 128–137.

Dimopoulos, Y., Nebel, B., and Koehler, J. 1997. Encoding planning problems in non-monotonic logic programs. In *Recent Advances in AI Planning, 4th European Conference on Planning, ECP'97, Toulouse, France, September 24-26, 1997, Proceedings*. Springer, 169–181.

Eiter, T., Faber, W., Leone, N., Pfeifer, G., and Polleres, A. 2003. A Logic Programming Approach to Knowledge State Planning, II: The DLV$^{\mathcal{K}}$ System. *Artificial Intelligence 144,* 1-2, 157–211.

https://doi.org/10.1017/S1471068406002948 Published online by Cambridge University Press

Gelfond, M. and Lifschitz, V. 1988. The stable model semantics for logic programming. In *Logic Programming: Proceedings of the Fifth International Conf. and Symp.*, R. Kowalski and K. Bowen, Eds. 1070–1080.

Gelfond, M. and Lifschitz, V. 1993. Representing actions and change by logic programs. *Journal of Logic Programming 17,* 2,3,4, 301–323.

Giunchiglia, E., Kartha, G., and Lifschitz, V. 1997. Representing action: indeterminacy and ramifications. *Artificial Intelligence 95*, 409–443.

Golden, K. 1998. Leap Before You Look: Information Gathering in the PUCCINI planner. In *Proceedings of the 4th International Conference on Artificial Intelligence Planning and Scheduling Systems*. 70–77.

Golden, K., Etzioni, O., and Weld, D. 1996a. Planning with execution and incomplete informations. Tech. rep., Dept of Computer Science, University of Washington, TR96-01-09. February.

Golden, K. and Weld, D. 1996b. Representing sensing actions: The middle ground revisited. In *Proceedings of the Fifth International Conference on Principles of Knowledge Representation and Reasoning (KR'96)*. Morgan Kaufmann Publishers, 174–185.

Hanks, S. and McDermott, D. 1987. Nonmonotonic Logic and Temporal Projection. *Artificial Intelligence 33*, 379–412.

Hoffmann, J. and Nebel, B. 2001. The FF Planning System: Fast Plan Generation Through Heuristic Search. *Journal of Artificial Intelligence Research 14*, 253–302.

Levesque, H. 1996. What is planning in the presence of sensing? In *Proceedings of the 14th Conference on Artificial Intelligence*. AAAI Press, 1139–1146.

Lierler, Y. and Maratea, M. 2004. Cmodels-2: SAT-based Answer Set Solver Enhanced to Non-tight Programs. In *Proceedings of the 7th International Conference on Logic Programming and NonMonotonic Reasoning Conference (LPNMR'04)*, V. Lifschitz and I. Niemelä, Eds. Vol. 2923. Springer Verlag, LNCS 2923, 346–350.

Lin, F. and Zhao, Y. 2002. ASSAT: Computing answer sets of a logic program by sat solvers. In *Proceedings of the National Conference on Artificial Intelligence (AAAI'02)*. AAAI Press. 112–117.

Lifschitz, V. 1999. Answer set planning. In *Proceedings of the 1999 international conference on Logic programming*. Massachusetts Institute of Technology, Cambridge, MA, USA, 23–37.

Lifschitz, V. 2002. Answer set programming and plan generation. *Artificial Intelligence 138,* 1–2, 39–54.

Lifschitz, V. and Turner, H. 1994. Splitting a logic program. In *Proceedings of the Eleventh International Conf. on Logic Programming*, P. Van Hentenryck, Ed. 23–38.

Lobo, J. 1998. COPLAS: a COnditional PLAnner with Sensing actions. Tech. Rep. FS-98-02, AAAI.

Lobo, J., Taylor, S., and Mendez, G. 1997. Adding knowledge to the action description language $\mathcal{A}$. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI'97)*, AAAI Press. 454–459.

Marek, V. and Truszczyński, M. 1999. Stable models and an alternative logic programming paradigm. In *The Logic Programming Paradigm: a 25-year Perspective*. 375–398.

McCain, N. and Turner, H. 1995. A causal theory of ramifications and qualifications. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*. Morgan Kaufmann Publishers, San Mateo, CA, 1978–1984.

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

Moore, R. 1985. A formal theory of knowledge and action. In *Formal theories of the commonsense world*, J. Hobbs and R. Moore, Eds. Ablex, Norwood, NJ.

https://doi.org/10.1017/S1471068406002948 Published online by Cambridge University Press

Niemelä, I. 1999. Logic programming with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence 25,* 3,4, 241–273.

Peot, M. and Smith, D. 1992. Conditional nonlinear planning. In *Proceedings of the First International Conference on AI Planning Systems*, J. Hendler, Ed. Morgan Kaufmann, College Park, Maryland, 189–197.

Pryor, L. and Collins, G. 1996. Planning for contingencies: A decision-based approach. *Journal of Artificial Intelligence Research 4*, 287–339.

Rintanen, J. 2000. Constructing conditional plans by a theorem prover. *Journal of Artificial Intelligence Research 10*, 323–352.

Scherl, R. and Levesque, H. 2003. Knowledge, action, and the frame problem. *Artificial Intelligence 144,* 1–2.

Simons, P., Niemelä, N., and Soininen, T. 2002. Extending and Implementing the Stable Model Semantics. *Artificial Intelligence 138,* 1–2, 181–234.

Smith, D. E. and Weld, D. S. 1998. Conformant Graphplan. In *Proceedings of the fifteenth national/tenth conference on Artificial intelligence (AAAI'98)*. AAAI Press, 889–896.

Son, T. and Baral, C. 2001. Formalizing sensing actions – a transition function based approach. *Artificial Intelligence 125,* 1–2 (January), 19–91.

Son, T., Baral, C., Nam, T., and McIlraith, S. 2005a. Domain-Dependent Knowledge in Answer Set Planning. *ACM Transactions on Computational Logic*. To Appear.

Son, T., Tu, P., and Baral, C. 2004. Planning with Sensing Actions and Incomplete Information using Logic Programming. In *Proceedings of the 7th International Conference on Logic Programming and NonMonotonic Reasoning Conference (LPNMR'04)*, V. Lifschitz and I. Niemelä, Eds. Vol. 2923. Springer Verlag, LNCS 2923, 261–274.

Son, T. C., Tu, P. H., Gelfond, M., and Morales, R. 2005b. Conformant Planning for Domains with Constraints – A New Approach. In *Proceedings of the the Twentieth National Conference on Artificial Intelligence*. 1211–1216.

Thiebaux, S., Hoffmann, J., and Nebel, B. 2003. In Defense of PDDL Axioms. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, G. Gottlob, Ed. Acapulco, Mexico, 961–966.

Thielscher, M. 2000a. The Fluent Calculus: A Specification Language for Robots with Sensors in Nondeterministic, Concurrent, and Ramifying Environments. Tech. Rep. CL-2000-01, Computational Logic Group, Department of Computer Science, Dresden University of Technology. Oct.

Thielscher, M. 2000b. Representing the knowledge of a robot. In *Proceedings of the Seventh International Conference on Principles of Knowledge and Representation and Reasoning (KR'2000)*. Morgan Kaufmann Publishers, 109–120.

Weld, D., Anderson, C., and Smith, D. 1998. Extending Graphplan to handle uncertainty and sensing actions. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence Conference (AAAI'98)*. AAAI Press, 897–904.

https://doi.org/10.1017/S1471068406002948 Published online by Cambridge University Press