Approximation of Action Theories and Its Application to Conformant Planning

Phan Huy Tu [a]

[a] *Microsoft Corporation, 1 Microsoft Way, Redmond, WA 98052*

Tran Cao Son [b]

[b] *Computer Science Department, New Mexico State University, Las Cruces, NM 88003, USA*

Michael Gelfond [c], A. Ricardo Morales [c]

[c] *Computer Science Department, Texas Tech University, Lubbock, TX 79409 USA*

Abstract

This paper describes our methodology for building conformant planners, which is based on recent advances in the theory of action and change and answer set programming. The development of a planner for a given dynamic domain starts with encoding the knowledge about fluents and actions of the domain as an action theory $\mathcal{D}$ of some action language. Our choice in this paper is $\mathcal{AL}$ - an action language with dynamic and static causal laws and executability conditions. An action theory $\mathcal{D}$ of $\mathcal{AL}$ defines a transition diagram $T(\mathcal{D})$ containing all the possible trajectories of the domain. A transition $\langle s, a, s' \rangle$ belongs to $T(\mathcal{D})$ iff the execution of the action a in the state s may move the domain to the state s'. The second step in the planner development consists in finding a deterministic transition diagram $T^{lp}(\mathcal{D})$ such that nodes of $T^{lp}(\mathcal{D})$ are partial states of $\mathcal{D}$, its arcs are labeled by actions, and a path in $T^{lp}(\mathcal{D})$ from an initial partial state δ^0 to a partial state satisfying the goal δ^f corresponds to a conformant plan for δ^0 and δ^f in $T(\mathcal{D})$. The transition diagram $T^{lp}(\mathcal{D})$ is called an 'approximation' of $T(\mathcal{D})$. We claim that a concise description of an approximation of $T(\mathcal{D})$ can often be given by a logic program $\pi(\mathcal{D})$ under the answer sets semantics. Moreover, complex initial situations and constraints on plans can be also expressed by logic programming rules and included in $\pi(\mathcal{D})$. If this is possible then the problem of finding a parallel or sequential conformant plan can be reduced to computing answer sets of $\pi(\mathcal{D})$. This can be done by general purpose answer set solvers. If plans are sequential and long then this method can be too time consuming. In this case, $\pi(\mathcal{D})$ is used as a specification for a procedural graph searching conformant planning algorithm. The paper illustrates this methodology by building several conformant planners which work for domains with complex relationship between the fluents. The efficiency of the planners is experimentally

Preprint submitted to Elsevier Science

evaluated on a number of new and old benchmarks. In addition we show that for a subclass of action theories of $\mathcal{AL}$ our planners are complete, i.e., if in $T^{lp}(\mathcal{D})$ we cannot get from δ^0 to a state satisfying the goal δ^f then there is no conformant plan for δ^0 and δ^f in $T(\mathcal{D})$.

Key words: Reasoning about action and change, Knowledge representation, Planning, Incomplete information, Answer set programming

---

1 Introduction

A conformant planner is a program that generates a sequence of actions, which achieves a goal from any possible initial state of the world, given the information about the initial state and the possible effects of actions. Such sequences are normally referred to as conformant plans. In this paper we describe our methodology for the design and implementation of conformant planners. The methodology is rooted in the ideas of declarative programming [42] and utilizes recent advances in answer set programming and the theory of action and change. This allows the designer to guarantee a substantially higher degree of trust in the planners' correctness, as well as a greater degree of elaboration tolerance [43].

The design of a declarative solution of a problem $\mathcal{P}$ normally involves the selection of a logical language capable of representing knowledge relevant to $\mathcal{P}$. We base our methodology on representing such knowledge in action languages - formal models of parts of natural language used for reasoning about actions and their effects. A theory in an action language (often called an action description) is used to succinctly describe the collection of all possible trajectories of a given dynamic domain. Usually this is done by defining the *transition diagram*, $T(\mathcal{D})$, of an action description $\mathcal{D}$. The states of $T(\mathcal{D})$ correspond to possible physical states of the domain represented by $\mathcal{D}$. Arcs of $T(\mathcal{D})$ are labeled by actions. A transition $\langle s, a, s' \rangle \in T(\mathcal{D})$ if the execution of the action a in the state s may move the domain to the state s'. In some action languages, actions are elementary (or atomic). In some others, an action a is viewed as a finite non-empty collection of elementary actions. Intuitively, execution of an action $a = \{e_1, \ldots, e_n\}$, where the e_i's are elementary actions, corresponds to the simultaneous execution of every $e_i \in a$.

There are by now a large number of action languages (see for instance

Email addresses: tuphan@microsoft.com (Phan Huy Tu), tson@cs.nmsu.edu (Tran Cao Son), mgelfond@cs.ttu.edu (Michael Gelfond), ricardo@cs.ttu.edu (A. Ricardo Morales).

[8,24,25,32,41,67]) capturing different aspects of dynamic domains. Our choice in this paper is $\mathcal{AL}$ [8] — an action language with *dynamic causal laws* describing direct effects of actions, *impossibility conditions* stating the conditions under which an action cannot be executed, and *static causal laws* (a.k.a. *state constraints*) describing static relations between fluents. For example the statement "*putting a block A on top of block B causes A to be on top of B*" can be viewed as a dynamic causal law describing the direct effect of action $put(A, B)$. The statement "*a block A cannot be put on B if there is a block located on A or on B*" represents an impossibility condition. The statement "*block A is above block C if A is on C or it is on B and B is above C*" is an example of a (recursive) static causal law. Note that static causal laws can cause actions to have *indirect effects*. Consider for instance the effects of executing the action $put(A, B)$ in a state in which both A and B are clear and B is located above some block C. The direct effects of this action (described by the dynamic causal law above) is $on(A, B)$. An indirect effect, $above(A, C)$, is obtained from our static causal law. The problem of determining such indirect effects, known as the *ramification problem*, remained open for a comparatively long time. In the last decade, several solutions to this problem have been proposed, for example [5,38,28,37,41,53,54,46,64]. One of these solutions [41] is incorporated in the semantics of $\mathcal{AL}$. The ability to represent causal laws makes $\mathcal{AL}$ a powerful modeling language. It was successfully used for instance to model the reactive control system of the space shuttle [4]. The system consists of fuel and oxidizer tanks, valves and other plumbing needed to provide propellant to the maneuvering jets of the shuttle. It also includes electronic circuitry; both to control the valves in the fuel lines and to prepare the jets to receive firing commands. Overall, the system is rather complex, in that it includes 12 tanks, 44 jets, 66 valves, 33 switches, and around 160 computer commands (computer-generated signals). The use of static causal laws (including recursive ones) was crucial for modeling the system and for the development of industrial size planning and diagnostic applications.

While static causal laws have been intensively studied by researchers interested in knowledge representation, they have rarely been considered by the mainstream planning community. Although the original specification of the Planning Domain Description Language (PDDL) – a language frequently used for the specification of planning problems by the planning community – includes axioms [1] (which correspond to non-recursive static causal laws in our terminology) [27], most of the planning domains investigated by this community, including those used for planning competitions [1,17,40] do not include axioms. This is partly due to the fact that the semantics for PDDL with axioms is not clearly specified, and partly to the (somewhat mistaken but apparently widespread) belief that static causal laws can always be replaced by dynamic

[1] In our view, static causal laws can be used to represent relationships between fluents and thus could be considered as axioms in PDDL.

causal laws. There is fortunately also an opposing view. For instance, in [63], the authors argue that the use of axioms not only increases the expressiveness and elegance of the problem representation but also improves the performance of planners. It is known that the complexity of the conformant planning problem is much higher than classical planning in deterministic domains (Σ_2^P vs. NP-complete) [6,68], and hence the question of efficiency becomes even more important.

An action description $\mathcal{D}$ of $\mathcal{AL}$ describing the corresponding dynamic domain can be used for multiple purposes including classical planning and diagnostics (see for instance [3,4,7,35]). One way to attack this problem is to replace the transition diagram $T(\mathcal{D})$ by a deterministic transition diagram $T^{lp}(\mathcal{D})$ such that nodes of $T^{lp}(\mathcal{D})$ are partial states of $\mathcal{D}$, its arcs are labeled by actions, and a path in $T^{lp}(\mathcal{D})$ from an initial partial state δ^0 to a partial state satisfying δ^f corresponds to a conformant plan for δ^0 and δ^f in $T(\mathcal{D})$. The transition diagram $T^{lp}(\mathcal{D})$ is called an *approximation* of $T(\mathcal{D})$. Even though $T^{lp}(\mathcal{D})$ normally has many more states than $T(\mathcal{D})$ does, validating whether a given sequence of actions is a conformant plan using $T^{lp}(\mathcal{D})$ is much easier. As pointed out in [6] the use of an approximation can substantially help reduce the complexity of the planning problem. Indeed, an approximation in domains with incomplete information and static causal laws has been developed and applied successfully in the context of conditional and conformant planning in [66]. Of course a drawback of this approach is the possible incompleteness of approximation based planners, i.e., existence of solvable planning problems for which such a planner might not find a solution.

According to this methodology the second step in the development of a conformant planner consists in finding a suitable approximation of $T(\mathcal{D})$. We claim that a concise description of an approximation of $T(\mathcal{D})$ can often be given by a logic program $\pi(\mathcal{D})$ under the answer sets semantics [26,62]. Moreover, complex initial situations and constraints on plans can be also expressed by logic programming rules and included in $\pi(\mathcal{D})$. If this is possible then the problem of finding a parallel or sequential conformant plan can be reduced to computing answer sets of $\pi(\mathcal{D})$. This can be done by general purpose answer set solvers (e.g., [19,33,55]).

If plans are sequential and long then this method can be too time consuming. In this case, the system designer may use $\pi(\mathcal{D})$ as a specification for a procedural graph searching conformant planning algorithm.

This paper illustrates the proposed methodology by building several conformant planners which take as input an action description of $\mathcal{AL}$, an incomplete description of an initial situation, and a description of the goal. In summary, the main contributions of this paper are

- A new approach to defining and computing an approximation of the transition graph of action theories with incomplete initial situation, static causal laws, and parallel actions;
- A sufficient condition for the completeness of the reasoning and/or planning tasks which employ the approximated transition diagram instead of the possible world semantics; and
- Different approximation-based planners that can generate sequential and/or parallel conformant plans. These include the planner CPASP and CPA. The former is a logic programming based planner and can generate both minimal and parallel conformant plans while the latter is a heuristic forward search planner, implemented in C++, and can only generate sequential plans.
- The introduction of fairly simple planning problems with static causal laws that appear to be challenging for many contemporary planners.

In addition, we discuss how complex initial situations and/or constraints on a planning problem can be easily incorporated into our logic programming based planner.

The paper is organized as follows. In the next section, we review the basics of the language $\mathcal{AL}$ including its syntax and semantics, the logic programming representation of transition diagrams specified by $\mathcal{AL}$ action theories, and the problem of conformant planning. In Section 3, we introduce the notion of an approximation of $\mathcal{AL}$ action theories and define a deterministic approximation of an action theory $\mathcal{D}$ by means of a logic program $\pi(\mathcal{D})$. In Section 4, we describe an implementation, in the answer set programming paradigm, of a conformant planner based on this approximation and investigate its completeness in Section 5. Section 6 extends the results in the previous section to planning problems with disjunctive initial states. In Section 7, we describe a heuristic based sequential planner whose basic component is a module for computing the approximation. We provide a comparative study of the performance of our planners against state-of-the-art conformant planners in Section 8. We discuss some advantages of the use of logic programming in conformant planning in Section 9 and conclude in Section 10.

Acknowledgments: We gratefully acknowledge our debt to the pioneering work and influence of John McCarthy. The important ideas used and investigated in this paper, including declarative programming, non-monotonic logic, reasoning about actions and their effects, finding solutions to the frame and ramification problems, and elaboration tolerance all originated from John McCarthy's research [34]. This paper can be viewed as an attempt to apply some of these ideas to the narrow (but hopefully sufficiently important) topic of

conformant planning [2].

The first two authors would like to acknowledge the partial support from the NSF grants CNS-0220590, HRD-0420407, and IIS-0812267. The third author would like to acknowledge the partial support from the NASA contract NASA-NEG05GP48G.

2 Background

We begin with a short review of the syntax and semantics of the language $\mathcal{AL}$ for domains with static causal laws from [8,67], and the notion of a planning problem and its solutions.

2.1 Syntax

The signature Σ of an action theory of $\mathcal{AL}$ consists of two disjoint, non-empty sets of symbols: the set **F** of fluents, and the set **A** of *elementary actions*. By an *action* we mean a non-empty set a of elementary actions. Informally an execution of an action a is interpreted as a simultaneous execution of its components. For simplicity we identify an elementary action e with the action $\{e\}$. A *fluent literal* (or literal for short) l is a fluent or its negation. By $\neg l$ we denote the fluent literal complementary to l, i.e., $\neg(f) = \neg f$ and $\neg(\neg f) = f$. An $\mathcal{AL}$ *action theory* is a set of statements of the following forms:

$$e \textbf{ causes } l \textbf{ if } \psi \tag{1}$$
$$l \textbf{ if } \psi \tag{2}$$
$$\textbf{impossible } a \textbf{ if } \psi \tag{3}$$

where e is an elementary action, a is an action, l is a fluent literal, and ψ is a set of fluent literals from the signature Σ. The set of fluent literals ψ is referred to as the *precondition* of the corresponding statement. When the precondition ψ is empty, the **if** part of the statement can be omitted. Statement (1), called a *dynamic causal law*, says that if e is executed in a state satisfying ψ then l will hold in any resulting state. Statement (2), called a *static causal law*, says that any state satisfying ψ must satisfy l. Statement (3), called an

[2] Some of these results have been reported in the proceedings of the 20th National Conference on Artificial Intelligence [62], the 8th International Conference on Logic Programming and Nonmonotonic Reasoning Conference [61], and the 20th International Joint Conference on Artificial Intelligence [47].

impossibility condition, says that action a cannot be executed in any state satisfying ψ.

To illustrate the syntax of $\mathcal{AL}$, let us consider an instance of (a variant of) the Bomb in the toilet domain [45].

Example 1 There are two packages p_1 and p_2 and two toilets t_1 and t_2. Each of the packages may contain a bomb which can be disarmed by dunking the package into a toilet. Dunking a package into a toilet also clogs the toilet. Flushing a toilet unclogs it. We are safe only if both packages are disarmed.

Figure 1 shows an action theory of $\mathcal{AL}$, denoted by $\mathcal{D}_{bomb}$, that describes the domain[3]. There are four impossibility statements in the action theory. The first one says that "it is impossible to dunk a package into a toilet that is being flushed". The second one states that "it is impossible to dunk two different packages into the same toilet at the same time". The third one says that "it is impossible to dunk a package into two different toilets at the same time". Unlike the first three statements that specify physical impossibilities of concurrent actions, the last one specifies physical impossibility of an elementary action. It says that "it is impossible to dunk a package into a clogged toilet".

In addition to the impossibility statements, the action theory also includes statements to describe the effects of actions *dunk* and $flush$ and the relationship between the fluents $safe$ and *armed*.

□

2.2 Semantics

Intuitively, an $\mathcal{AL}$ action theory describes a transition diagram containing all possible trajectories of the corresponding domain. Before providing the precise definition of such a transition diagram, let us introduce some terminology and notation.

Given an action theory $\mathcal{D}$, a set σ of fluent literals is *consistent* if it does not contain two complementary fluent literals. We say that σ is *complete* if for every fluent f, either f or $\neg f$ belongs to σ. A fluent literal l *holds* in σ if l belongs to σ; l *possibly holds* in σ if $\neg l$ does not belong to σ. A set γ of fluent literals holds (resp. possibly holds) in σ if every fluent literal in γ holds (resp. possibly holds) in σ.

[3] Note that in the description of an action theory, we often use typed variables. A statement with variables are understood as a shorthand for the collection of its ground instances.

Meta variables:

p_i's stand for packages, $i \in \{1, 2\}$, $p_1 \neq p_2$

t_j's stand for toilets, $j \in \{1, 2\}$, $t_1 \neq t_2$.

Fluents:

$armed(p_i)$: package p_i contains the bomb

$clogged(t_j)$: toilet t_j is clogged

$safe$: all the bombs are disarmed

Actions:

$dunk(p_i, t_j)$: dunk package p_i into toilet t_j

$flush(t_j)$: flush toilet t_j

Action theory:

impossible $\{dunk(p_i, t_j), flush(t_j)\}$

impossible $\{dunk(p_1, t_j), dunk(p_2, t_j)\}$

impossible $\{dunk(p_i, t_1), dunk(p_i, t_2)\}$

impossible $dunk(p_i, t_j)$ **if** $clogged(t_j)$

$dunk(p_i, t_j)$ **causes** $\neg armed(p_i)$

$dunk(p_i, t_j)$ **causes** $clogged(t_j)$

$flush(t_j)$ **causes** $\neg clogged(t_j)$

$safe$ **if** $\neg armed(1), \neg armed(2)$

$\neg safe$ **if** $armed(1)$

$\neg safe$ **if** $armed(2)$

Fig. 1. $\mathcal{D}_{bomb}$, the bomb in the toilet theory

A set of fluent literals σ is *closed* under a static causal law (2) if l holds in σ whenever ψ holds in σ. By $Cl_{\mathcal{D}}(\sigma)$ we denote the smallest set of fluent literals that contains σ and is closed under the static causal laws of $\mathcal{D}$.

A *state* s is a complete, consistent set of fluent literals closed under the static causal laws of $\mathcal{D}$. An action b is said to be *prohibited* in s if $\mathcal{D}$ contains an impossibility condition (3) such that ψ holds in s and $a \subseteq b$; otherwise, b is said to be *executable* in s. An action is executable in a set of states S if it is executable in every state $s \in S$.

Given a state s and an action a that is executable in s, a fluent literal l is called a *direct effect* of a in s if there exists a dynamic causal law (1) such that $e \in a$ and ψ holds in s. By $de(a, s)$ we denote the set of all direct effects of a

in s.

The action theory $\mathcal{D}$ describes a transition diagram $T(\mathcal{D})$ whose nodes correspond to possible physical states of the domain and whose arcs are labeled with actions. The transitions of the diagram are defined as follows.

Definition 1 *For an action a and two states s and s', a transition $\langle s, a, s' \rangle \in T(\mathcal{D})$ iff a is executable in s and $s' = Cl_{\mathcal{D}}(de(a, s) \cup (s \cap s'))$.*

Intuitively $\langle s, a, s' \rangle \in T(\mathcal{D})$ indicates that if the system is in state s then after the execution of a the system may move in state s'. Such state s' is called a *possible successor state* of s as a result of the execution of a. If action a is clear from the context then we simply say that s' is a possible successor state of s. It is worth to note that for action theories without concurrent actions, the equation in Definition 1 is equivalent to the one proposed in [41].

Example 2 Consider the action theory $\mathcal{D}_{bomb}$ from Example 1. Let

$$s_0 = \{armed(1), armed(2), \neg clogged(1), \neg clogged(2), \neg safe\}$$

and

$$a = \{dunk(1,1), dunk(2,2)\}$$

Then

$$s_1 = \{\neg armed(1), \neg armed(2), clogged(1), clogged(2), safe\}$$

is the unique successor state of s_0, i.e., $\langle s_0, a, s_1 \rangle \in T(\mathcal{D}_{bomb})$, because

$$\begin{gathered} Cl_{\mathcal{D}_{bomb}}(de(a, s_0) \cup (s_0 \cap s_1)) = \\ Cl_{\mathcal{D}_{bomb}}(\{\neg armed(1), \neg armed(2), clogged(1), clogged(2)\} \cup \emptyset) = \\ \{\neg armed(1), \neg armed(2), clogged(1), clogged(2), safe\} = s_1 \end{gathered}$$

Note that $safe$ belongs to the closure of $\sigma = \{\neg armed(1), \neg armed(2), clogged(1), clogged(2)\}$ because $\mathcal{D}_{bomb}$ contains the static causal law

$$safe \textbf{ if } \neg armed(1), \neg armed(2)$$

and both $\neg armed(1)$ and $\neg armed(2)$ hold in σ.

Now let

$$b = \{dunk(1,1), flush(2)\}$$

Then,

$$s_2 = \{\neg armed(1), armed(2), clogged(1), \neg clogged(2), \neg safe\}$$

is the unique successor state of s_0, i.e., $\langle s_0, b, s_2 \rangle \in T(\mathcal{D}_{bomb})$. □

We next define the notion of a consistent action theory.

Definition 2 (Consistent Action Theory) *An action theory $\mathcal{D}$ is* consistent *if for any state s and action a executable in s, there exists at least one state s' such that $\langle s, a, s' \rangle \in T(\mathcal{D})$.*

Observe that action $\mathcal{AL}$-theories could be nondeterministic and therefore determining whether or not a given action theory is consistent is not a simple task. Indeed, we can prove the following complexity result.

Theorem 1 *Deciding whether or not a given action theory is consistent or not is an NP-complete problem.*

The proof of this theorem is a straightforward reformulation of a similar result in [68] and is therefore not presented here.

It is worth mentioning that the problem is a P-problem for action theories without static causal laws.

Example 3 (Consistent and Inconsistent Action Theories) Consider the following action theory:

$$\mathcal{D}_0 = \begin{cases} e \textbf{ causes } f \textbf{ if } g \\ e \textbf{ causes } \neg f \textbf{ if } h \end{cases}$$

We have that e is executable in $s = \{f, g, h\}$. If s' is a successor state of s then it is easy to see that both f and $\neg f$ belong to s'. This is a contradiction because s' must be consistent. Hence, there exists no successor state for s. According to the above definition, this implies that $\mathcal{D}_0$ is inconsistent.

However, if we add to $\mathcal{D}_0$ the following impossibility condition

$$\textbf{impossible } e \textbf{ if } g, h$$

then the action theory will become consistent because e cannot be executed in any state in which both g and h holds. Hence, at most one of the above dynamic causal laws takes effect, which guarantees the consistency of the theory. □

The consistency of an action theory ensures that the execution of a legal action in a state yields at least one possible successor state. In this paper, *we are interested in consistent action theories only*. We will next define the notion of deterministic action theory.

Definition 3 (Deterministic Action Theory) *An action theory $\mathcal{D}$ is* deterministic *if for any state s and action a, there exists at most one state s'*

such that $\langle s, a, s' \rangle \in T(\mathcal{D})$.

It is easy to see that if an action theory $\mathcal{D}$ does not contain any static causal laws then it is deterministic. In the presence of static causal laws, an action theory, however, may be non-deterministic [4]. The following example shows such an action theory.

Example 4 (Non-Deterministic Action Theory) Consider the following action theory:

$$\mathcal{D}_1 = \begin{cases} e \textbf{ causes } f \\ g \textbf{ if } f, \neg h \\ h \textbf{ if } f, \neg g \end{cases}$$

Let $s_0 = \{\neg f, \neg g, \neg h\}$. We can verify that $\langle s_0, e, s_1 \rangle \in T(\mathcal{D}_1)$ and $\langle s_0, e, s_2 \rangle \in T(\mathcal{D}_1)$ where $s_1 = \{f, h, \neg g\}$ and $s_2 = \{f, g, \neg h\}$. Hence, by definition, $\mathcal{D}_1$ is non-deterministic. □

For our later discussion, the following definition will be useful.

Definition 4 (Entailment) *Let $\mathcal{D}$ be an action theory and M be a path in $T(\mathcal{D})$, i.e., M is an alternate sequence of states and actions $\langle s_0, a_0, s_1, \ldots, a_{n-1}, s_n \rangle$ such that $\langle s_i, a_i, s_{i+1} \rangle \in T(\mathcal{D})$ for $0 \leq i < n$. We say that M* entails *a set of fluent literals σ, written as $M \models \sigma$, if σ holds in s_n.*

For a path $M = \langle s_0, a_0, s_1, \ldots, a_{n-1}, s_n \rangle$ in $T(\mathcal{D})$, s_0 and s_n are referred to as the *initial state* and *final state*, respectively, of M. The sequence of actions $\alpha = \langle a_0, \ldots, a_{n-1} \rangle$ is referred to as a *chain of events*. We also say that M is a *model* of α and sometimes write $\langle s_0, \alpha, s_n \rangle \in T(\mathcal{D})$ to denote that there exists a model of α whose initial state and final state are s_0 and s_n respectively.

A chain of events $\alpha = \langle a_0, a_1, \ldots, a_{n-1} \rangle$ is *executable* in a state s if either (i) $n = 0$, i.e., α is an empty chain of events, or (ii) a_0 is executable in s and $\langle a_1, \ldots, a_{n-1} \rangle$ is executable in every s' such that $\langle s, a, s' \rangle \in T(\mathcal{D})$. A chain of events is executable in a set of states S if it is executable in every state $s \in S$.

2.3 A Logic Programming Representation of $T(\mathcal{D})$

We now describe a logic program, called $lp(\mathcal{D})$, which can be used to compute the transitions in $T(\mathcal{D})$. $lp(\mathcal{D})$ consists of rules for reasoning about the effects

[4] This shows that there exist $\mathcal{AL}$ action theories with deterministic actions which can not be represented in PDDL with deterministic actions and non-recursive axioms.

of actions. Among these rules, the inertial rule encodes the solution to the frame problem, first discussed by John McCarthy and Pat Hayes in their landmark paper on reasoning about actions and changes [44].

The signature of $lp(\mathcal{D})$ includes terms corresponding to fluent literals and actions of $\mathcal{D}$, as well as non-negative integers used to represent time steps. We often write $lp(\mathcal{D}, n)$ to denote the restriction of the program $lp(\mathcal{D})$ to time steps between 0 and n. Atoms of $lp(\mathcal{D})$ are formed by the following (sorted) predicate symbols:

- $fluent(F)$ is true if F is a fluent
- $literal(L)$ is true if L is a fluent literal;
- $h(L, T)$ is true if the fluent literal L holds at time-step T; and
- $o(E, T)$ is true if the elementary action E occurs at time-step T.

In our representation, letters T, F, L, A, and E (possibly indexed) (resp. t, f, l, a, and e) are used to represent variables (resp. constants) of sorts time step, fluent, fluent literal, action, and elementary action correspondingly. Moreover, we also use some shorthands: if a is an action then $o(a, T) = \{o(e, T) \mid e \in a\}$. For a set of fluent literals γ, $h(\gamma, T) = \{h(l, T) \mid l \in \gamma\}$, $not\ h(\gamma, T) = \{not\ h(l, T) \mid l \in \gamma\}$, $\neg\gamma = \{\neg l \mid l \in \gamma\}$, and $lit(\psi) = \{literal(l) \mid l \in \psi\}$. The set of rules of $lp(\mathcal{D})$ is divided into the following five subsets [5]:

(1) *Dynamic causal laws:* for each statement of the form (1) in $\mathcal{D}$, the rule:

$$h(l, T) \leftarrow o(e, T-1), h(\psi, T-1), T > 0 \tag{4}$$

belongs to $lp(\mathcal{D})$. This rule states that if the elementary action e occurs at time step $T-1$ and the precondition ψ holds at that time step then l holds afterward.

(2) *Static causal laws:* for each statement of the form (2) in $\mathcal{D}$, $lp(\mathcal{D})$ contains the rule:

$$h(l, T) \leftarrow h(\psi, T) \tag{5}$$

This rule states that if ψ holds at T then so does l.

(3) *Impossibility conditions:* for each statement of the form (3) in $\mathcal{D}$, $lp(\mathcal{D})$ contains the following rule:

$$\leftarrow o(a, T), not\ h(\neg\psi, T) \tag{6}$$

This rule states that if the precondition ψ possibly holds at time step T then the action a cannot occur at that time step.

[5] For simplicity, we omit atoms of the form $lit(l)$, $lit(\psi)$, and $step(T)$ in the body of the rules.

(4) *Inertia:* $lp(\mathcal{D})$ contains the following rule which solves the frame problem [44]:

$$h(L,T) \leftarrow h(L,T-1), not\ h(\neg L,T), T>0 \tag{7}$$

This rule says that a fluent literal L holds at time step T if it holds at the previous time step and its negation does not hold at T.

(5) *Auxiliary rules:* $lp(\mathcal{D})$ also contains the following rules:

$$\leftarrow h(F,T), h(\neg F,T) \tag{8}$$
$$literal(F) \leftarrow fluent(F) \tag{9}$$
$$literal(\neg F) \leftarrow fluent(F) \tag{10}$$

The first constraint impedes two complementary fluent literals from holding at the same time. The last two rules are used to define fluent literals.

For an action a and a state s, let

$$\Phi(a,s) = lp(\mathcal{D},1) \cup h(s,0) \cup o(a,0) \tag{11}$$

The next theorem states that the program $lp(\mathcal{D})$ correctly implements $T(\mathcal{D})$ (see [58,67]).

Theorem 2 *[58,67] Let s be a state and a be an action. Then $\langle s,a,s'\rangle \in T(\mathcal{D})$ iff there exists an answer set A of $\Phi(a,s)$ such that $s' = \{l \mid h(l,1) \in A\}$.*

2.4 Conformant Planning

The conformant planning problem, as investigated in this paper, has been discussed in [12–15,18,21,52,56] and in our papers [60–62]. Given an action theory $\mathcal{D}$, a set of fluent literals δ is a *partial state* if it is a subset of some state s and is closed under the static causal laws. Intuitively, a partial state represents the knowledge of an agent associated with $\mathcal{D}$ about the current state of the world. For example, $\emptyset$ is a partial state of the action theory $\mathcal{D}_{bomb}$ because $\emptyset$ is closed under the set of static causal laws of $\mathcal{D}_{bomb}$ and it is a subset of the state s_0 in Example 2. Likewise, $\{armed(1), armed(2), \neg safe\}$ is another partial state of $\mathcal{D}_{bomb}$. However, $\{\neg armed(1), \neg armed(2)\}$ is not a partial state of $\mathcal{D}_{bomb}$ because it is not closed under the laws in $\mathcal{D}_{bomb}$.

From now on, we will use symbols σ, s, and δ (possibly indexed) to denote a set of fluent literals, a state and a partial state respectively. For a partial state δ, the *completion of* δ, denoted by $comp(\delta)$, is the set of all states s such that $\delta \subseteq s$.

A (conformant) planning problem is defined as follows.

Definition 5 *A* planning problem $\mathcal{P}$ *is a tuple* $\langle \mathcal{D}, \delta^0, \delta^f \rangle$ *where* $\mathcal{D}$ *is an action theory, and* δ^0 *and* δ^f *are partial states of* $\mathcal{D}$*.*

Observe that in this section we consider planning problems whose initial state description is a set of literals. More general description will be considered in Sections 6 and 9. The solutions of a planning problem are defined as follows.

Definition 6 *Let* $\mathcal{P} = \langle \mathcal{D}, \delta^0, \delta^f \rangle$ *be a planning problem. A chain of events* $\alpha = \langle a_0, \ldots, a_{n-1} \rangle$ *is a* solution *of* $\mathcal{P}$ *if* α *is executable in* $comp(\delta^0)$ *and for every model* M *of* α *with the initial state in* $comp(\delta^0)$*,* $M \models \delta^f$*.*

We often refer to such an α as a *plan* for δ^f. If δ^0 is a state and the action theory $\mathcal{D}$ is deterministic then α is called a *classical* plan; otherwise it is a *conformant plan*. Furthermore, if each a_i of α is an elementary action then α is called a *sequential* plan; otherwise it is called a *parallel* plan. We next illustrate these definitions using the bomb-in-the-toilet example.

Example 5 Consider the action theory $\mathcal{D}_{bomb}$ and let $\delta^0 = \emptyset$ and $\delta^f = \{safe\}$. Then, $\mathcal{P}_{bomb} = \langle \mathcal{D}, \delta^0, \delta^f \rangle$ is a planning problem.

We can check that

$$\alpha_1 = \langle flush(1), dunk(1,1), flush(1), dunk(2,1) \rangle$$

and

$$\alpha_2 = \langle \{flush(1), flush(2)\}, \{dunk(1,1), dunk(2,2)\} \rangle$$

are two solutions of $\mathcal{P}_{bomb}$. The first one is a sequential plan, whereas the second one is a parallel plan. □

3 Approximations of $\mathcal{AL}$ Action Theories

Let $\mathcal{D}$ be an action theory. In this section, we first define what we mean by an *approximation* of the transition diagram $T(\mathcal{D})$ and discuss how approximations can be used to find a solution of a planning problem. Then we introduce a logic program for defining such an approximation.

Let us begin with the definition of an approximation.

Definition 7 (Approximation) *A transition diagram* $T'(\mathcal{D})$ *is an* approximation *of* $T(\mathcal{D})$ *if*

(1) nodes of $T'(\mathcal{D})$ *are partial states of* $\mathcal{D}$ *and arcs of* $T'(\mathcal{D})$ *are labeled with*

actions, and
(2) if $\langle \delta, a, \delta' \rangle \in T'(\mathcal{D})$ *then for every* $s \in comp(\delta)$,
(a) a *is executable in* s *and,*
(b) $\delta' \subseteq s'$ *for every* s' *such that* $\langle s, a, s' \rangle \in T(\mathcal{D})$.

Intuitively, the first condition describes that an approximation $T'(\mathcal{D})$ is a transition diagram between partial states and the second condition requires $T'(\mathcal{D})$ to be *sound* with respect to $T(\mathcal{D})$.

Given an approximation $T'(\mathcal{D})$, we will write $\langle \delta, \alpha, \delta' \rangle \in T'(\mathcal{D})$ to denote that there exists a path corresponding to α from δ to δ' in $T'(\mathcal{D})$ and by convention $\langle \delta, \langle\rangle, \delta \rangle \in T'(\mathcal{D})$ for every partial state δ. We say that $T'(\mathcal{D})$ is *deterministic* if for each partial state δ and action a, there exists at most one δ' such that $\langle \delta, a, \delta' \rangle \in T'(\mathcal{D})$. Even though approximations can be non-deterministic, in this paper we are interested in deterministic approximations only. The next observation shows how the soundness of an approximation extends from transitions to paths.

Observation 3.1 *Let* $T'(\mathcal{D})$ *be an approximation of* $T(\mathcal{D})$. *Then, for every chain of events* α *if* $\langle \delta, \alpha, \delta' \rangle \in T'(\mathcal{D})$ *then for every* $s \in comp(\delta)$,

(1) α *is executable in* s, *and*
(2) $\delta' \subseteq s'$ *for every* s' *such that* $\langle s, \alpha, s' \rangle \in T(\mathcal{D})$.

Observation 3.1 shows that given an approximation $T'(\mathcal{D})$, each path from δ to δ' corresponds to a solution of the planning problem $\langle \mathcal{D}, \delta, \delta^f \rangle$, where $\delta^f \subseteq \delta'$. This gives rise to the following questions:

(1) How to find an approximation of $T(\mathcal{D})$?
(2) How an approximation can be used to solve conformant planning problems?

In the rest of this section, we define an approximation of $T(\mathcal{D})$ called T^{lp}. In the next section, we will use T^{lp} to construct a conformant planner.

In our approach, the transitions in $T^{lp}(\mathcal{D})$ are defined by a logic program $\pi(\mathcal{D})$ called the *cautious encoding* of $\mathcal{D}$. Following the *lp*-function theory from [22], $\pi(\mathcal{D})$ is obtained from $lp(\mathcal{D})$ (see Section 2.3) by adding to it some new rules and modifying the inertial rule (7) to allow $\pi(\mathcal{D})$ to deal with partial states.

Let b be an action and δ be a partial state. We say that b is *safe* in δ if there exists no impossibility condition (3) such that $a \subseteq b$ and ψ possibly holds in δ. A fluent literal l is a *direct effect* (resp. *possible direct effect*) of b in δ if there exists a dynamic causal law (1) such that $e \in b$ and ψ holds (resp. possibly holds) in δ (Recall that a partial state is also a set of fluent literals and thus the concepts of "holds" and "possibly holds" are already defined in Section

2.2). Observe that if b is safe in δ then b is executable in every state $s \supseteq \delta$. Furthermore, the direct effects of b in δ are also the direct effects of b in s which in turn are the possible direct effects of b in δ.

The program $\pi(\mathcal{D})$: The signature of $\pi(\mathcal{D})$ is the same as the signature of $lp(\mathcal{D})$. As before, we write $\pi(\mathcal{D}, n)$ to denote the restriction of $\pi(\mathcal{D})$ to time steps between 0 and n. Atoms of $\pi(\mathcal{D})$ are atoms of $lp(\mathcal{D})$ and those formed by the following (sorted) predicate symbols:

- $de(l, T)$ is true if the fluent literal l is a direct effect of an action that occurs at the previous time step; and
- $ph(l, T)$ is true if the fluent literal l possibly holds at time step T.

We still use letters T, F, L, E, and A (possibly indexed) to represent variables of sorts time step, fluent, fluent literal, elementary action, and action correspondingly. $\pi(\mathcal{D})$ includes

(1) the rules (4)–(6) and (8)–(10) from the program $lp(\mathcal{D})$; and
(2) additional rules defined as follows.
   (a) For each dynamic causal law (1) in $\mathcal{D}$, $\pi(\mathcal{D})$ contains the rule

$$de(l, T) \leftarrow o(e, T-1), h(\psi, T-1), T > 0 \tag{12}$$

   This rule encodes a direct effect of an elementary action e at the time step T. It says that if e occurs at time step $T-1$ and the precondition ψ holds then the fluent literal l is a direct effect of e.

   Since the agent's knowledge about the state of the world at a time step might be incomplete, we add to $\pi(\mathcal{D})$ the rule to define what possibly holds after the execution of an action e at the time step $T-1$:

$$ph(l, T) \leftarrow o(e, T-1), not\ h(\neg\psi, T-1), not\ de(\neg l, T), T > 0 \tag{13}$$

   This rule says that a fluent literal l possibly holds after the execution of an elementary action e if (i) e occurs at the previous time step; (ii) there exists a dynamic causal law (1) for e such that the precondition ψ possibly holds at the previous step; and (iii) $\neg l$ is not a direct effect of some action occurring in the previous time step.
   (b) For each static causal law (2) in $\mathcal{D}$, $\pi(\mathcal{D})$ contains the rule:

$$ph(l, T) \leftarrow ph(\psi, T) \tag{14}$$

   This rule states that if ψ possibly holds at T then so does l.
   (c) In addition, $\pi(\mathcal{D})$ contains the following rule

$$ph(L, T) \leftarrow not\ h(\neg L, T-1), not\ de(\neg L, T), T > 0 \tag{15}$$

This rule completes the definition of the predicate ph. It defines what possibly holds by inertia: a fluent literal possibly holds if (i) it possibly holds at the previous time step; and (ii) its negation is not a direct effect of an action occurring in the previous time step.

(d) Finally, the inertial law is encoded in $\pi(\mathcal{D})$ as follows:

$$h(L,T) \leftarrow not\ ph(\neg L,T), T > 0 \tag{16}$$

which says that L holds at the time moment $T > 0$ if its negation does not possibly hold at T.

For an action a and a partial state δ, let

$$\Pi(a,\delta) = \pi(\mathcal{D},1) \cup h(\delta,0) \cup o(a,0) \tag{17}$$

Then, the program $\Pi(a,\delta)$ has the following property.

Proposition 1 *Let δ be a partial state and a be an action. If $\Pi(a,\delta)$ is consistent then it has a unique answer set B and $\delta' = \{l \mid h(l,1) \in B\}$ is a partial state.*

Proof. See Appendix A.1. □

We define a transition diagram, called $T^{lp}(\mathcal{D})$, based on the program $\pi(\mathcal{D})$ as follows.

Definition 8 *Let $T^{lp}(\mathcal{D})$ be a transition diagram such that*

(1) nodes of $T^{lp}(\mathcal{D})$ are partial states of $\mathcal{D}$ and arcs of $T^{lp}(\mathcal{D})$ are labeled with actions of $\mathcal{D}$, and
(2) $\langle \delta, a, \delta' \rangle \in T^{lp}(\mathcal{D})$ iff $\Pi(a,\delta)$ is consistent and $\delta' = \{l \mid h(l,1) \in B\}$ where B is the answer set of $\Pi(a,\delta)$.

The next theorem states that $T^{lp}(\mathcal{D})$ is indeed an approximation of $T(\mathcal{D})$ and furthermore it is deterministic.

Theorem 3 *$T^{lp}(\mathcal{D})$ is a deterministic approximation of $T(\mathcal{D})$.*

Proof. See Appendix A.2. □

At this point, it is instructive to mention that an approximation for action theories with static causal laws has been introduced in [66]. This approximation is an extension of the 0-approximation in [57]. The approximation defined in this paper is also an extension of the 0-approximation in [57]. Indeed, the following result holds (similar to Theorem 4.6 in [65]).

Observation 3.2 *Let $\mathcal{D}$ be action theory without static causal laws, δ be a*

partial state, and a be an action executable in δ. Then, $\langle \delta, a, \delta' \rangle \in T^{lp}(\mathcal{D})$ if and only if $\delta' = \Phi^0(a, \delta)$ where Φ^0 denotes the 0-approximation in [57].

While the approximations in [57,62,66] are proposed in the context of a given action language, the approximation proposed in this paper, first published in [61], is applicable for action languages whose semantics can be specified by a transition diagram. We note that the approximation proposed in this paper and the one in [66] deal with action theories with static causal laws and the one in [57] does not. Furthermore, only the approximation proposed in this paper is applicable to transition diagrams which allow parallel actions.

The approximation in [66] is based on the idea of computing what can possibly change after an action is executed. A fluent literal l possibly changes its value after the execution of an action e if (a) it is a direct effect of e; or (b) there exists some dynamic law [e **causes** l **if** ψ] such that ψ possibly changes; or there exists some static causal law [l **if** ψ] such that ψ possibly changes. Furthermore, the approximation in [66] does not consider action theories with parallel actions. As such, instead of computing what possibly holds (rules (13), (14), and (15)) and employing this result in the inertial law (rule (16)), the planner in [66] implements rules for computing what possible changes as follows [6]:

$$pc(l, T) \leftarrow o(e, T-1), not\ h(\neg\psi, T)$$
$$pc(l, T) \leftarrow not\ h(l, T-1), pc(l', T), not\ de(\neg\psi, T)$$
$$h(l, T) \leftarrow h(l, T-1), not\ pc(\neg l, T)$$

The first rule is for a dynamic causal law of the form [e **causes** l **if** ψ] and the second rule is for a static causal law of the form [l **if** ψ] with $l' \in \psi$. The last rule encodes the inertial law. It is worth mentioning that the rule encoding the inertial law (16) does not include the atom $h(l, T-1)$, i.e., it is applicable to fluents whose value is unknown (w.r.t. the approximation) before the execution of an action. For this reason, we now favor this approximation over the one developed in [66].

Observe that $T^{lp}(\mathcal{D})$ and the approximation in [66] are incomparable in the sense that $T^{lp}(\mathcal{D})$ sometimes entails some conclusions that could not be derived using the approximation in [66] and vice versa. For instance, for the theory

$$\mathcal{D}_2 = \{a \textbf{ causes } \neg h,\ \neg f \textbf{ if } g\}$$

we have that $\langle \{f\}, a, \{\neg h\} \rangle \in T^{lp}(\mathcal{D})$ (or $\{\neg h\}$ is the partial state resulting from the execution of a in $\{f\}$ according to the approximation T^{lp}) whereas

[6] We adapt the encoding style used in this paper in encoding the approximation in [66].

$\{f, \neg h\}$ is the partial state resulting from the execution of a in $\{f\}$ according to [66]. On the other hand, for the theory

$$\mathcal{D}_3 = \left\{ \begin{array}{lll} a \ \textbf{causes} \ f & a \ \textbf{causes} \ g \ \textbf{if} \ k & g \ \textbf{if} \ f, h \\ g \ \textbf{if} \ f, \neg h & k \ \textbf{if} \ f & p \ \textbf{if} \ g, q \end{array} \right\}$$

we have that $\langle \{\neg f, \neg g, \neg p, \neg q\}, a, \{f, \neg p, \neg q, k\} \rangle \in T^{lp}(\mathcal{D}_3)$ while $\{f, k, \neg q\}$ is the partial state resulting from the execution of a in $\{\neg f, \neg g, \neg p, \neg q\}$ according to [66].

4 An Approximation Based Conformant Planner

Let $\mathcal{P} = \langle \mathcal{D}, \delta^0, \delta^f \rangle$ be a planning problem. Observation 3.1 implies that for any (deterministic) approximation $T'(\mathcal{D})$ of $T(\mathcal{D})$, if α is a chain of events such that $\langle \delta^0, \alpha, \delta' \rangle \in T'(\mathcal{D})$ and $\delta^f \subseteq \delta'$, then α is a solution of $\mathcal{P}$. Because $T^{lp}(\mathcal{D})$ is a deterministic approximation of $T(\mathcal{D})$, we consider the following decision problem.

Conformant planning with respect to $T^{lp}(\mathcal{D})$: Given a planning problem $\mathcal{P} = \langle \mathcal{D}, \delta^0, \delta^f \rangle$, determine whether $\mathcal{P}$ has a solution with respect to $T^{lp}(\mathcal{D})$.

The following complexity result is similar to Theorem 1 in [66].

Theorem 4 *The conformant planning problem with respect to $T^{lp}(\mathcal{D})$ is NP-complete.*

The fact that $T^{lp}(\mathcal{D})$ is deterministic also allows us to use the program $\pi(\mathcal{D})$ to compute solutions of $\mathcal{P}$. In this section, we describe how to construct a logic program from $\pi(\mathcal{D})$ for this purpose. This logic program, denoted by $\pi(\mathcal{P}, n)$, has two input parameters: a planning problem $\mathcal{P}$ and an integer n. The answer sets of $\pi(\mathcal{P}, n)$ contain solutions of length n of $\mathcal{P}$.

Like $\pi(\mathcal{D})$, the signature of $\pi(\mathcal{P}, n)$ includes terms corresponding to fluent literals and actions of $\mathcal{D}$. Rules of $\pi(\mathcal{P}, n)$ include all the rules of $\pi(\mathcal{D}, n)$ and the following rules:

(1) *Initial partial state encoding:* we add to $\pi(\mathcal{P}, n)$ the following facts to describe the initial partial state

$$h(\delta^0, 0)\,. \tag{18}$$

Note that the above is a shorthand for the set of facts $\{h(l, 0). \mid l \in \delta^0\}$.

(2) *Goal encoding:* for each $l \in \delta^f$, $\pi(\mathcal{P}, n)$ contains the constraint:

$$\leftarrow not\ h(l,n) \tag{19}$$

This set of constraints makes sure that every fluent literal in δ^f holds in the final state.

(3) *Action generation rule:* $\pi(\mathcal{P},n)$ contains the following rules for generating action occurrences:

$$o(E,T) \vee \neg o(E,T) \leftarrow T < n \tag{20}$$
$$\leftarrow not\ o(\mathbf{A},T), T < n \tag{21}$$

(Recall that $\mathbf{A}$ is the set of all actions.) These rules state that at any time step $T < n$ at least one action occurs [7].

The following theorem shows that we can use $\pi(\mathcal{P},n)$ to find solutions of $\mathcal{P}$.

Theorem 5 *Let C be an answer set of $\pi(\mathcal{P},n)$ and let $a_i = \{e \mid o(e,i) \in C\}$ $(0 \leq i < n)$. Then, $\alpha = \langle a_0, \ldots, a_{n-1} \rangle$ is a solution of $\mathcal{P}$.*

Proof. See Appendix B. □

It is worth noticing that the program $\pi(\mathcal{P},n)$ is similar to the one presented in [58] in that each of its answer sets corresponds to a solution of the planning problem $\mathcal{P}$. Nevertheless, there are two important differences between $\pi(\mathcal{P},n)$ and the program in [58]. $\pi(\mathcal{P},n)$ can deal with incomplete initial situation and also considers parallel actions. None of these aspects were considered in [58]. The program $\pi(\mathcal{P},n)$ is strongly related to the planner in [66]. They differ from each other in that they implement different approximations and $\pi(\mathcal{P},n)$ can deal with parallel actions and the one in [66] cannot.

Theorem 5 implies that each answer set of $\pi(\mathcal{P},n)$ corresponds to a solution of length n of $\mathcal{P}$. To find *minimal* solutions, we run the program $\pi(\mathcal{P},n)$ with $n = 0, 1, \ldots$ sequentially until it returns an answer set (i.e., the first n such that $\pi(\mathcal{P},n)$ is consistent). The chain of events corresponding to this answer set is a minimal solution of $\mathcal{P}$. This framework is hereafter referred to as the planner CPASP [8].

As will be seen in Section 8, CPASP can solve a wide range of planning problems. However, it is *incomplete*, i.e. there are some planning problems for which a solution exists but cannot be found by CPASP. A precise definition of the (in-)completeness of CPASP is given below.

[7] An alternative for this set of rules is a choice rule

$$1\{o(E,T) : action(E)\} \leftarrow T < n$$

which were introduced in [55].

[8] CPASP stands for **C**onformant **P**lanning using **A**nswer **S**et **P**rogramming.

Definition 9 (Completeness and Incompleteness of CPASP**)** *Let $\mathcal{P}$ be a planning problem. We say that* CPASP *is* complete *with respect to $\mathcal{P}$ if either (i) $\mathcal{P}$ does not have a solution; or (ii) $\mathcal{P}$ has a solution and there exists an integer n such that $\pi(\mathcal{P}, n)$ is consistent. Otherwise, we say that* CPASP *is* incomplete *with respect to $\mathcal{P}$.*

Observe that Theorem 5 shows that if $\mathcal{P}$ does not have a solution then and $\pi(\mathcal{P}, n)$ is inconsistent for every n. One of the main reasons for the incompleteness of CPASP is the inability of the program $\pi(\mathcal{P}, n)$ to do reasoning by cases. The following example demonstrates this issue.

Example 6 Consider the action theory $\mathcal{D}_4$ with two dynamic causal laws

$$e \textbf{ causes } f \textbf{ if } g$$
$$e \textbf{ causes } f \textbf{ if } \neg g$$

Let $\mathcal{P}_4 = \langle \mathcal{D}_4, \emptyset, \{f\} \rangle$. Clearly $\langle e \rangle$ is a solution of $\mathcal{P}_4$ because either g or $\neg g$ is true in any state belonging to $comp(\emptyset)$ and thus one of the above dynamic causal laws would take effect when e is performed. Yet, it is easy to verify that this solution cannot be generated by CPASP due to the fact that for every n, $\pi(\mathcal{P}_4, n)$ does not have any answer set (Constraint (19) cannot be satisfied). □

The next example shows that not only conditional effects but also static causal laws can cause CPASP to be incomplete.

Example 7 Consider the action theory $\mathcal{D}_5$:

$$e \textbf{ causes } f$$
$$g \textbf{ if } f, h$$
$$g \textbf{ if } f, \neg h$$

We can check that $\langle e \rangle$ is a solution of the planning problem $\mathcal{P}_5 = \langle \mathcal{D}_5, \{\neg f, \neg g\}, \{g\} \rangle$ since e causes f to be true and the two static causal laws make g become true whenever f is true.

Now suppose that the program $\pi(\mathcal{P}_5, 1)$ has an answer set C which corresponds to the plan $\langle e \rangle$. This implies that $o(e, 0) \in C$. Then, because of the rule (12), we have that $de(f, 1) \in C$. Furthermore, any atom of the form $de(\dots, \dots)$ belongs to C if and only if it is the head of a ground instance of the rule (12). Thus, the only atom of the form $de(\dots, \dots)$ in C is $de(f, 1) \in C$.

By rule (13), this implies that $ph(f, 1) \in C$. By rule (15), $ph(\neg g, 1)$, $ph(h, 1)$, $ph(\neg h, 1)$ all belong to C. By rule (16), because $ph(\neg g, 1) \in C$, $h(g, 1)$ cannot

belong to C. As a result, constraint (19) is not satisfied. This is a contradiction because C must satisfies all the constraints of $\pi(\mathcal{P}_5, 1)$.

Hence, $\pi(\mathcal{P}_5, 1)$ does not have any answer set. In fact, we can verify that for any integer n, $\pi(\mathcal{P}_5, n)$ does not have an answer set. This implies that CPASP cannot find a solution of $\mathcal{P}_5$. □

The above examples raise a question about the applicability of CPASP: what kind of planning problems can CPASP solve? Observe that the action theories presented in Examples 6 and 7 are rather artificial and are not likely to come up in the specification of real-world domains. In fact, the theories in Examples 6 and 7 can be simplified to $\mathcal{D}'_4 = \{e \textbf{ causes } f\}$ and $\mathcal{D}'_5 = \{e \textbf{ causes } f\} \cup \{g \textbf{ if } h\}$ respectively. It is easy to see that CPASP is complete with respect to the simplified domains. In [59], we developed a transformation that removes such artificial situations. It is worth to mention that CPASP can solve all benchmark problems with non-disjunctive initial state that we have encountered so far.

5 A Sufficient Condition for the Completeness of CPASP

In this section, we present our initial study of the completeness of CPASP. Specifically, we introduce a class of planning problems, called *simple planning problems* (Definition 14) with respect to which CPASP is complete. For convenience, given an action theory $\mathcal{D}$, for a set S of states and action a, by $Res(a, S)$ we denote the set of possible successor states of states in S after the execution of a, i.e.,

$$Res(a, S) = \{s' \mid s \in S, \langle s, a, s' \rangle \in T(\mathcal{D})\}$$

For a chain of events α, by $Res(\alpha, S)$ we denote the set of possible states reachable from some state in S after the execution of α, i.e.,

$$Res(\alpha, S) = \{s' \mid s \in S, \langle s, \alpha, s' \rangle \in T(\mathcal{D})\}$$

Definition 10 (Simple Action Theories) *A static causal law is* simple *if its precondition contains at most one fluent literal. An action theory $\mathcal{D}$ is* simple *if each of its static causal laws is simple.*

By this definition, action theories without static causal laws are simple [9]. Furthermore, we observe that many real world static causal laws are simple

[9] This implies that all benchmarks used in the international planning competitions involve only simple action theories as static causal laws are not used in their representation.

in nature. For example, to represent the unique location of a robot at a time, we can use the following collection of static causal laws:

$$\{\neg at(X) \textbf{ if } at(Y) \mid X \neq Y\}$$

Likewise, to represent the unique value of a multi-valued object obj, we can use:

$$\{\neg value(obj, X) \textbf{ if } value(obj, Y) \mid X \neq Y\}.$$

Observe that simple action theories can be translated into action theories without static causal laws by

- computing $S(l) = \{h \mid$ there exists a sequence $l_0 = l, \ldots, l_n = h$ such that l_i **if** l_{i-1} for $1 \leq i \leq n\}$, for each fluent literal l, and
- replacing a dynamic law $[a$ **causes** l **if** $\varphi]$ with the set of dynamic laws

$$\{a \textbf{ causes } h \textbf{ if } \varphi \mid h \in S(l)\}.$$

A main disadvantage of this approach lies in that the number of dynamic laws might increase quadratically in the number of static laws.

To characterize situations in Examples 6-7, we define a notion of dependency between fluent literals.

Definition 11 (Dependencies Between Literals) *A fluent literal l* depends on *a fluent literal g, written as $l \triangleleft g$, if one of the following conditions holds.*

(1) $l = g$.
(2) There exists a (dynamic or static) causal law of $\mathcal{D}$ such that l is the head and g belongs to the precondition of the law.
(3) There exists a fluent literal h such that $l \triangleleft h$ and $h \triangleleft g$.
(4) The complementary of l depends on the complementary of g, i.e., $\neg l \triangleleft \neg g$.

Note that the dependency relationship between fluent literals is reflexive, transitive but not symmetric. The next definition is about the dependence between actions and fluent literals.

Definition 12 (Dependencies Between Actions and Fluent Literals) *An action b* depends on *a fluent literal l, written as $b \triangleleft l$, if*

(1) there exists an impossibility condition (3) such that $a \subseteq b$ and $\neg l \in \psi$, or
(2) there exists a fluent literal g such that $b \triangleleft g$ and $g \triangleleft l$.

For a set of fluent literals σ and a fluent literal l, we write $l \triangleleft \sigma$ to denote that $l \triangleleft g$ for some $g \in \sigma$ and $l \not\triangleleft \sigma$ to denote that there exists no $g \in \sigma$ such that $l \triangleleft g$.

The next definition characterizes when a set of states S representing the possible states of the world can be reduced to a single partial state δ.

Definition 13 (Reducibility) *Let S be a set of states, δ be a partial state, and σ be a set of fluent literals. We say that S is* reducible *to δ with respect to σ, denoted by $S \gg_\sigma \delta$ if*

(1) δ is a subset of every state s in S,
(2) for any fluent literal $l \in \sigma$, there exists a state $s \in S$ such that $l \not\triangleleft (s \setminus \delta)$, and
(3) for any action a, there exists a state $s \in S$ such that $a \not\triangleleft (s \setminus \delta)$.

There are two interesting properties about the reducibility of a set of states. First, if $S \gg_\sigma \delta$ and S represents the set of possible states of the world then to reason about (formulae composed from) σ, it suffices to know δ.

Proposition 2 *Let $\mathcal{D}$ be a simple action theory, S be a set of states, δ be a partial state, and σ be a set of fluent literals such that $S \gg_\sigma \delta$. Then,*

$$\bigcup_{s \in S} s \cap \sigma = \delta \cap \sigma$$

Proof. See Appendix C.1. □

The reducibility of a set of states is preserved along the course of the execution of actions.

Proposition 3 *Let $\mathcal{D}$ be a simple action theory, S be a set of states, δ be a partial state, and σ be a set of fluent literals such that $S \gg_\sigma \delta$. For any action a, if a is executable in S then*

(1) a is safe in δ,
(2) $Res(a, S) \gg_\sigma \delta'$ where $\langle \delta, a, \delta' \rangle \in T^{lp}(\mathcal{D})$.

Proof. See Appendix C.2. □

The second property can be extended to a chain of events α as follows.

Proposition 4 *Let $\mathcal{D}$ be a simple action theory, S be a set of states, δ be a partial state, and σ be a set of fluent literals such that $S \gg_\sigma \delta$. For any chain of events α, if α is safe in S then*

(1) α is safe in δ
(2) $Res(a, S) \gg_\sigma \delta'$ where $\langle \delta, a, \delta' \rangle \in T^{lp}(\mathcal{D})$.

Proof. See Appendix C.3. □

We define a class of planning problems, called *simple* planning problems, as follows.

Definition 14 (Simple Planning Problems) *A planning problem* $\langle \mathcal{D}, \delta^0, \delta^f \rangle$ *is* simple *if*

(1) $\mathcal{D}$ *is simple, and*
(2) $comp(\delta^0) \gg_{\delta^f} \delta^0$.

The following theorem states that for the class of simple planning problems the planner CPasp is complete.

Theorem 6 *Let* $\mathcal{P} = \langle \mathcal{D}, \delta^0, \delta^f \rangle$ *be a planning problem. If* $\mathcal{P}$ *is simple then* CPasp *is complete with respect to* $\mathcal{P}$.

Proof. See Appendix C.4. □

We would like to conclude this section by relating our completeness condition for approximation based reasoning in action theories with incomplete initial state to other works. The proposed condition is strongly related to the result in [60], in which a completeness condition was developed for action theories without static causal laws [10] . Observe that the proofs of this result in [60] make use of different techniques while the proofs of the completeness result in this paper relies on techniques developed by the logic programming community.

In [49], Palacios and Geffner presented several translations that convert a planning problem $\mathcal{P} = \langle \mathcal{D}, \delta^0, \delta^f \rangle$ into a classical planning problem $\mathcal{P}' = \langle \mathcal{D}', \gamma^0, \gamma^f \rangle$ (γ^0 is complete) whose solutions can be computed using classical planners (e.g. FF). A translation might introduce new fluents and actions. Roughly, the completeness of a translation depends on the set of new fluents and actions. The authors of [49] identified a class of complete translations. They can be characterized by the width of the problem, which depends on a relevance relation between literals. This relation is similar to the dependency relation between literals in Definition 11. Similar to [60], the approach in [49] does not deal with action theories with static causal laws. Nor does it consider parallel actions.

The determinicity of $T^{lp}(\mathcal{D})$ and the fact, that computing the result of the execution of an action in a given partial state can be done in polynomial time (in the size of the theory), imply that computing the result of the execution of an action sequence from a given state can be done in polynomial time. In other words, the temporal projection problem [30] in the class of simple action

[10] This result is similar to Theorem 4 in [60]. Theorem 3 in [60] is incorrect but this does not invalidate Theorem 4. A corrected version of Theorem 3 can be found in [65].

theories is tractable. This result is similar to the result developed by Liu and Levesque in [39]. As with other works, the work [39] does not deal with static causal laws or parallel actions.

6 A Logic Programming Based Planner for Disjunctive Initial State

Besides being sometimes incomplete, another weakness of CPasp is that it does not consider planning problems with disjunctive information about the initial state. We call such problems *disjunctive planning problems* and formally define them as follows.

Definition 15 (Disjunctive Planning Problems) *A* disjunctive planning problem $\mathcal{P}$ *is a tuple* $\langle \mathcal{D}, \Delta^0, \delta^f \rangle$ *where* $\mathcal{D}$ *is an action theory,* Δ^0 *is a non-empty set of partial states and* δ^f *is a partial state.*

It is easy to see that a planning problem in Definition 5 is a special case of a disjunctive planning problem where Δ^0 is a singleton set. Solutions of a disjunctive planning problem are defined as follows.

Definition 16 *A chain of events* $\alpha = \langle a_0, \ldots, a_{n-1} \rangle$ *is a solution of a disjunctive planning problem* $\mathcal{P} = \langle \mathcal{D}, \Delta^0, \delta^f \rangle$ *if for every* $\delta^0 \in \Delta^0$, α *is a solution of the planning problem* $\langle \mathcal{D}, \delta^0, \delta^f \rangle$.

Example 8 Consider the domain

$$\mathcal{D}_6 = \begin{cases} e \textbf{ causes } f \textbf{ if } g \\ e \textbf{ causes } f \textbf{ if } h \end{cases}$$

and let $\Delta^0 = \{\{g\}, \{h\}\}$ and $\delta^f = \{f\}$. Then, $\mathcal{P}_6 = \langle \mathcal{D}_6, \Delta^0, \delta^f \rangle$ is a disjunctive planning problem. By Definition 16, it is easy to see that $\alpha = \langle \{e\} \rangle$ is a solution of $\mathcal{P}_6$. □

It turns out that the framework in the previous section can be naturally extended to find solutions of a disjunctive planning problem $\mathcal{P}$. First, we extend the program $\pi(\mathcal{D})$ to deal with explicit disjunctive information about the initial state; the newly extended program will be referred to as $\Gamma(\mathcal{D})$. Then, we construct a program $\Gamma(\mathcal{P}, n)$ from $\Gamma(\mathcal{D})$ so that every answer set of $\Gamma(\mathcal{P}, n)$, if exists, represents a solution of $\mathcal{P}$.

The basic idea in the development of $\Gamma(\mathcal{D})$ is based on the approach in [66]. We add a second constant to $\Gamma(\mathcal{D})$, called *worlds*, to denote the number of initial partial states in Δ^0, i.e., $worlds = |\Delta^0|$. With the exception of the predicate

$o(A,T)$ and the auxiliary predicates (e.g., $literal(L)$, $fluent(F)$, etc.), every other predicate of $\pi(\mathcal{D})$ is modified to accept a third parameter W to indicate the possible world that the reasoner may be in during the execution of the plan, in the following way:

- $h(l,T,W)$ is true if the fluent literal l holds at time-step T in the world W;
- $de(l,T,W)$ is true if the fluent literal l is a direct effect of an action that occurs at the previous time step in the world W; and
- $ph(l,T,W)$ is true if fluent literal l possibly holds at time step T in the world W.

The rules of $\Gamma(\mathcal{D})$ are obtained from the rules of $\pi(\mathcal{D})$ by replacing predicates $h(l,T)$, $de(l,T)$ and $ph(l,T)$ with the new predicates $h(l,T,W)$, $de(l,T,W)$ and $ph(l,T,W)$ respectively. For example, the rule (4) will become

$$h(l,T+1,W) \leftarrow o(e,T), h(\psi,T,W).$$

Similarly, the rule (13) becomes

$$ph(l,T+1,W) \leftarrow o(e,T), not\ h(\neg\psi,T,W), not\ de(\neg l,T+1,W).$$

As before, we use the notation $\Gamma(\mathcal{D},n)$ to denote the restriction of $\Gamma(\mathcal{D})$ to the time step to take value between 0 and n. Suppose $\Delta^0 = \{\delta_0,\ldots,\delta_{k-1}\}$. The program $\Gamma(\mathcal{P},n)$ is constructed as follows:

(1) The value of the constant *worlds* is k.
(2) The set of rules of $\Gamma(\mathcal{P},n)$ includes
  (a) the rules of $\Gamma(\mathcal{D},n)$, and
  (b) for each $\delta_i \in \Delta^0$, the rule

$$h(\delta_i,0,i).$$

  (c) for each $l \in \delta^f$ and $i \in \{1,\ldots,k\}$, the rule

$$\leftarrow not\ h(l,n,i).$$

The following proposition establishes the relationship between $\Gamma(\mathcal{P})$ and $\Pi(\mathcal{P})$.

Proposition 5 *Let $\mathcal{P} = \langle \mathcal{D}, \{\delta^0\}, \delta^f\rangle$ and $\mathcal{P}' = \langle \mathcal{D}, \delta^0, \delta^f\rangle$. A set of atoms A is an answer set of $\Gamma(\mathcal{P},n)$ iff there exists an answer set B of $\pi(\mathcal{P}',n)$ such that*

(1) for every fluent literal l, $h(l,i,0) \in A$ iff $h(l,i) \in B$, and
(2) for every elementary action e, $o(e,i,0) \in A$ iff $o(e,i) \in B$.

Theorem 7 *Let $\mathcal{P} = \langle \mathcal{D}, \Delta^0, \delta^f \rangle$ be a disjunctive planning problem. If A is an answer set of $\Gamma(\mathcal{P}, n)$ then $\alpha = \langle a_0, \ldots, a_{n-1} \rangle$, where $a_i = \{e \mid o(e,i) \in A\}$, is a solution of $\mathcal{P}$.*

Proposition 5 is intuitive and it is not difficult to prove its correctness. The proof of Theorem 7 is similar to the proof of Theorem 5 in Appendix B. Hence, we omit the proofs of both of them here for brevity.

7 Heuristic Search Through T^{lp}

In the previous sections, we demonstrated the usefulness of approximations in planning in the logic programming paradigm. The experiments (Section 8) show that CPASP is really good in planning with parallel actions and in domains rich in static causal laws. However, it does not perform well in domains with large grounded representation. One of the main reasons for this problem is because the current answer set solvers do not scale up well to programs that require large grounded representation. Various approaches have been proposed to attack this issue from different perspectives [9,20]. To further investigate the usefulness of approximations, we implemented a C++ sequential planner, called CPA, based on the transition diagram induced by T^{lp}. CPA employs the best first search strategy with repeated state avoidance and the number of fulfilled subgoals as the heuristic function. The initial state is described as a CNF formula and thus CPA allows for disjunctive information as well.

One of the main functions inside CPA is to compute the transition function specified by T^{lp}. The algorithm for this function is presented in Figure 2. It takes as input an action theory $\mathcal{D}$, an action a and a partial state δ and returns as output the successor partial state of δ. Note that because T^{lp} is deterministic, such a successor partial state exists and is unique, provided that a is safe in δ. In the algorithm, variables *de* and *pde* denote the set of direct effects and the set of possible direct effects respectively; *ph* is the set of fluent literals that possibly holds in the successor partial state; and *lit* is the set of all fluent literals. The only **for** loop is to compute the sets of direct effects and possible direct effects.

The algorithm makes two calls to the CLOSURE function (depicted in Figure 3). The CLOSURE function takes as input an action theory $\mathcal{D}$ and a set of fluent literals σ and returns as output $Cl_{\mathcal{D}}(\sigma)$. This process is performed in the following steps. First, the closure is set to σ. Then the function loops over the static causal laws, adding to the closure the head of static causal laws whose bodies hold in the previous iteration. The loop terminates when no more fluent literals can be added to the closure.

RES($\mathcal{D}$,a,δ)
Input: An action theory $\mathcal{D}$, an action a, and a partial state δ
Output: successor partial state of δ
1. BEGIN
2. $de = \emptyset$ $pde = \emptyset$ $lit = \mathbf{F} \cup \neg\mathbf{F}$
4. **for each** dynamic causal law (1) in $\mathcal{D}$ **do**
5. **if** ψ holds in δ **then** $de = de \cup \{l\}$
6. **if** ψ possibly holds in δ **then** $pde = pde \cup \{l\}$
6. $ph = \text{CLOSURE}(\mathcal{D}, (pde \cup (lit \setminus \neg\delta)) \setminus \neg de)$
7. **return** $\text{CLOSURE}(\mathcal{D}, de \cup (lit \setminus \neg ph))$
8. END

Fig. 2. An algorithm for computing the transition function of T^{lp}

CLOSURE($\mathcal{D}$,σ)
Input: An action theory $\mathcal{D}$ and a set of fluent literal σ
Output: $Cl_{\mathcal{D}}(\sigma)$
1. BEGIN
2. $\sigma_1 = \sigma_2 = \sigma$
3. **repeat**
4. $fixpoint = true$
5. **for each** static causal law (2) in $\mathcal{D}$ **do**
6. **if** ψ holds in σ_1 and $l \notin \sigma_2$ **then**
7. $\sigma_2 = \sigma_2 \cup \{l\}$ $fixpoint = false$
8. $\sigma_1 = \sigma_2$
9. **until** $fixpoint$
10. **return** σ_1;
11. END

Fig. 3. Computing the closure of a set of fluent literals

The correctness of the algorithm is stated in the following theorem.

Theorem 8 *Let $\mathcal{D}$ be an action theory, a be an action, and δ be a partial state. If a is safe in δ then* $\text{RES}(\mathcal{D}, a, \delta) = \delta'$ *iff* $\langle \delta, a, \delta' \rangle \in T^{lp}(\mathcal{D})$.

Proof. See Appendix D. □

We would like to note that CPA also deals with disjunctive information about the initial state. CPA employs an explicit DNF representation of the initial state. Given an CNF representation of the initial state [11] , CPA converts it into its internal DNF representation. Naturally, this might be very time consuming

[11] In PDDL, this is specified by `oneof-` and `or`-clauses.

and could affect the planner's performance. Finding an implementation of CpA that does not require an DNF representation of the initial state is an interesting topic but is beyond the scope of this paper and is considered as one of our goals in the future.

8 Experiments

In this section, we will present our experimental evaluation of the performance of CPasp and CpA. The platform used for testing CPasp is a 2.4 GHz CPU, 768MB RAM machine, running Slackware 10.0 operating system. Experiments with CpA and sequential planners are conducted on a 3.2 Ghz CPU, 2GB RAM, running Suse Linux 9.0 operating system. Every experiment had a time limit of 30 minutes.

8.1 Evaluation of CPasp

8.1.1 Planning Systems Used

We compared CPasp with three other conformant planners: CMBP [15], DLV$^{\mathtt{K}}$ [18], and *C*-Plan [14]; We selected these planners because they are designed in spirit similar to CPasp (that is, a planning problem is translated into an equivalent problem in a more general setting which can be solved by an off-the-shelf software system) and can directly deal with static causal laws. The main difference between these planners and CPasp lies in the use of different types of reasoning in searching for plans. CMBP, DLV$^{\mathtt{K}}$, and *C*-Plan use the possible world semantics whereas CPasp uses the approximation. A brief overview of these planners is given below.

- CMBP (Conformant Model Based Planner): CMBP is a conformant planner developed by Cimatti and Roveri [15]. CMBP employs BDD (Binary Decision Diagram) techniques to represent planning domains and search for solutions. CMBP allows non-deterministic domains with uncertainty in both the initial state and action effects. However, it does not have the capability of generating concurrent plans. The input language is the action language $\mathcal{AR}$ [28]. The version used for testing was downloaded from `http://www.cs.washington.edu/research/jair/contents/v13.html` .
- DLV$^{\mathtt{K}}$: DLV$^{\mathtt{K}}$ is a declarative, logic-programming-based planning system built on top of the DLV system (`http://www.dbai.tuwien.ac.at/proj/dlv/`). Its input language $\mathcal{K}$ is a logic-based planning language described in [18]. The version used for testing was downloaded from `http://www.dbai.tuwien.ac.at/proj/dlv/K/` . DLV$^{\mathtt{K}}$ is capable of generating both concurrent and confor-

mant plans.
- *C*-PLAN: *C*-PLAN is a SAT-based conformant planner. *C*-PLAN works using a generate-and-test method. The input language is the action language $\mathcal{C}$ [21,29]. *C*-PLAN is primarily designed for generating concurrent plans.

8.1.2 Benchmarks

We prepared two test suites. The first test suite contains sequential, conformant planning benchmarks and the second contains concurrent, conformant planning benchmarks. Many benchmarks are taken from the literature (e.g. [12,14,18]) and some were developed for the international planning competition (e.g. [17]). To test the performance of our systems in domains with static causal laws, we developed two simple domains which are rich in static causal laws.

Sequential Benchmarks. The sequential benchmark test suite includes the following domains:

- **BT(m,n)**: This domain is a variant of the well-known Bomb-In-the-Toilet domain. In this domain, there are m packages and n toilets. One of the packages contains a bomb. The bomb can be disarmed by dunking the package that contains it into a toilet. The goal is to disarm the bomb.
- **BTC(m,n)**: This domain is similar to BT. However, we assume that dunking a package into a toilet will clog the toilet; flushing the toilet will make it unclogged.
- **RING(n)**: The encoded version of this problem follows the encoding in the distribution of CFF. In this domain, the agent can move (*forward* or *backward*) around a building with n-rooms arranged in a ring in a cyclic fashion. Each room has a window which can be closed or open. Closed windows may be locked. Initially, neither the location of the agent nor the states (open/locked) of the windows is known. The goal is to have all windows locked. A possible conformant plan consists of performing actions *forward, close, lock* repeatedly. Notice that the initial location of the agent needs to be represented as a disjunction.

  Observe that the location of the agent satisfies the well-known static causal law stating that an agent cannot be in two places at the same time. We therefore created a domain, called **RING-C(n)**, by introducing this static causal law.

  We should remark here that this domain is used in different ways in our experiments. In testing CPASP, we assume that the location of the agent is known in the initial state. This is because CPASP does not deal with disjunctive information. On the other hand, we assume that the location of the agent is unknown in the initial state when testing CPA.

- **Domino(n)**: We have n dominoes standing on a line in such a way that if one of them falls then the domino on its right side 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 dominoes are unknown. The goal is to have the rightmost domino fall.
- **Gaspipe(n)**: The objective of this domain is to start a flame in a burner which is connected to a gas tank through a pipe line. The gas tank is on the left-most end of the pipe line and the burner is on the right-most end. The pipe line is made of sections connected with each other by valves. The pipe sections can be either pressured by the tank or un-pressured. Opening a valve causes the section on its right side to be pressured if the section to its left is pressured. Moreover, for safety reasons, a valve can be opened only if the next valve on the line is closed. Closing a valve causes the pipe section on its right side to be un-pressured. Static causal laws are useful in the representation of this domain. One such law is that if a valve is open and the section on its left is pressured then the section on its right will be pressured. Otherwise (either the valve is closed or the section on the left is un-pressured), the pipe on the right side is un-pressured. The burner will start a flame if the pipe connecting to it is pressured. The gas tank is always pressured. The uncertainty in the initial situation is that the states of the valves are unknown. A possible conformant plan will be to close all valves except the first one (that is, the one that directly connects to the gas tank) from right to left and then opening them from left to right.
- **Cleaner(n,p)**: This domain is a modified version of the Ring domain in which instead of locking windows, the goal of the agent is to clean multiple objects located in every room (there are p objects in each room). To contrast with the Ring domain, we assume that initially, the agent is in the first room and does not know whether or not any of the objects are cleaned.

Concurrent Benchmarks. There are four domains in this test suite, namely BT^p, BTC^p, Gaspipe^p and Cleaner^p. The BT^p and BTC^p domains are modifications the BT and BTC domains respectively in which we allow to dunk different packages into different toilets at the same time. The Gaspipe^p domain is a modification of the Gaspipe domain, which allows to close multiple valves at the same time. In addition, it is possible to open a valve while closing other valves. However, it is not allowed to open and close the same valve or open two different valves at the same time. The final domain in the test suite, Cleaner^p, is a modified version of the Cleaner domain where we allow the robot to concurrently clean multiple objects in the same room. Modifications to the Ring and Domino domain are not considered as these problems only have sequential solutions.

8.1.3 Experimental Results

We ran CPasp using the answer set solvers `smodels` [55] and `cmodels` [33] and observed that `cmodels` yielded better performance in general. The running times of CPasp reported here were obtained using `cmodels`. The timing results for the sequential benchmarks is shown in Tables 1 & 2, and the results for the concurrent benchmarks is shown in Table 3. We did not test *C*-Plan on the sequential planning benchmarks since it was developd only for concurrent planning. In these tables, times are shown in seconds; The "PL" column shows the length of the plan found by the planner. "-" denotes that the planner did not return a solution within the time limit for some reasons: e.g., out of time or out of memory. "NA" denotes that the problem was not run with the planner. Since both DLV^K and CPasp require as an input parameter the length of a plan to search for, we ran them inside of a loop in which we incrementally increase the plan length to search for, starting from 1 [12] , until a plan is found. Notice that in this way CPasp is not only finding conformant plans but minimal conformant plans with respect to the defined approximation. For example, *C*-Plan and CPasp took 0.74 second and 2.72 second, respectively, to find the solution for the BTC(6,2) problem. Nevertheless, CPasp's solution is shorter.

Table 1
Sequential Benchmarks: Bomb & Ring domains

	CMBP		DLV^K		CPasp	
Problem	PL	Time	PL	Time	PL	Time
BT(2,2)	2	0.03	2	0.04	2	0.20
BT(4,2)	4	0.17	4	0.55	4	0.41
BT(6,2)	6	0.21	6	216.55	6	0.77
BT(8,4)	8	0.63		-	8	6.73
BT(10,4)	10	1.5		-	10	890.06
BTC(2,2)	2	0.16	2	0.12	2	0.22
BTC(4,2)	6	0.26	6	72.44	6	0.71
BTC(6,2)	10	0.74		-	8	2.72
BTC(8,4)		-		-		-
BTC(10,4)		-		-		-
Ring(2)	5	0.06	5	0.10	5	0.52
Ring(4)	11	0.10	11	2.14	11	2.98
Ring(6)	17	0.48		-	17	44.43
Ring(8)		-		-	23	1424.07
Ring(10)		-		-		-

As it can be seen in Table 1, in the BT and BTC domains CMBP outperforms both DLV^K and CPasp on most problem instances. In general CPasp has better

[12] We did not start from 0 because none of the benchmarks has a plan of length 0.

performance than DLV^K on these domains. As an example, DLV^K took more than three minutes to solve BT(6,2), while it took only 0.77 seconds for CPASP to solve the same problem. In addition, within the time limit, CPASP was able to solve more problems than DLV^K. In the Ring domain, although outperformed by both CMBP and DLV^K in some small instances, CPASP is the only planner that was able to solve Ring(8).

Table 2
Sequential Benchmarks: Domino, Gaspipe & Cleaner domains

	CMBP		DLV^K		CPASP	
Problem	PL	Time	PL	Time	PL	Time
Domino(100)	1	0.26	1	0.1	1	0.21
Domino(200)	1	1.79	1	0.35	1	0.28
Domino(500)	1	7.92	1	2.40	1	0.74
Domino(1000)	1	13.20	1	13.10	1	1.23
Domino(2000)	1	66.60	1	62.42	1	2.41
Domino(5000)	1	559.46		-	1	6.07
Domino(10000)		-		-	1	12.584
Gaspipe(3)		NA	5	0.13	5	1.34
Gaspipe(5)		NA	9	0.42	9	2.22
Gaspipe(7)		NA	13	42.62	13	6.18
Gaspipe(9)		NA		-	17	39.32
Gaspipe(11)		NA		-	21	868.10
Cleaner(2,2)	5	0.1	5	0.104	5	0.49
Cleaner(2,5)	11	0.61	11	214.69	11	3.88
Cleaner(2,10)		-		-		-
Cleaner(4,2)	11	0.13	11	14.82	11	2.09
Cleaner(4,5)		-		-		-
Cleaner(4,10)		-		-		-
Cleaner(6,2)	17	4.1		-	17	224.39
Cleaner(6,5)		-		-		-
Cleaner(6,10)		-		-		-

CPASP works well with domains rich in static causal laws (Domino and Gaspipe). In the Domino domain, CPASP outperforms all the other planners in most of instances. It took only 2.41 seconds to solve Domino(2000), while both DLV^K and CMBP took more than one minute. In fact CPASP could scale up very well to larger instances, e.g., Domino(10000). In the Gaspipe domain, CPASP also outperforms DLV^K: it was able to solve all the problem instances while DLV^K was able to solve only the first three problem instances [13].

[13] We tried to test this domain with CMBP but had some problem with the encoding. We contacted the author of CMBP and are still waiting for response

Table 3
Concurrent Benchmarks: Bomb, Gaspipe & Cleaner domains

	C-Plan		DLV^K		CPasp	
Problem	PL	Time	PL	Time	PL	Time
BT^p(2,2)	1	0.07	1	0.07	1	0.11
BT^p(4,2)	2	0.05	2	0.09	2	0.26
BT^p(6,2)	3	1.81	3	3.06	3	0.34
BT^p(8,4)	2	4.32	2	10.52	2	0.24
BT^p(10,4)		-		-	3	1.91
BTC^p(2,2)	1	0.05	1	0.05	1	0.13
BTC^p(4,2)	3	0.07	3	0.90	3	0.30
BTC^p(6,2)	5	7.51	5	333.27	5	0.67
BTC^p(8,4)		-		-	3	0.50
BTC^p(10,4)		-		-	5	1192.45
Gaspipe^p(3)		-	4	0.08	4	0.40
Gaspipe^p(5)		-	6	0.17	6	0.75
Gaspipe^p(7)		-	8	0.44	8	1.22
Gaspipe^p(9)		-	10	17.44	10	3.17
Gaspipe^p(11)		-		-	12	8.83
Cleaner^p(2,2)	3	0.05	3	0.07	3	0.26
Cleaner^p(2,5)	3	0.12	3	0.06	3	0.30
Cleaner^p(2,10)	3	0.06	3	0.07	3	0.30
Cleaner^p(4,2)	7	0.06	7	0.19	7	0.77
Cleaner^p(4,5)	7	0.09	7	0.80	7	0.93
Cleaner^p(4,10)	7	0.13	7	237.63	7	1.16
Cleaner^p(6,2)	11	0.11	11	4.47	11	1.98
Cleaner^p(6,5)	11	0.19	11	986.73	11	2.94
Cleaner^p(6,10)	11	0.35		-	11	3.73

The Cleaner domain turns out to be hard for the planners: they could solve very small instances only. In this domain, CPasp is outperformed by CMBP. To solve the Cleaner(6,2), CMBP took only 4.1 seconds while CPasp took more than 3 minutes. However, CPasp performs better than DLV^K in general: DLV^K reported a timeout with the problem Cleaner(6,2).

We have seen that CPasp can be competitive with CMBP and DLV^K on the sequential benchmarks. Let us move our attention now to the concurrent benchmarks. As can be seen from Table 3, CPasp outperforms both DLV^K and *C*-Plan on most instances of the BT^p, BTC^p, and Gaspipe^p domains. Furthermore, CPasp is the only planner that was able to solve all the instances in the test suite. In the Cleaner domain, *C*-Plan is the best. To solve the Cleaner^p(6,10) problem, *C*-Plan took only 0.35 seconds, whereas DLV^K reported a timeout and CPasp needed 3.73 seconds. As CMBP does not produce concurrent plans, it is not included in this table.

8.2 Evaluation of CpA

8.2.1 Planning Systems Used

We compared CpA with four conformant planners: CFF [12], KACMBP [16], t_0 [49], and POND [13]. These planners were, at the time of our experiments, known as the fastest conformant planners on most of the conformant planning benchmarks in the literature [14]. CFF [12] is superior to other state-of-the-art conformant planners like GPT [11], MBP [15]; KACMBP is reported [16] to outperform DLV$^{\text{K}}$ and *C*-Plan in many domains in the literature. t_0 [49] is known to be better than CFF in many domains. We did not compare our planners with the PKS system [50] (improved in [51]) which employs a richer representation language and the knowledge-based approach to reason about effects of actions in the presence of incomplete information as most planners used in the comparison are domain-independent planners. A brief overview of these planners is given below.

- KACMBP: KACMBP is an extension of CMBP. Similarly to CMBP, KACMBP uses techniques from symbolic model checking to search in 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 designed for both sequential and concurrent settings. The input language of KACMBP is SMV. The system used in the experiments was downloaded from `http://sra.itc.it/tools/mbp/AIJ04/`.
- Conformant-FF (CFF): CFF [15] extends the classical FF planner [31] 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 the PDDL language with a minor change that allows the users to specify the initial state as a CNF formula. CFF supports sequential conformant planning. However, it does not support concurrent and conditional planning.
- POND: POND extends the planning graph algorithm [10] to deal with sensing actions. Conformant planning is also supported as a feature of POND. The input language is a subset of PDDL. The version for testing was obtained through email communication with Daniel Bryce, author of POND. This is the version 1.1.1. of POND.

[14] In a recent planning competition, an improved version of CpA won the first price over t_0 (see `http://ippc-2008.loria.fr/wiki/index.php/Results`). The version of CpA used in this paper, however, is the version used at the time of the original submission of this paper.

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

- t_0 is the winner of the 2006 International Planning Competition, in the "Conformant Planning" category. t_0 solves a planning problem $\mathcal{P}$ by translating it into a classical planning problem $\mathcal{P}'$. The earlier version of t_0, called `cf2cs(ff)`, [48] is incomplete but t_0 [49] is complete. As with other planners (e.g., CFF, POND), t_0 does not deal with static causal laws.

8.2.2 Benchmarks

We evaluated the performance of the planning systems on the four domains tested with CPASP: Ring, Domino, Cleaner, and BTC. In addition, we use four other domains: Cube-Center (Cube-C), Logistic (Log), Sortnet, and UTS.

The Ring, Domino, BTC and Cleaner domains have been described in the previous section. We remark that the initial location of the agent is unknown in the Ring domain for the experiments conducted in this section.

The Cube-Center($2k+1$) domain is described in [16]. In this domain, an agent can move *up, down, left*, and *right* in a three-dimension grid of the size $2k+1$. A move results in the agent being in a corresponding cell (e.g., moving *up, down* changes the y-coordinate of the agent) or being in the same cell (e.g., moving *up* when the y coordinate is $2k+1$ does not result in any change). The goal of the agent is to be at the center of the Cube. Initially, the agent does not know its location.

The Logistic(l,c,p) domain is described in [12] and distributed together with CFF. In this domain, we need to transport p packages between locations of c cities (each city has p locations) via trucks and airplanes. The uncertainty in the initial state is that the exact locations of the packages are unknown in the beginning.

The Sortnet(n) and UTS(n) are two domains from the 2006 International Planning Competition, downloaded from `http://www.ldc.usb.ve/~bonet/`. The Sortnet(n) domain is a synthesis of sorting networks which has disjunctive goals and a large number of possible initial states. The UTS(n) domain is the computation of universal transversal sequence for graphs. In this domain, the goal is to visited all the nodes. To do so, an agent must *start* traveling. Afterwards, he/she can visit neighbor nodes. The uncertainty lies in the initial location of the agent.

8.2.3 Experimental Results

The experimental results are shown in Tables 4-7. Times are shown in seconds. "-" indicates that the planner did not return a solution within the time limit and "NA" indicates that the problem is not applicable. "AB" denotes that

Table 4
Classical Domains: Ring, Cube-Center, BTC, and Logistics

Prob.	KACMBP		CFF		POND		t_0		CPA	
Size	PL	Time	PL	Time	PL	Time	PL	Time	PL	Time
Ring(n)										
2	8	0.00	7	0.01	6	0.00	5	0.04	5	0.00
3	15	0.01	15	0.12	13	0.10	8	0.05	8	0.05
4	22	0.02	26	2.13	16	4.34	13	0.06	11	0.40
5	35	0.03	45	40.77	20	247.45	17	0.05	14	2.05
Cube-Center(n)										
3	14	0.04	6	0.00	6	0.55	6	0.08	6	0.10
5	25	0.16	33	0.28	15	10.00	15	0.12	24	9.82
7	35	0.16	40	22.62	24	162.85	28	0.14	42	117.57
9	45	0.32		-		-	45	0.32	73	797.84
11	55	0.36		-		-	48	0.23	81	1405.24
BTC(p,t)										
10,1	19	0.01	19	0.01	19	0.00	19	0.06	19	0.00
20,1	39	0.04	39	0.06	39	0.02	39	0.06	39	0.05
50,1	99	0.35	99	2.18	99	0.19	99	0.10	99	0.64
100,1	199	2.52	199	57.77	199	2.05	199	0.24	199	4.98
10,5	15	0.06	15	0.01	15	0.01	15	0.03	15	0.03
20,5	35	0.18	35	0.02		AB	35	0.06	35	0.12
50,5	95	0.96	95	1.82	97	0.63	95	0.12	95	1.48
100,5	195	3.94	195	53.49	195	1.80	195	0.42	195	9.85
10,10	10	0.17	10	0.00	10	0.02	10	0.05	10	0.05
20,10	30	0.55	30	0.03	30	0.13	30	0.08	30	0.31
50,10	90	2.96	90	1.52	94	1.45	90	0.16	90	2.88
100,10	190	17.42	190	47.85	194	17.97	190	0.42	190	16.84
Logistics(l,c,p)										
2,2,2	14	0.13	16	0.00	16	0.07	16	0.05	10	0.11
2,3,3	34	161.13	24	0.00	30	1.49	24	0.08	48	7.43
3,2,2	14	0.12	20	0.02	22	0.45	20	0.08	44	3.33
3,3,3	40	.19	34	0.03	38	24.07	34	0.11	350	340.09
4,3,3		-	37	0.04	37	34.32	36	0.15		-

the program encountered some error and halted abnormally.

As can be seen in Table 4, in the Ring domain, KACMBP performs the best. t_0 is almost as good as KACMBP. CFF and POND on the contrary did not perform well on this domain. As explained in [12], CFF does not perform well on this domain because of the lack of informativity of the heuristic function

in the presence of non-unary effect conditions and the problem with checking repeated states. Although CpA was able to solve all the instances, its performance is not as good as KACMBP or t_0, e.g., to solve Ring(5), CpA needed more than two seconds, while KACBMP or t_0 needed less than 0.05 seconds.

A similar performance trend can be observed in the Cube-Center domain. t_0 and KACMBP dominate in this domain and t_0 yields better performance and also shorter plans than KACMBP in large instances. CpA outperforms POND but is slower than CFF in instances solvable by CFF. Again, CpA was able to solve all instances in this domain but its solutions are usually longer than the solutions returned by other planners.

In the BTC domain, t_0 is better in general. POND and KACMBP are comparable to each other. CpA is competitive with others in many instances. It outperforms CFF, POND, and KACMBP in BTC(100,10) but takes twice the time comparing to POND and KACMBP in solving the BTC(100,1) instance. It is interesting to observe that CFF and t_0 seem to have no problem when the number of toilets increases, while there is a significant increase in the amount of time needed to find a solution for KACMBP, POND, and CpA. This increase is, however, less substantial for CpA than for KACMBPor POND. For example, if the number of packages is fixed to 100 and the number of toilets increases from 5 to 10, the amount of time needed by CFF even decreases, while the time needed by KACMBP increases about 5 times, CpA's time doubles, and POND's time increases more than 10 times while t_0's remains the same.

In the Logistic domain, both KACMBP and CpA had difficulty in finding plans. Although KACMBP performs better than CpA, its performance is far from that of CFF or t_0, which solved every problem instance in less than one second. POND can solve all problems but the time increases very fast. We believe that the poor performance of CpA on this domain lies in the not-so-good heuristic function used. This is reflected in the lengths of the plans found by CpA.

Table 5 contains experimental results on the performance of the planners in domains that are different from domains described in Table 4 in the following way: these domains are either rich in static causal laws or have a high degree of uncertainty in the initial state. The domains encoded for CpA use static causal laws. These static causal laws are compiled away in the encodings used for other planners, following the procedure suggested in [63].

In the Domino domain, with the exception of CpA, none of the other planners performed well. The reason being that this domain is rich in static causal laws – a feature that is not directly supported by KACMBP, CFF, t_0, or POND. Thus, we had to compile them away when encoding this domain for these

planners, which requires the introduction of extra actions and fluents. As a result, the performance of these planners is hit by an extra overhead that CPA is not affected with. It is worth mentioning that the 'NA' in CFF's column is due to some predefined constants of the system (e.g. maximal plan length is set to 500).

In the Cleaner domain, CPA also obtained good performance and it is the only planner that could solve all instances. KACMBP behaves well only on small problems but does not scale up as well as CFF and CPA. CFF is very good at all instances of the domain, except Cleaner(5,100) where it reported the error message: "maximum length of plans is exceeded". We believe that this can easily be fixed by increasing some constant for the maximum length of plans allowed. POND could not solve only one instance in this set of problems. In this domain, the high degree of uncertainty in the initial state increases with the number of objects.

It is interesting to observe how static causal laws affect the performance of CPA and how different reasoning method could be useful in other planners. As we have mentioned before, the only difference between the Ring-C domain and the Ring domain is that we add the encoding of the static causal laws "an agent cannot be at two different locations at the same time" to the problem specification. In the encoding used to run CPA, it is expressed as a static causal law. This information is expressed in the encoding for other planners by adding the additional (negative) effects to the *forward/backward* of the domain. For example, if we have three rooms 1, 2, and 3, the encoding for CPA will include the action $forward$ **causes** in_{i+1} **if** in_i where $i = 1, 2, 3$ and $in_4 \equiv in_1$, and the static causal law $\neg in_j$ **if** in_i for $i \neq j$; the encoding of this action for other planners will have conditional effects of the form $(\texttt{when}(in_3)(in_1 \wedge \neg in_2 \wedge \neg in_3))$ etc. CPA yields the best result in this domain. Only CPA and t_0 were able to solve all instances of this problem. Neither CFF nor POND returned a solution for the smallest instance of this domain (Ring-C(10)). The difference between t_0 and CFF indicates that in this domain, the method employed by t_0 is better than that of CFF. It is likely that the translation t_0 introduces only a few actions and fluents into the problem which ultimately results in better performance.

We have seen that CPA performed reasonably well in several domains. Let us now focus on its performance in the two domains Sortnet (Table 6) and UTS (Table 7). As we have mentioned earlier, the Sortnet domain has disjunctive goals. The two systems CFF and t_0 does not consider problems with disjunctive goals. CPA does not consider disjunctive goals directly and we have to introduce additional fluents and static causal laws to encode the goal. For example, we replace a goal of the form $f \vee g$ by the fluents fg and add to the

Table 5
Domains with Static Causal Laws and High Degree of Uncertainty: Ring-C, Domino, an Cleaner

Prob.	KACMBP		CFF		POND		t_0		CPA	
Size	PL	Time	PL	Time	PL	Time	PL	Time	PL	Time
Domino(n)										
10	23	0.04	10	0.00	10	0.00	10	0.05	1	0.00
50	163	0.13	50	0.00	50	0.35	50	0.07	1	0.01
100	376	1.45		NA	100	4.51	100	0.21	1	0.02
200	852	16.42		NA	200	128.26	200	1.93	1	0.02
500		-		NA	500	378.44	500	58.34	1	0.05
1000		-		NA	1000	1258.17	1000	745.95	1	0.16
2000		-		NA		-		-	1	0.52
5000		-		NA		-		-	1	3.12
Cleaner(r,o)										
2,10	21	0.04	21	0.00	21	19.30	21	0.05	21	0.03
2,20	41	0.36	41	0.02		-	41	0.08	41	0.19
2,50	101	7.89	101	0.29		-	101	.10	101	2.76
2,100	201	113.99	201	3.48		-	201	0.23	201	22.71
5,10	56	0.57	54	0.03		-	54	0.06	54	0.26
5,20	106	4.75	104	0.27		-	104	0.10	104	1.78
5,50	256	143.40	254	7.46		-	254	0.23	254	26.66
5,100		-		NA		-	504	0.79	504	214.27
Ring with Static Causal Laws, Ring-C(n)										
10	85	0.49		-		-	30	1.01	30	0.83
15	231	9.03		-		-	45	6.76	45	5.57
20	270	503.99		-		-	60	27.44	60	22.20
25		-		-		-	75	79.58	75	65.32

problem the set of static causal laws $\{fg$ **if** f, fg **if** $g, \neg fg$ **if** $\neg f \wedge \neg g\}$. We observe that this encoding is not applicable to the CFF and t_0 systems. In this domain, POND is the best. It can solve every instance in less than half a second with the exception of the instance Sortnet(9) where it halts abnormally. KACMBP can solve all instances but it was much slower than POND. CPA, on the other hand, can only solve instances from 1 to 10 (when $n = 11$, the number of partial states in the initial state is 2^{11}). This domain indicates that the explicit representation of the set of partial states used by CPA needs to be compensated in some ways for the planner to scale up.

Finally, Table 7 indicates that the UTS domain is also a challenging problem for CPA and KACMBP. In this domain, the number of initial states is linear in the size of the problem. We believe that the heuristic used by CPA is not

Table 6
Sortnet Domains

	KACMBP		CFF		POND		t_0		CPA	
Problem	PL	Time	PL	Time	PL	Time	PL	Time	PL	Time
Sortnet(1)	1	0.00		NA	1	0.00		NA	1	0.00
Sortnet(2)	3	0.00		NA	3	0.00		NA	3	0.00
Sortnet(3)	6	0.02		NA	5	0.00		NA	5	0.05
Sortnet(4)	10	0.05		NA	9	0.00		NA	9	0.40
Sortnet(5)	15	0.19		NA	12	0.01		NA	15	0.66
Sortnet(6)	21	0.20		NA	16	0.00		NA	21	9.76
Sortnet(7)	28	0.42		NA	19	0.01		NA	27	35.51
Sortnet(8)	36	0.76		NA	26	0.01		NA	35	117.41
Sortnet(9)	45	1.24		NA	31	*		NA	44	371.58
Sortnet(10)	55	2.18		NA	38	0.05		NA	55	1010.10
Sortnet(11)	66	3.22		NA	43	0.06		NA		-
Sortnet(12)	78	5.55		NA	50	0.11		NA		-
Sortnet(13)	91	8.24		NA	55	0.13		NA		-
Sortnet(14)	105	20.62		NA	61	0.20		NA		-
Sortnet(15)	120	28.54		NA	65	0.22		NA		-

providing good guidance for the search in this domain. Even though CPA can solve all problems with the exception of the $l09$ problem, it requires much more time than other planners in large instances and also returns significant longer plans. For example, in the `k10`-problem, CPA needs 401 seconds while CFF and POND only require around 24 seconds and t_0 needs less than 3 second; CPA's plan is also contains 20 or 30 actions more than those of POND or CFF and t_0. On the other hand, CPA can solve all problems but one while KACMBP can solve only three smaller problems.

Observe that CPA relies on the approximation defined by $T^{lp}(\mathcal{D})$ in its search for a solution. Therefore, it is, theoretically, a sound but incomplete planner because $T^{lp}(\mathcal{D})$ is deterministic whereas $\mathcal{D}$ can be nondeterministic as discussed Subsection 2.2. To make sure that our approach can cover a broader spectrum of planning problems, we also tested CPA with classical planning problems. The first domain considered was the Blocks World, and we performed tests with five problem instances described in [18]. We then tested with problems in the Rovers domain [16]. We experimented with five problem instances, different from each other in the numbers of way points, rovers, cameras, rock and soil samples, and objectives. It turns out that CPA can solve all those problems.

[16] http://planning.cis.strath.ac.uk/competition/

Table 7
UTS Domains

	KACMBP		CFF		POND		t_0		CPA	
Problem	PL	Time	PL	Time	PL	Time	PL	Time	PL	Time
k01	4	0.01	4	0.00	4	0.00	4	0.04	4	0.00
k02	11	0.20	10	0.00	12	0.01	11	0.06	12	0.03
k03	25	12.11	16	0.03	19	0.06		AB	20	0.28
k04		-	22	0.08	26	0.24	23	0.15	28	1.62
k05		-	28	0.33	33	0.67	29	0.25	40	7.69
k06		-	34	1.04	40	1.65	35	0.42	47	18.81
k07		-	40	2.57	47	3.55	41	0.73	57	51.52
k08		-	46	5.98	54	7.20	47	1.20	82	122.22
k09		-	52	12.09	61	14.02	53	1.91	81	217.95
k10		-	58	23.64	68	24.76	59	2.94	90	400.79
l01		-	4	0.01	4	0.00	4	0.05	4	0.00
l02		-	11	0.00	14	0.02	13	0.04	14	0.02
l03		-	17	0.01	23	0.08	23	0.07	30	0.36
l04		-	23	0.01	33	0.32	30	0.08	60	2.24
l05		-	29	0.33	46	1.11	47	0.14	53	9.32
l06		-	35	0.08		AB	62	0.22	89	24.75
l07		-	41	0.21	64	7.40	67	0.35	141	106.37
l08		-	47	0.45	70	16.99	64	0.70	213	456.24
l09		-	52	0.86	82	39.53	90	2.30		-
l10		-	58	1.60	88	100.00	85	4.12	194	1147.51
r03		-	17	0.02	17	0.08	19	0.07	18	0.23
r04		-	25	0.04	24	0.23	26	0.10	31	1.81
r05		-	33	1.47	32	0.65	34	0.20	50	9.94
r06		-	38	9.65	41	1.73	40	0.30	60	24.15
r07		-	47	1.60	53	4.14	44	0.59	105	98.41
r08		-	48	4.76	54	8.95	49	1.03	107	230.63
r09		-	62	3.82	64	17.44	57	1.46	146	480.28
r10		-	66	11.63	68	36.59	67	2.12	128	698.11

9 Conformant Planning —Logic Programming vs. Heuristic Search Through Approximation

The experimental results in the previous section indicate that the C++ planner CPA yields better performance than the logic programming based planner CPASP in most of the sequential benchmarks. However, CPASP is competitive for the planning problems with short solutions. Our experimental results also show that in problems with complex static causal laws, CPASP seems to do better than CPA and other state-of-the-art planners but it does not do well on large problems. One of the main reasons for this weakness of CPASP is its

reliance on a general answer set solver in computing a solution. On the other hand, the use of logic programming brings a number of advantages. We will now discuss some of these advantages in detail.

(1) *Generating Parallel Plans*: Logic programming based planners such as CPASP and DLV$^{\mathrm{K}}$ can generate parallel plans. In CPASP, this is achieved by the rules (20) and (21). On the other hand, most of the state-of-the-art planners do not compute parallel plans.

(2) *Incorporation of Control Knowledge and/or Preferences*: Logic programming provides an ideal environment for adding control knowledge and/or preferences in searching for plans satisfying some qualitative criteria. For example, in the bomb in the toilet domain with one package, flushing the toilet followed by dunking the package $(flush; dunk)$ is a plan achieving the goal of having the package disarmed, even when we know that the toilet is unclogged in the initial situation. Although it is a valid solution, this plan is hardly a preferable one. The problem is that the action theory does not include knowledge stipulating the planner not to consider such a plan when the plan with a single action *dunk* would be sufficient. This type of knowledge cannot be represented by an impossibility condition for $flush$. As such, for planners relying on a fixed representation language (e.g. PDDL), the incorporation of knowledge in planning requires a lot of work (e.g. TPlan [2] develops separate modules to deal with temporal knowledge). In logic programming, we can easily express the above knowledge by the constraint

$$\leftarrow occ(flush(t), T), h(\neg clogged(t), T).$$

which disallows answer sets in which the action $flush$ occurs when the toilet is *unclogged*. The paper [58] discusses in detail how different types of control knowledge can be easily incorporated in answer set planning.

(3) *Dealing with Complex Initial Situations*: Section 6 discusses how CPASP can be easily adapted to handle disjunctive information about the initial situations. This type of knowledge generates multiple branches in the search tree and can be dealt with fairly efficiently by state-of-the-art sequential planners. Yet, the specification of the initial situation in CPASP and its extension described in Section 6 remains simple in the sense that the initial state can be expressed by a set of facts. While this is adequate in all benchmarks found in the literature, the following example, discussed in [26], shows that allowing the specification of more complex initial situations is sometimes necessary and useful.

Example 9 *(Complex initial situation)* Assume that the packages in the bomb in the toilet domain are coming from different sources belonging to one of two hierarchically structured organizations, called b (bad) and g (good). The hierarchies are described in the usual way using relation $link(D_1, D_2)$ which indicates that a department D_1 is a subdivision of

a department D_2. Organization g for instance can be represented by a collection of atoms:

$$link(d_1, g).$$

$$link(d_2, g).$$

$$link(d_3, d_1).$$

$$link(d_4, d_1).$$

These atoms represent the fact that d_i $(i = 1, 2, 3, 4)$ are departments in the good company. It is known that packages coming from the organization g are safe and the bomb is sent by someone working for b. There are packages labeled by the name of the department the sender works for, which can be recorded by the atom $from(P, D)$ - package P came from department D. There are also some unlabeled packages. The initial situation of the modified bomb in the toilet problem will be described by the program H consisting of the above atoms and the following rules which define the organization the package came from and our trust in the good guys from g.

$$from(P, D_2) \leftarrow from(P, D_1), link(D_1, D_2)$$

$$h(\neg armed(P), 0) \leftarrow from(P, g)$$

As usual, P ranges over packages and D's range over the departments. It is easy to see that such a program has a unique answer set and can be used to specify the initial situation.

It is easy to see that for $\pi(\mathcal{D})$ (as well as $\Gamma(\mathcal{D})$) to work with the planning problems with this type of initial situation, we only need to replace the rule (18) by the above program.

10 Conclusion and Future Work

We define the notion of an approximation of action theories with static causal laws, concurrent actions, and incomplete information. The proposed approximation is deterministic and can be computed efficiently, using either logic programs or an imperative language (C++).

We develop two conformant planners using the proposed approximation, a logic programming based planner (called CPasp) and a heuristic search based planner (CpA). CPasp can generate parallel plans and CpA is a sequential planner. Both can handle disjunctive information about the initial state. Unlike many state-of-the-art conformant planners, these planners deal directly with static causal laws. Their performance is comparable with state-of-the-art

conformant planners over several benchmark domains as well as over newly invented domains. Due to the simple heuristic used in the implementation of CpA, we believe that the good performance of CpA lies in the use of the approximation. In a recent development, one of the authors has worked to improve this aspect of CpA and entered an improved version of CpA into the International Planning Competition. The improved version of CpA won the best price in the conformant planning category [17]. This result and the development of these planners demonstrates that a careful study in approximated reasoning may pay off well in the development of practical planners.

We also provide a sufficient condition for the completeness of the approximation. The condition is applicable to simple action theories whose static causal laws are of a special form. We do believe that this condition can be extended to cover a broader class of action theories. We leave this as one of our primary tasks in the near future. Furthermore, we would also like to investigate the use of more informative heuristics in improving the performance of CpA.

References

[1] F. Bacchus. The AIPS'00 Planning Competition. *AI Magazine*, 22(3), 2001. `http://www.cs.toronto.edu/aips2000/`.

[2] F. Bacchus and F. Kabanza. Using temporal logics to express search control knowledge for planning. *Artificial Intelligence*, 116(1,2):123–191, 2000.

[3] M. Balduccini and M. Gelfond. Diagnostic Reasoning with A-Prolog. *Theory and Practice of Logic Programming*, 3(4,5):425–461, 2003.

[4] M. Balduccini, M. Gelfond, and M. Nogueira. Answer Set Based Design of Knowledge Systems. *Annals of Mathematics and Artificial Intelligence*, 2006.

[5] C. Baral. Reasoning about Actions : Non-deterministic effects, Constraints and Qualification. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, pages 2017–2023. Morgan Kaufmann Publishers, San Francisco, CA, 1995.

[6] C. Baral, V. Kreinovich, and R. Trejo. Computational complexity of planning and approximate planning in the presence of incompleteness. *Artificial Intelligence*, 122:241–267, 2000.

[7] C. Baral, S. McIlraith, and T.C. Son. 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)*, pages 311–322, 2000.

[17] `http://ippc-2008.loria.fr/wiki/index.php/Results`

[8] C. Baral and M. Gelfond. Reasoning agents in dynamic domains. In J. Minker, editor, *Logic-Based Artificial Intelligence*, pages 257–279. Kluwer Academic Publishers, 2000.

[9] S. Baselice, P. A. Bonatti, and M. Gelfond. Towards an integration of answer set and constraint solving. In Maurizio Gabbrielli and Gopal Gupta, editors, *Logic Programming, 21st International Conference, ICLP 2005, Sitges, Spain, October 2-5, 2005, Proceedings*, volume 3668 of *Lecture Notes in Computer Science*, pages 52–66, 2005.

[10] A. Blum and M. Furst. Fast planning through planning graph analysis. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, pages 1636–1642. Morgan Kaufmann Publishers, San Francisco, CA, 95.

[11] B. Bonet and H. Geffner. GPT: a tool for planning with uncertainty and partial information. In A. Cimatti, H. Geffner, E. Giunchiglia, and J. Rintanen, editors, *Proc. IJCAI-01 Workshop on Planning with Uncertainty and Partial Information*, pages 82–87, Seattle, WA, 2001.

[12] R. Brafman and J. Hoffmann. Conformant planning via heuristic forward search: A new approach. In Sven Koenig, Shlomo Zilberstein, and Jana Koehler, editors, *Proceedings of the 14th International Conference on Automated Planning and Scheduling (ICAPS-04)*, pages 355–364, Whistler, Canada, 2004. Morgan Kaufmann.

[13] D. Bryce, S. Kambhampati, and D. Smith. Planning Graph Heuristics for Belief Space Search. *Journal of Artificial Intelligence Research*, 26:35–99, 2006.

[14] C. Castellini, E. Giunchiglia, and A. Tacchella. SAT-based Planning in Complex Domains: Concurrency, Constraints and Nondeterminism. *Artificial Intelligence*, 147:85–117, July 2003.

[15] A. Cimatti and M. Roveri. Conformant Planning via Symbolic Model Checking. *Journal of Artificial Intelligence Research*, 13:305–338, 2000.

[16] A. Cimatti, M. Roveri, and P. Bertoli. Conformant Planning via Symbolic Model Checking and Heuristic Search. *Artificial Intelligence Journal*, 159:127–206, 2004.

[17] S. Edelkamp, J. Hoffmann, M. Littman, and H. Younes. The IPC-2004 Planning Competition, 2004. `http://ls5-www.cs.uni-dortmund.de/~edelkamp/ipc-4/`.

[18] T. Eiter, W. Faber, N. Leone, G. Pfeifer, and A. Polleres. A Logic Programming Approach to Knowledge State Planning, II: The DLV^K System. *Artificial Intelligence*, 144(1-2):157–211, 2003.

[19] T. Eiter, N. Leone, C. Mateis, G. Pfeifer, and F. Scarcello. The KR System `dlv`: Progress Report, Comparisons, and Benchmarks. In *International Conference on Principles of Knowledge Representation and Reasoning*, pages 406–417, 1998.

[20] I. Elkabani, E. Pontelli, and T. C. Son. Smodels with CLP and Its Applications: A Simple and Effective Approach to Aggregates in ASP. In Bart Demoen and Vladimir Lifschitz, editors, *Logic Programming, 20th International Conference, ICLP 2004, Saint-Malo, France, September 6-10, 2004, Proceedings*, volume 3132 of *Lecture Notes in Computer Science*, pages 73–89. Springer, 2004.

[21] P. Ferraris and E. Giunchiglia. Planning as satisfiability in nondeterministic domains. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on on Innovative Applications of Artificial Intelligence, July 30 - August 3, 2000, Austin, Texas, USA*, pages 748–753. AAAI Press / The MIT Press, 2000.

[22] M. Gelfond and A. Gabaldon. From functional specifications to logic programs. In J. Maluszynski, editor, *Proceedings of International symposium on logic programming*, pages 355–370, 1997.

[23] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In R. Kowalski and K. Bowen, editors, *Logic Programming: Proceedings of the Fifth International Conf. and Symp.*, pages 1070–1080, 1988.

[24] M. Gelfond and V. Lifschitz. Representing actions and change by logic programs. *Journal of Logic Programming*, 17(2,3,4):301–323, 1993.

[25] M. Gelfond and V. Lifschitz. Action languages. *ETAI*, 3(6), 1998.

[26] M. Gelfond and R. Morales. Encoding conformant planning in a-prolog. In *Proceedings of DRT'04*, Lecture Notes in Computer Science. Springer, 2004.

[27] M. Ghallab, A. Howe, C. Knoblock, D. McDermott, A. Ram, M. Veloso, D. Weld, and D. Wilkins. PDDL — the Planning Domain Definition Language. Version 1.2. Technical Report CVC TR98003/DCS TR1165, Yale Center for Comp, Vis and Ctrl, 1998.

[28] E. Giunchiglia, G. Kartha, and V. Lifschitz. Representing action: indeterminacy and ramifications. *Artificial Intelligence*, 95:409–443, 1997.

[29] E. Giunchiglia and V. Lifschitz. An action language based on causal explanation: preliminary report. In *Proceedings of AAAI 98*, pages 623–630, 98.

[30] S. Hanks and D. V. McDermott. Default reasoning, nonmonotonic logics, and the frame problem. In *Proceedings of the 5th National Conference on Artificial Intelligence. Philadelphia, PA*, pages 328–333. Morgan Kaufmann, 1986.

[31] J. Hoffmann and B. Nebel. The FF Planning System: Fast Plan Generation Through Heuristic Search. *Journal of Artificial Intelligence Research*, 14:253–302, 2001.

[32] G. Kartha and V. Lifschitz. Actions with indirect effects: Preliminary report. In *KR 94*, pages 341–350, 1994.

[33] Y. Lierler and M. Maratea. Cmodels-2: SAT-based Answer Set Solver Enhanced to Non-tight Programs. In Vladimir Lifschitz and Ilkka Niemelä, editors, *Proceedings of the 7th International Conference on Logic Programming and NonMonotonic Reasoning Conference (LPNMR'04)*, volume 2923, pages 346–350. Springer Verlag, LNCS 2923, 2004.

[34] V. Lifschitz, editor. *Formalizing Common Sense: Papers by John McCarthy.* Ablex Publishing Corporation, 1989.

[35] V. Lifschitz. Answer set programming and plan generation. *Artificial Intelligence*, 138(1–2):39–54, 2002.

[36] V. Lifschitz and H. Turner. Splitting a logic program. In Pascal Van Hentenryck, editor, *Proceedings of the Eleventh International Conf. on Logic Programming*, pages 23–38, 1994.

[37] V. Lifschitz. On the Logic of Causal Explanation (Research Note). *Artif. Intell.*, 96(2):451–465, 1997.

[38] F. Lin. Embracing causality in specifying the indirect effects of actions. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, pages 1985–1993. Morgan Kaufmann Publishers, San Mateo, CA, 95.

[39] Y. Liu and H. Levesque. Tractable reasoning with incomplete first-order knowledge in dynamic systems with context-dependent actions. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence, Edinburgh, Scotland*, IJCAI, 2005.

[40] D. Long and M. Fox. The 3rd International Planning Competition: Results and Analysis. *Journal of Artificial Intelligence Research (JAIR)*, 20:1–59, 2003. `http://planning.cis.strath.ac.uk/competition/`.

[41] N. McCain and H. Turner. A causal theory of ramifications and qualifications. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, pages 1978–1984. Morgan Kaufmann Publishers, San Mateo, CA, 1995.

[42] J. McCarthy. Programs with common sense. In *Proceedings of the Teddington Conference on the Mechanization of Thought Processes*, pages 75–91, London, 1959. Her Majesty's Stationery Office.

[43] J. McCarthy. Elaboration tolerance. In *Proceedings of the 98's Common Sense Workshop*, 1998. `http://www-formal.stanford.edu/jmc/elaboration.html`.

[44] J. McCarthy and P. Hayes. Some philosophical problems from the standpoint of artificial intelligence. In B. Meltzer and D. Michie, editors, *Machine Intelligence*, volume 4, pages 463–502. Edinburgh University Press, Edinburgh, 1969.

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

[46] S. McIlraith. Intergrating actions and state constraints: A closed-form solution to the ramification problem (sometimes). *Artificial Intelligence*, 116:87–121, 2000.

[47] A. R. Morales, P. H. Tu, and T. C. Son. An extension to conformant planning using logic programming. In Manuela M. Veloso, editor, *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007*, pages 1991–1996, 2007.

[48] H. Palacios and H. Geffner. Compiling Uncertainty Away: Solving Conformant Planning Problems Using a Classical Planner (Sometimes). In *Proceedings of the the Twenty-First National Conference on Artificial Intelligence*, 2006.

[49] H. Palacios and H. Geffner. From Conformant into Classical Planning: Efficient Translations that may be Complete Too. In *Proceedings of the 17th International Conference on Planning and Scheduling*, 2007.

[50] R. P. A. Petrick and F. Bacchus. A knowledge-based approach to planning with incomplete information and sensing. In *Proceedings of the Sixth International Conference on Artificial Intelligence Planning Systems, April 23-27, 2002, Toulouse, France*, pages 212–222. AAAI, 2002.

[51] R. P. A. Petrick and F. Bacchus. Extending the knowledge-based approach to planning with incomplete information and sensing. In *Proceedings of the Sixth International Conference on Automated Planning and Scheduling, 2004*, pages 2–11, 2004.

[52] J. Rintanen. Asymptotically optimal encodings of conformant planning in qbf. In *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence, July 22-26, 2007, Vancouver, British Columbia, Canada*, pages 1045–1050. AAAI Press, 2007.

[53] E. Sandewall. Assessments of ramification methods that use static domain constraints. In L. C. Aiello, J. Doyle, and S. Shapiro, editors, *Proceedings of KRR*, pages 89–110. Morgan Kaufmann, 1996.

[54] M. Shanahan. The ramification problem in the event calculus. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, IJCAI 99, Stockholm, Sweden, July 31 - August 6, 1999. 2 Volumes, 1450 pages*, pages 140–146, 1999.

[55] P. Simons, N. Niemelä, and T. Soininen. Extending and Implementing the Stable Model Semantics. *Artificial Intelligence*, 138(1–2):181–234, 2002.

[56] D.E. Smith and D.S. Weld. Conformant graphplan. In *AAAI*, pages 889–896, 1998.

[57] T.C. Son and C. Baral. Formalizing sensing actions - a transition function based approach. *Artificial Intelligence*, 125(1-2):19–91, January 2001.

[58] T. C. Son, C. Baral, N. Tran, and S. McIlraith. Domain-Dependent Knowledge in Answer Set Planning. *ACM Transactions on Computational Logic*, 7(4):613–657, 2006.

[59] T. C. Son and E. Pontelli. Some Results on The Completeness of Approximation Based Reasoning. In *Proceedings of the the Tenth Pacific Rim International Conference on Artificial Intelligence*, pages 358–369, 2008.

[60] T. C. Son and P. H. Tu. On the Completeness of Approximation Based Reasoning and Planning in Action Theories with Incomplete Information. In *Proceedings of the 10th International Conference on Principles of Knowledge Representation and Reasoning*, pages 481–491, 2006.

[61] T. C. Son, P. H. Tu, M. Gelfond, and R. Morales. An Approximation of Action Theories of AL and its Application to Conformant Planning. In *Proceedings of the the 7th International Conference on Logic Programming and NonMonotonic Reasoning*, pages 172–184, 2005.

[62] T. C. Son, P. H. Tu, M. Gelfond, and R. Morales. Conformant Planning for Domains with Constraints — A New Approach. In *Proceedings of the the Twentieth National Conference on Artificial Intelligence*, pages 1211–1216, 2005.

[63] S. Thiebaux, J. Hoffmann, and B. Nebel. In Defense of PDDL Axioms. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI'03)*, 2003.

[64] M. Thielscher. Ramification and causality. *Artificial Intelligence*, 89(1-2):317–364, 1997.

[65] P. H. Tu. *Reasoning AND Planning With Incomplete Information In The Presence OF Static Causal Laws*. PhD thesis, New Mexico State University, 2007.

[66] P. H. Tu, T. C. Son, and C. Baral. Reasoning and Planning with Sensing Actions, Incomplete Information, and Static Causal Laws using Logic Programming. *Theory and Practice of Logic Programming*, 7:1–74, 2006.

[67] H. Turner. Representing actions in logic programs and default theories. *Journal of Logic Programming*, 31(1-3):245–298, May 1997.

[68] H. Turner. Polynomial-length planning spans the polynomial hierarchy. In *Proc. of Eighth European Conf. on Logics in Artificial Intelligence (JELIA'02)*, pages 111–124, 2002.

A Proofs of Proposition 1 and Theorems 3 & 5

Suppose an action theory $\mathcal{D}$ is given. Let s be a state, δ be a partial state and a be an action. From Theorem 2, we have the following result.

Lemma 1 *Let A be an answer set of $\Phi(a, s)$ and let $s' = \{l \mid h(l, 1) \in A\}$. Then s' is a state and $\langle s, a, s' \rangle \in T(\mathcal{D})$.*

Since this appendix mainly deals with programs $\Phi(a, s)$ and $\Pi(a, \delta)$ (defined by (17)), to simplify the proofs, we remove from the programs atoms and rules that are of no interest.

First, let $\Phi_0(a, s)$ (resp. $\Pi_0(a, \delta)$) denote the program obtained from $\Phi(a, s)$ (resp. $\Pi(a, \delta)$) by removing its constraints.

Lemma 2 *$\Pi_0(a, \delta)$ is a stratified program.*

Proof. We need to find a function λ which maps atoms in $\Pi_0(a, \delta)$ into non-negative integers such that for every r in $\Pi_0(a, \delta)$ with the head y,

- $\lambda(y) \geq \lambda(x)$ for every atom x such that x appears in the body of r; and
- $\lambda(y) > \lambda(x)$ for every atom x such that *not* x appears in the body of r.

It is easy to check that the following function λ satisfies the above property.

- $\lambda(de(l, 1)) = 1$;
- $\lambda(ph(l, 1)) = 2$;
- $\lambda(h(l, 1)) = 3$; and
- $\lambda(at) = 0$ for any other atom at, e.g., $\lambda(h(l, 0)) = 0$ for every fluent literal l.

□

Lemma 3 *The program $\Pi_0(a, \delta)$ is consistent and has a unique answer set.*

Proof. It follows from Lemma 2 and [23]. □

Let X be the set of atoms of the forms $fluent(F)$ and $literal(L)$. Clearly X is a splitting set [36] of both $\Phi_0(a, s)$ and $\Pi_0(a, \delta)$. It is easy to see that the bottom parts of $\Phi_0(a, s)$ and $\Pi_0(a, \delta)$ with respect to X are positive programs and have only one answer set

$$U = \{fluent(F), literal(F), literal(neg(F)) \mid F \in \mathbf{F}\} \tag{A.1}$$

Let $\Phi_1(a, s)$ and $\Pi_1(a, \delta)$ denote the evaluation of the top parts of $\Phi_0(a, s)$ and $\Pi_0(a, \delta)$ with respect to U. The rules of these programs are listed below (the condition for each rule follows the rule).

- $\Phi_1(a, s)$ contains the following rules:

$$h(l,1) \leftarrow o(e,0), h(\psi,0) \quad ([e \textbf{ causes } l \textbf{ if } \psi] \in \mathcal{D}) \tag{A.2}$$
$$h(l,0) \leftarrow h(\psi,0) \quad ([l \textbf{ if } \psi] \in \mathcal{D}) \tag{A.3}$$
$$h(l,1) \leftarrow h(\psi,1) \quad ([l \textbf{ if } \psi] \in \mathcal{D}) \tag{A.4}$$
$$h(L,1) \leftarrow h(L,0), not\ h(\neg L,1) \tag{A.5}$$
$$h(s,0) \leftarrow \tag{A.6}$$
$$h(a,0) \leftarrow \tag{A.7}$$

- $\Pi_1(a,\delta)$ contains the following rules:

$$h(l,1) \leftarrow o(e,0), h(\psi,0) \quad ([e \textbf{ causes } l \textbf{ if } \psi] \in \mathcal{D}) \tag{A.8}$$
$$de(l,1) \leftarrow o(e,0), h(\psi,0) \quad ([e \textbf{ causes } l \textbf{ if } \psi] \in \mathcal{D}) \tag{A.9}$$
$$ph(l,1) \leftarrow o(e,0), not\ h(\neg\psi,0), not\ de(\neg l,1) \quad ([e \textbf{ causes } l \textbf{ if } \psi] \in \mathcal{D}) \tag{A.10}$$
$$ph(L,1) \leftarrow not\ h(\neg L,0), not\ de(\neg L,1) \tag{A.11}$$
$$h(l,0) \leftarrow h(\psi,0) \quad ([l \textbf{ if } \psi] \in \mathcal{D}) \tag{A.12}$$
$$h(l,1) \leftarrow h(\psi,1) \quad ([l \textbf{ if } \psi] \in \mathcal{D}) \tag{A.13}$$
$$ph(l,1) \leftarrow ph(\psi,1) \quad ([l \textbf{ if } \psi] \in \mathcal{D}) \tag{A.14}$$
$$h(L,1) \leftarrow not\ ph(\neg L,1) \tag{A.15}$$
$$h(\delta,0) \leftarrow \tag{A.16}$$
$$h(a,0) \leftarrow \tag{A.17}$$

Let Y be the set of atoms of the form $o(e,0)$, $h(l,0)$, or $ph(l,0)$. Then it is easy to see that Y is a splitting set of both $\Phi_1(a,s)$ and $\Pi_1(a,\delta)$.

Because s is a state, the bottom part of $\Phi_1(a,s)$ with respect to Y has the unique answer set

$$V = o(a,0) \cup h(s,0) \tag{A.18}$$

Hence, the evaluation of the top part of $\Phi_1(a,s)$ with respect to V is the following set of rules

$$h(l,1) \leftarrow \quad (l \text{ is a direct effect of } a \text{ in } s) \tag{A.19}$$

$$h(l,1) \leftarrow h(\psi,1) \quad ([l \textbf{ if } \psi] \in \mathcal{D}) \tag{A.20}$$

$$h(L,1) \leftarrow not\ h(\neg L,1) \quad (L \text{ holds in } s) \tag{A.21}$$

Let $\Phi_2(a,s)$ denote this program.

Lemma 4 *A is an answer set of $\Phi(a,s)$ iff there exists an answer set A_1 of $\Phi_2(a,s)$ such that*

$$A = U \cup V \cup A_1$$

where U and V are defined by (A.1) and (A.18). Furthermore, we have

$$h(l,1) \in A \text{ iff } h(l,1) \in A_1$$

Proof. By the splitting set theorem [36]. □

Similarly, because δ is a partial state, the bottom part of $\Pi_1(a,\delta)$ has the unique answer set

$$W = o(a,0) \cup h(\delta,0) \tag{A.22}$$

The evaluation of the top part of $\Pi_1(a,\delta)$ with respect to W is the following set of rules

$$h(l,1) \leftarrow \quad (l \text{ is a direct effect of } a \text{ in } \delta) \tag{A.23}$$

$$de(l,1) \leftarrow \quad (l \text{ is a direct effect of } a \text{ in } \delta) \tag{A.24}$$

$$ph(l,1) \leftarrow not\ de(\neg l,1) \quad (l \text{ is a possible direct effect of } a \text{ in } \delta) \tag{A.25}$$

$$ph(L,1) \leftarrow not\ de(\neg L,1) \quad (L \text{ possibly holds in } \delta) \tag{A.26}$$

$$h(l,1) \leftarrow h(\psi,1) \quad ([l \textbf{ if } \psi] \in \mathcal{D}) \tag{A.27}$$

$$ph(l,1) \leftarrow ph(\psi,1) \quad ([l \textbf{ if } \psi] \in \mathcal{D}) \tag{A.28}$$

$$h(L,1) \leftarrow not\ ph(\neg L,1) \tag{A.29}$$

Let us denote this program by $\Pi_2(a,\delta)$.

Lemma 5 *B is an answer set of $\Pi_0(a,\delta)$ iff there exists an answer set B_0 of*

$\Pi_2(a,\delta)$ *such that*

$$B = U \cup W \cup B_0 \tag{A.30}$$

where U *and* W *are defined by (A.1) and (A.22). Furthermore, we have*

$$h(l,1) \in B \text{ iff } h(l,1) \in B_0 \tag{A.31}$$

Proof. It follows from the splitting set theorem. □

Let us further split the program $\Pi_2(a,\delta)$ by using the splitting set consisting of atoms of the form $de(l,1)$. The bottom part of $\Pi_2(a,\delta)$ contains only rules of the form (A.24) and thus it has the only answer set

$$\{de(l,1) \mid l \text{ is a direct effect of } a\}$$

Hence, the evaluation of the top part of $\Pi_2(a,\delta)$ with respect to this answer set, denoted by $\Pi_3(a,\delta)$, contains the following rules:

$$h(l,1) \leftarrow \tag{A.32}$$
$$(l \text{ is a direct effect of } a \text{ in } \delta)$$
$$ph(l,1) \leftarrow \tag{A.33}$$
$$(l \text{ is a possible direct effect of } a \text{ in } \delta, \neg l \text{ is not a direct effect of } a)$$
$$ph(L,1) \leftarrow \tag{A.34}$$
$$(L \text{ possibly holds in } \delta, \neg L \text{ is not a direct effect of } a) \tag{A.35}$$
$$h(l,1) \leftarrow h(\psi,1) \tag{A.36}$$
$$([l \textbf{ if } \psi] \in \mathcal{D})$$
$$ph(l,1) \leftarrow ph(\psi,1) \tag{A.37}$$
$$([l \textbf{ if } \psi] \in \mathcal{D})$$
$$h(L,1) \leftarrow not\ ph(\neg L,1) \tag{A.38}$$

Lemma 6 *B is an answer set of $\Pi_0(a,\delta)$ iff there exists an answer set B_1 of $\Pi_3(a,\delta)$ such that*

$$B = U \cup W \cup \{de(l,1) \mid l \text{ is a direct effect of } a \text{ in } \delta\} \cup B_1 \tag{A.39}$$

Furthermore, we have

$$h(l,1) \in B \text{ iff } h(l,1) \in B_1 \tag{A.40}$$

Proof. It follows from Lemma 5 and the splitting set theorem. □

For a set of atoms X, let $\Phi_2^X(a,s)$ (resp. $\Pi_3^X(a,\delta)$) denote the reduct of $\Phi_2(a,s)$ (resp. $\Pi_3(a,\delta)$) with respect to X. That is, $\Phi_2^X(a,s)$ is the following set of rules:

$$h(l,1) \leftarrow \quad (l \text{ is a direct effect of } a \text{ in } s) \tag{A.41}$$

$$h(l,1) \leftarrow h(\psi,1) \quad ([l \textbf{ if } \psi] \in \mathcal{D}) \tag{A.42}$$

$$h(L,1) \leftarrow \quad (L \text{ holds in } s, h(\neg L,1) \notin X) \tag{A.43}$$

and $\Pi_3^X(a,\delta)$ is the following set of rules:

$$h(l,1) \leftarrow \quad (l \text{ is a direct effect of } a \text{ in } \delta) \tag{A.44}$$

$$ph(l,1) \leftarrow \quad (l \text{ is a possible direct effect of } a \text{ in } \delta, \neg l \text{ is not a direct effect of } a) \tag{A.45}$$

$$ph(L,1) \leftarrow \quad (L \text{ possibly holds in } \delta, \neg L \text{ is not a direct effect of } a) \tag{A.46}$$

$$h(l,1) \leftarrow h(\psi,1) \quad ([l \textbf{ if } \psi] \in \mathcal{D}) \tag{A.47}$$

$$ph(l,1) \leftarrow ph(\psi,1) \quad ([l \textbf{ if } \psi] \in \mathcal{D}) \tag{A.48}$$

$$h(L,1) \leftarrow \quad (ph(\neg L,1) \notin X) \tag{A.49}$$

Let us prove the following lemma

Lemma 7 *Let A_1 be an answer set of $\Phi_2(a,s)$ and B_1 be an answer set of $\Pi_3(a,\delta)$. If $\delta \subseteq s$ then for any fluent literal l, $ph(l,1) \in B_1$ implies that $h(l,1) \in A_1$.*

Proof. Suppose $\delta \subseteq s$. Let $P = \Phi_2^{A_1}(a,s)$ and $Q = \Pi_3^{B_1}(a,\delta)$. Because A_1 and B_1 are answer sets of $\Phi_2(a,s)$ and $\Pi_3(a,\delta)$ respectively, we have

$$A_1 = \bigcup_i T_P^i(\emptyset) \tag{A.50}$$

$$B_1 = \bigcup_i T_Q^i(\emptyset) \tag{A.51}$$

where T_P and T_Q are the immediate consequence operators of programs P and Q respectively.

First of all, we show that for any integer $i \geq 0$ the following result holds

$$h(l,1) \in T_P^i(\emptyset) \Rightarrow ph(l,1) \in T_Q^i(\emptyset) \tag{A.52}$$

Let us prove by induction on i.

(1) Base case: $i = 0$. (A.52) holds because $T_P^0(\emptyset) = T_Q^0(\emptyset) = \emptyset$.
(2) Inductive step: suppose (A.52) holds for $i = k$, we need to show that it also holds for $i = k + 1$.

Let l be a fluent literal such that $h(l, 1) \in T_P^{k+1}(\emptyset)$. By the definition of $T_P^{k+1}(\emptyset)$, there are three cases

(a) $h(l, 1)$ holds in $T_P^{k+1}(\emptyset)$ by rule (A.41). This means that l is a direct effect of a in s. Observe that a direct effect of a in s is always a possible direct effect of a in δ. Furthermore, because $\mathcal{D}$ is consistent, $\neg l$ is not a direct effect of a in s. Hence, $\neg l$ is not a direct effect of a in δ. As a result, rule (A.45) belongs to Q, which implies that $ph(l, 1) \in T_Q^{k+1}(\emptyset)$.

(b) $h(l, 1)$ holds in $T_P^{k+1}(\emptyset)$ by rule (A.42). This means that there exists a static causal law (2) such that

$$h(\psi, 1) \subseteq T_P^k(\emptyset)$$

By the inductive hypothesis, we have

$$ph(\psi, 1) \subseteq T_Q^k(\emptyset)$$

By rule (A.48), this implies that $ph(l, 1) \in T_Q^{k+1}(\emptyset)$.

(c) $h(l, 1)$ holds in $T_P^{k+1}(\emptyset)$ by rule (A.43). This means that l holds in s and $h(\neg l, 1) \notin A_1$.

Because l holds in s, l possibly holds in δ. On the other hand, $h(\neg l, 1) \notin A_1$ implies that $\neg l$ is not a direct effect of a in s (because of rule (A.41)). Hence, $\neg l$ is not a direct effect of a in δ. Accordingly, Q contains the rule of the form (A.46) with $L = l$. Thus, it follows that $ph(l, 1) \in T_Q^{k+1}(\emptyset)$.

As a result, (A.52) holds. The lemma directly follows from (A.52), (A.50), and (A.51). □

Lemma 8 *Let A be an answer set of $\Phi(a, s)$ and B be an answer set of $\Pi_0(a, \delta)$. If $\delta \subseteq s$ then for any fluent literal l, $h(l, 1) \in B$ implies $h(l, 1) \in A$.*

Proof. By Lemmas 4 and 6, the programs $\Phi_2(a, s)$ and $\Pi_3(a, s)$ have answer sets A_1 and B_1, respectively, such that

$$A = U \cup V \cup A_1$$

and

$$B = U \cup W \cup \{de(l, 1) \mid l \text{ is a direct effect of } a \text{ in } \delta\} \cup B_1$$

Let P and Q be programs defined in Lemma 7. First of all, we will show that

$$h(l,1) \in T_Q^i(\emptyset) \Rightarrow h(l,1) \in A_1 \tag{A.53}$$

for $i \geq 0$ by using induction on i.

(1) Base case: trivial because there exist no fluent literal l such that $l \in T_Q^i(\emptyset) = \emptyset$.
(2) Inductive step: suppose (A.53) holds for $i \leq k$.
Let l be a fluent literal such that $h(l,1) \in T_Q^{k+1}(\emptyset)$. We need to show that $h(l,1) \in A_1$. Consider the following cases
(a) $h(l,1) \in T_Q^{k+1}(\emptyset)$ by rule (A.44). This means that l is a direct effect of a in δ. On the other hand, a direct effect of a in δ is also a direct effect of a in s. As a result, l is a direct effect of a in s. By rule (A.19), this implies that $h(l,1) \in A_1$.
(b) $h(l,1) \in T_Q^{k+1}(\emptyset)$ by rule (A.47). This implies that $h(\psi,1) \subseteq T_Q^k(\emptyset)$. By the inductive hypothesis, we have $h(\psi,1) \subseteq A_1$. As a result, by rule (A.20), it follows that $h(l,1) \in A_1$.
(c) $h(l,1) \in T_Q^{k+1}(\emptyset)$ by rule (A.49). This implies that $ph(\neg l,1) \notin B_1$. It follows from Lemma 7 that $h(\neg l,1) \notin A_1$, i.e.,

$$h(\neg l,1) \notin A \tag{A.54}$$

Let $s' = \{g \mid h(g,1) \in A\}$. From (A.54), we have $\neg l \notin s'$. On the other hand, by Lemma 1, s' is a state, which implies that either l or $\neg l$ belongs to s'. Accordingly, we have $l \in s'$. From this, it follows that $h(l,1) \in A$ and thus $h(l,1) \in A_1$.

From (A.53) and (A.51), we have

$$h(l,1) \in B_1 \Rightarrow h(l,1) \in A_1$$

On the other hand, by Lemmas 4 and 6, we have

$$h(l,1) \in A_1 \text{ iff } h(l,1) \in A$$

and

$$h(l,1) \in B_1 \text{ iff } h(l,1) \in B$$

Consequently, we can conclude that the lemma holds.

□

Lemma 9 *$\Pi(a,\delta)$ is consistent iff a is safe in δ.*

Proof. Suppose $\Pi(a,\delta)$ is consistent. Let B denote the answer set of $\Pi(a,\delta)$. According to the definition of an answer set of a program with constraints, B is an answer set of $\Pi_0(a,\delta)$ and B does not violate constraints (6) and (8).

This implies that there exists no impossibility condition

$$\textbf{impossible } b \textbf{ if } \psi$$

in $\mathcal{D}$ such that $o(b,0) \subseteq B$ and $h(\neg\psi,0) \cap B = \emptyset$. Because $o(a,0)$ is the set of all atoms of the form $o(e,0)$, where e is an elementary action, contained in B (see Lemma 5), there exists no impossibility condition

$$\textbf{impossible } b \textbf{ if } \psi$$

such that $b \subseteq a$ and ψ possibly holds in δ. By definition, this means a is safe in δ.

Now suppose that a is safe in δ. By Lemma 3, $\Pi_0(a,\delta)$ has a unique answer set B. We will show that B is also an answer set of $\Pi(a,\delta)$ by showing that it satisfies constraints (6) and (8):

(1) Constraint (6): Trivial because a is safe in δ.
(2) Constraint (8): Since δ is a partial state, there exists a state s such that $\delta \subseteq s$. As a is safe in δ, it is executable in s. By Theorem 3, it follows that the program $\Phi(a,s)$ has an answer set A and this answer set satisfies constraint (8). By Lemma 8, it follows that B also satisfies constraint (8).

□

Lemma 10 *If $\Pi(a,\delta)$ is consistent then the only answer set of $\Pi(a,\delta)$ is the answer set of $\Pi_0(a,\delta)$.*

Proof. The lemma follows from Lemma 3 and from the fact that $\Pi(a,\delta)$ differs from the program $\Pi_0(a,\delta)$ in two constraints (6) and (8) only. □

A.1 Proof of Proposition 1

Suppose $\Pi(a,\delta)$ is consistent. Lemma 10 implies that the only answer set of $\Pi(a,\delta)$ is the answer set of $\Pi_0(a,\delta)$. Let

$$\delta' = \{l \mid h(l,1) \in B\}$$

To complete the proof, we need to show that δ' is a partial state. First of all, observe that δ' satisfies all the static causal laws of $\mathcal{D}$ because of constraint (5). So, we only need to show that there exists a state s' such that $\delta' \subseteq s'$.

Since δ is a partial state there exists a state s such that $\delta \subseteq s$. Because $\Pi(a,\delta)$ is consistent, by Lemma 9, a is safe in δ. Thus, it is executable in s. Since we assume $\mathcal{D}$ is consistent, there must be a state s' such that $\langle s,a,s'\rangle \in T(\mathcal{D})$.

By Theorem 2, this implies that the program $\Phi(a, s)$ has an answer set A such that $s' = \{l \mid h(l, 1) \in A\}$. By Lemma 8, it is easy to see that $\delta' \subseteq s'$.

A.2 Proof of Theorem 3

Let $\langle \delta, a, \delta' \rangle$ be a transition in $T^{lp}(\mathcal{D})$. It follows from Definition 8 that the program $\Pi(a, \delta)$ is consistent and has an answer set B such that $\delta' = \{l \mid h(l, 1) \in B\}$. Note that by Lemma 10, such an answer set B is unique and it is also the answer set of $\Pi_0(a, \delta)$.

First, let us show that $T^{lp}(\mathcal{D})$ is an approximation of $T(\mathcal{D})$. Clearly, to prove that, it suffices to show that for every $s \in comp(\delta)$,

(1) a is executable in s
(2) for every state s' such that $\langle s, a, s' \rangle \in T(\mathcal{D})$, $\delta' \subseteq s'$.

Consider a state $s \in comp(\delta)$. By Lemma 9, a is safe in δ. Because $\delta \subseteq s$, a is executable in s. Now let s' be a state such that $\langle s, a, s' \rangle \in T(\mathcal{D})$. By Theorem 2, this implies that there exists an answer set A of $\Phi(a, s)$ such that $s' = \{l \mid h(l, 1) \in A\}$. By Lemma 8, we have $\delta' = \{l \mid h(l, 1) \in B\} \subseteq \{l \mid h(l, 1) \in A\} = s'$.

We have showed that $T^{lp}(\mathcal{D})$ is an approximation of $T(\mathcal{D})$. The determinism of $T^{lp}(\mathcal{D})$ follows directly from the fact that B is unique.

B Proof of Theorem 5

This appendix contains the proof of Theorem 5. We assume that a planning problem $\mathcal{P} = \langle \delta^0, \mathcal{D}, \delta^f \rangle$ is given. For the sake of simplicity of the proof, similarly to the previous section, we will begin with a simplification of the program $\pi(\mathcal{P}, n)$.

Let $\pi_0(\mathcal{P}, n)$ be the program obtained from $\pi(\mathcal{P}, n)$ by removing constraints (6) and (8). Let X be the set of atoms of the forms $fluent(F)$ and $literal(L)$. Then, X is a splitting set of $\pi_0(\mathcal{P}, n)$. The bottom part of $\pi_0(\mathcal{P}, n)$ is a positive program and has a unique answer set U defined by (A.1). Let $\pi_1(\mathcal{P}, n)$ denote the evaluation of the top part of $\pi_0(\mathcal{P}, n)$ with respect to U.

Lemma 11 *A set of atoms C is an answer set of $\pi_0(\mathcal{P}, n)$ iff $C = C_1 \cup U$ where C_1 is an answer set of $\pi_1(\mathcal{P}, n)$.*

Proof. Follows from the splitting set theorem. □

For an integer $0 \leq i \leq n$, let X_i denote the set of atoms whose time parameters are less than or equal to i. Then, it is easy to see that the sequence $\langle X_i \rangle_{i=0}^{n}$ is a splitting sequence [36] of $\pi_1(\mathcal{P}, n)$.

Lemma 12 *A set of atoms C_1 is an answer set of $\pi_1(\mathcal{P}, n)$ iff there is a sequence of sets of atoms $\langle D_i \rangle_{i=0}^{n}$ such that the following conditions are satisfied.*

(1) D_0 is an answer set of

$$\mu_0 = b_{X_0}(\pi_1(\mathcal{P}, n)) \tag{B.1}$$

(2) For every $1 \leq i \leq n$, D_i is an answer set of

$$\mu_i = e_{X_i}(b_{X_i}(\pi_1(\mathcal{P}, n)) \setminus b_{X_{i-1}}(\pi_1(\mathcal{P}, n)), \bigcup_{1 \leq j \leq i-1} D_j) \tag{B.2}$$

(3)

$$C_1 = \bigcup_{i=0}^{n} D_i \tag{B.3}$$

where $b_X(P)$ denote the bottom part of a program P with respect to X and $e_X(Q, V)$ denote the evaluation of a program Q relative to V.

Proof. Follows from the splitting sequence theorem [36]. □

Now suppose that C is an answer set of $\pi(\mathcal{P}, n)$. By definition C is also an answer set of $\pi_0(\mathcal{P}, n)$ and C does not violate any constraint of $\pi(\mathcal{P}, n)$. By Lemma 11, the program $\pi_1(\mathcal{P}, n)$ has an answer set C_1 such that $C = C_1 \cup U$. By Lemma 12, it follows that there exists a sequence of sets of atoms $\langle D_i \rangle_{i=0}^{n}$ that satisfies (B.1)–(B.3). Let $\delta_i = \{l \mid h(l, i) \in D_i\}$ and let $a_i = \{l \mid o(e, i) \in D_i\}$. It is easy to see that μ_0 is the following set of rules

$$h(l, 0) \leftarrow h(\psi, 0) \quad ([l \textbf{ if } \psi] \in \mathcal{D}) \tag{B.4}$$

$$h(\delta^0, 0) \leftarrow \tag{B.5}$$

$$o(E, 0) \vee \neg o(E, 0) \leftarrow \tag{B.6}$$

and for $i \geq 1$, μ_i is the following set of rules

$$h(l,i) \leftarrow \quad (l \text{ is a direct effect of } a_{i-1} \text{ in } \delta_{i-1}) \tag{B.7}$$

$$de(l,i) \leftarrow \quad (l \text{ is a direct effect of } a_{i-1} \text{ in } \delta_{i-1}) \tag{B.8}$$

$$ph(l,i) \leftarrow not\ de(\neg l,i) \quad (l \text{ is a possible direct effect of } a_{i-1} \text{ in } \delta_{i-1}) \tag{B.9}$$

$$ph(L,i) \leftarrow not\ de(\neg L,i) \quad (h(\neg L, i-1) \notin D_{i-1}) \tag{B.10}$$

$$h(l,i) \leftarrow h(\psi,i) \quad ([l \textbf{ if } \psi] \in \mathcal{D}) \tag{B.11}$$

$$ph(l,i) \leftarrow ph(\psi,i) \quad ([l \textbf{ if } \psi] \in \mathcal{D}) \tag{B.12}$$

$$h(L,i) \leftarrow not\ ph(\neg L,i) \tag{B.13}$$

$$o(E,0) \vee \neg o(E,0) \leftarrow \tag{B.14}$$

Lemma 13 *If δ_{i-1} is a partial state then $\langle \delta_{i-1}, a_{i-1}, \delta_i \rangle \in T^{lp}(\mathcal{D})$*

Proof. Suppose δ_{i-1} is a partial state. To prove that $\langle \delta_{i-1}, a_{i-1}, \delta_i \rangle \in T^{lp}(\mathcal{D})$ we need to show that $\Pi(a_{i-1}, \delta_{i-1})$ is consistent and its only answer set B satisfies

$$\{l \mid h(l,1) \in B\} = \delta_i$$

First, observe that because C is an answer set of $\pi(\mathcal{P}, n)$, its satisfies the constraint (6). As a result, a_{i-1} is safe in δ_{i-1}. By Lemma 9, this implies that $\Pi(a_{i-1}, \delta_{i-1})$ is consistent and thus, by Proposition 1, it has a unique answer set B. By Lemma 5, the program $\Pi_2(a_{i-1}, \delta_{i-1})$ (rules (A.23)–(A.29) with $a = a_{i-1}$ and $\delta = \delta_{i-1}$) has an answer set B_0 such that

$$\{l \mid h(l,1) \in B\} = \{l \mid h(l,1) \in B_0\}$$

Furthermore, such B_0 is unique because $\Pi_2(a_{i-1}, \delta_{i-1})$ is stratified.

Observe that the program $\Pi_2(a_{i-1}, \delta_{i-1})$ is the same as μ_i except that the time parameter of predicates in the former is 1 while it is i in the latter. Hence, we have

$$\{l \mid h(l,1) \in B_0\} = \delta_i$$

That is, the lemma holds. □

Let us go back to the proof of Theorem 5. It is easy to see that $\delta_0 = \delta^0$ and thus it is a partial state. By Lemma 13, it easy to see that for all $1 \leq i \leq n$ we have $\langle \delta_{i-1}, a_{i-1}, \delta_i \rangle \in Tlp(\mathcal{D})$.

Hence, we have $\langle \delta^0, \alpha, \delta_n \rangle \in T^{lp}(\mathcal{D})$. On the other hand, because C satisfies constraint (19), we have $\delta^f \subseteq \delta_n$. Accordingly, α is a solution of $\mathcal{P}$. Thus, the

theorem is proved.

C Proofs of Propositions 2–4 and Theorem 6

This appendix contains the proofs of Propositions 2–4 and Theorem 6. We assume that a simple planning problem $\mathcal{P}$ is given. In addition, for simplicity, we assume that the body of each static causal law of $\mathcal{D}$ has exactly one fluent literal as with some minor changes, the proofs in this appendix can be applied to simple action theories with arbitrary simple static causal laws, including those with an empty body. To make the proofs easy to follow, let us define some notions.

Definition 17 *Let a be an action and s be a state. A fluent literal l is called an* effect *of a in s if either*

(1) l is a direct effect of a in s; or
(2) $\mathcal{D}$ contains a static causal law

$$l \textbf{ if } g$$

such that g is an effect of a in s.

Definition 18 *Let a be an action and δ be a state. A fluent literal l is called a* possible effect *of a in δ iff either*

(1) l is a possible direct effect of a in δ; or
(2) l possibly holds by inertia, i.e., l possibly holds in δ and $\neg l$ is not a direct effect of a in δ; or
(3) $\mathcal{D}$ contains a static causal law

$$l \textbf{ if } g$$

such that g is a possible effect of a in δ.

The proofs in this appendix will make use of programs $\Phi_2(a, s)$ (rules (A.19)–(A.21)) and $\Pi_3(a, \delta)$ (rules (A.32)–(A.38)) and some results from Appendices A & B.

Let s and s' be states, δ and δ' be partial states and a be an action such that $\langle s, a, s' \rangle \in T(\mathcal{D})$ and $\langle \delta, a, \delta' \rangle \in T^{lp}(\mathcal{D})$. By Theorems 2 & 3, and Definition 8,the programs $\Phi(a, s)$ and $\Pi(a, \delta)$ have answer sets A and B, respectively, such that

$$s' = \{l \mid h(l, 1) \in A\} \tag{C.1}$$

and

$$\delta' = \{l \mid h(l,1) \in B\} \tag{C.2}$$

By Lemmas 4 and 6, this implies that the programs $\Phi_2(a,\delta)$ and $\Pi_3(a,\delta)$ have answer sets A_1 and B_1 respectively such that

$$A = U \cup V \cup A_1 \tag{C.3}$$

$$B = U \cup W \cup \{de(l,1) \mid l \text{ is a direct effect of } a \text{ in } \delta\} \cup B_1 \tag{C.4}$$

where U, V, and W are defined by (A.1) and (A.18), and (A.22). Furthermore, we have

$$h(l,1) \in A \text{ iff } h(l,1) \in A_1 \tag{C.5}$$

$$h(l,1) \in B \text{ iff } h(l,1) \in B_1 \tag{C.6}$$

Let P and Q be the reducts of $\Phi_2(a,s)$ and $\Pi_3(a,\delta)$ with respect to A_1 and B_1 respectively. That is, P is the set of rules (A.41)–(A.43) where $X = A_1$, and Q is the set of rules (A.44)–(A.49) where $X = B_1$.

Lemma 14 *If $g \notin s$ and $h(g,1) \in A_1$ then g is an effect of a in s.*

Proof. Let $i \geq 0$ be an arbitrary integer. Because $A_1 = \bigcup_i T_P^i(\emptyset)$, to prove the lemma, it suffices to show that if $g \notin s$ and $h(g,1) \in T_P^i(\emptyset)$ then g is an effect of a in s. We prove this by induction on i.

(1) Base case: $i = 0$. Trivial because there exists no fluent literal g such that $h(g,1) \in T_P^0(\emptyset) = \emptyset$.

(2) Inductive step: Suppose the lemma holds for $i \leq k$. We will show that it also holds for $i = k+1$. Let g be a fluent literal such that

$$g \notin s \wedge h(g,1) \in T_P^{k+1}(\emptyset)$$

Because g does not hold in s, the rule (A.43) with $L = g$ does not belong to P. As a result, there are two possibilities for $h(g,1) \in T_P^{k+1}(\emptyset)$:

(a) g is a direct effect of a in s. By Definition 17, g is an effect of a in s.

(b) $\mathcal{D}$ contains a static causal law

$$g \textbf{ if } h \tag{C.7}$$

such that

$$h(h,1) \in T_P^k(\emptyset) \tag{C.8}$$

It is easy to see that

$$h \notin s \tag{C.9}$$

because if otherwise, we would have $g \in s$ (note that s is a state and

thus it satisfies static causal law (C.7)).

From (C.8) and (C.9) and by the inductive hypothesis, it follows that h is an effect of a in s. Hence, by Definition 17, g is also an effect of a in s.

□

Lemma 15 *If $ph(g,1) \in B_1$ then g is a possible effect of a in δ.*

Proof. Let $i \geq 0$ be an arbitrary integer. Clearly, to prove the lemma, it suffices to show that

$$\text{if } ph(g,1) \in T_Q^i(\emptyset) \text{ then } g \text{ is a possible effect of } a \text{ in } \delta \tag{C.10}$$

Let us prove (C.10) by induction on i.

(1) Base case: $i = 0$. Trivial because there is no i such that $ph(g,1) \in T_Q^0(\emptyset) = \emptyset$.

(2) Inductive step: Suppose (C.10) is true for $i \leq k$. We will show that it is also true for $i = k+1$.

Let g be a fluent literal such that $ph(g,1) \in T_Q^{k+1}(\emptyset)$. Recall that Q is the set of rules of the form (A.44)–(A.49) where $X = B_1$. Hence, there are three possibilities for $ph(g,1) \in T_Q^{k+1}(\emptyset)$.

(a) g is a possible direct effect of a in δ and $\neg g$ is not a direct effect of a in δ. By definition, g is also a possible effect of a in δ.

(b) g possibly holds in δ and $\neg g$ is not a direct effect of a in δ. By definition, in this case g is also a possible effect of a in δ.

(c) $\mathcal{D}$ contains a static causal law

$$g \textbf{ if } h$$

such that $ph(h,1) \in T_Q^k(\emptyset)$. By the inductive hypothesis, h is a possible effect of a in δ. Hence, by Definition 18, g is also a possible effect of a in δ.

So, (C.10) is true. The lemma follows directly from this result. □

We will also need the following propostion.

Proposition 6 *Let $\mathcal{D}$ be a simple action theory. Let $\langle s, a, s' \rangle \in T(\mathcal{D})$ and $\langle \delta, a, \delta' \rangle \in T^{lp}(\mathcal{D})$. If $\delta \subseteq s$ then for every fluent literal $l \in s' \setminus \delta'$, we have $l \lhd (s \setminus \delta)$.*

Proof. Suppose $\delta \subseteq s$ and l be a fluent literal in $s' \setminus \delta'$. We need to show that there exists a fluent literal $g \in s \setminus \delta$ such that $l \lhd g$.

If $l \in s \setminus \delta$ then the proposition is trivial because by definition, l depends on itself and thus we can take $g = l \in s \setminus \delta$ to have $l \lhd g$. Now, consider the case that $l \notin s \setminus \delta$. There are two possibilities

(1) $l \notin s$, or
(2) $l \in \delta$.

Let us consider each possibility in turn.

(1) $l \notin s$. As $l \in s' \setminus \delta'$, we have $l \in s'$. This implies that $h(l, 1) \in A_1$. According to Lemma 14, l is an effect of a in s. It follows from Definition 17, that one of the following two cases occurs
   (a) $\mathcal{D}$ contains a dynamic causal law

$$e \textbf{ causes } l \textbf{ if } \psi$$

   such that $e \in a$ and ψ holds in s.

   If ψ holds in δ then l is a direct effect of a in δ. By rule (A.44), we have $h(l, 1) \in B_1$, i.e., $l \in \delta'$. This contradicts to $l \in s' \setminus \delta'$.

   Because ψ holds in s and does not hold in δ, we have

$$\psi \subseteq s \qquad \text{and} \qquad \psi \not\subseteq \delta$$

   As a result, there exists a fluent literal $g \in \psi$ such that $g \in s \setminus \delta$. By the definition of dependencies (Definition 11), we have $l \lhd g$.
   (b) $\mathcal{D}$ contains a dynamic causal law e **causes** l_0 **if** ψ and a sequence of static causal laws $[l_1 \textbf{ if } l_0], [l_2 \textbf{ if } l_1], \ldots, [l_n \textbf{ if } l_{n-1}], [l \textbf{ if } l_n]$ such that $e \in a$ and ψ holds in s.

   If ψ holds in δ then by rule (A.44), we have $l_0 \in \delta'$; on the other hand, because δ' is closed under the static causal laws of $\mathcal{D}$, it follows that $l \in \delta'$; this contradicts to the assumption $l \in s' \setminus \delta'$.

   So, we have ψ does not hold in δ. Similarly to previous case, this implies that there exists a fluent literal $g \in \psi$ such that $g \in s \setminus \delta$. Hence, we have

$$l \lhd l_n \lhd l_{n-1} \lhd \ldots \lhd l_0 \lhd g$$

(2) $l \in \delta$.

   First, we will show that $ph(\neg l, 1) \in B_1$. Since $l \notin \delta'$, we have $h(l, 1) \notin B_1$. By rule (A.49), it follows that $ph(\neg l, 1) \in B_1$.

   According to Lemma 15, $ph(\neg l, 1) \in B_1$ implies that $\neg l$ is a possible effect of a in δ. By the definition of a possible effect (Definition 18), we have the following three cases
   (a) $\neg l$ is a possible direct effect of a in δ. That is, $\mathcal{D}$ contains a dynamic causal law

$$e \textbf{ causes } \neg l \textbf{ if } \psi$$

such that $e \in a$ and ψ possibly holds in δ. This implies that

$$\neg\psi \cap \delta = \emptyset \tag{C.11}$$

As $l \in s'$ and s' is a state, we have $\neg l \notin s'$. This means that $h(\neg l, 1) \notin A$. By rule (A.41), it follows that $\neg l$ is not a direct effect of a in s. Hence, ψ does not hold in s, i.e.,

$$\neg\psi \cap s \neq \emptyset \tag{C.12}$$

From (C.11) and (C.12), it follows that there exists a fluent literal $g \in \neg\psi$ such that $g \in s \setminus \delta$. Because $[e$ **causes** $\neg l$ **if** $\psi]$ belongs to $\mathcal{D}$, this implies that $\neg l \triangleleft \neg g$. By the definition of dependencies, it follows that $l \triangleleft g$.

(b) l does not hold in δ and l is not a direct effect of a. Because $l \in \delta$, this case never happens.

(c) $\mathcal{D}$ contains a sequence of static causal laws

$$[l_1 \textbf{ if } l_0], [l_2 \textbf{ if } l_1], \ldots, [l_n \textbf{ if } l_{n-1}], [\neg l \textbf{ if } l_n]$$

such that l_0 is a possible effect of a in δ or l_0 possibly holds by inertia.

(i) l_0 is a possible direct effect of a. By definition, this means that $\mathcal{D}$ contains a dynamic causal law

$$e \textbf{ causes } l_0 \textbf{ if } \psi$$

such that $e \in a$ and ψ possibly holds in δ.

As ψ possibly holds in δ, we have

$$\neg\psi \cap \delta = \emptyset \tag{C.13}$$

On the other hand, it is easy to see that ψ does not hold in s as if otherwise, we would have $\neg l \in s'$, which contradicts to the assumption $l \in s'$. Therefore, we have

$$\neg\psi \cap s \neq \emptyset \tag{C.14}$$

By (C.13) and (C.14), there exists a fluent literal $g \in s \setminus \delta$ such that $\neg g \in \psi$. It is easy to see that

$$\neg l \triangleleft l_n \triangleleft l_{n-1} \triangleleft \ldots l_0 \triangleleft \neg g$$

Hence, we have $l \triangleleft g$.

(ii) l_0 possibly holds by inertia. This means that l_0 possibly holds in δ and $\neg l$ is not a direct effect of a in δ.

It is easy to see that l_0 does not hold in s as if otherwise, we would have $\neg l \in s'$, which is impossible due to $l \in s'$. Because s

is a state, it follows that $\neg l_0 \in s$.

As l_0 possibly holds in δ and $\neg l_0 \in s$, we have $\neg l_0 \in s \setminus \delta$. On the other hand, by the definition of dependencies, we have $l \triangleleft \neg l_0$. Accordingly, we can select $g = \neg l_0 \in s \setminus \delta$ to have $l \triangleleft g$.

C.1 Proof of Proposition 2

By the definition of $\gg_\sigma$ (Definition 13), δ is a subset of every state s in S. Hence, the right hand side of the equation of the proposition is a subset of the left hand side. Therefore, to prove the lemma, it is sufficient to show that

$$(\bigcap_{s \in S} s) \cap \sigma \subseteq \delta \cap \sigma$$

Suppose otherwise, that is, there exists a fluent literal l such that $l \in (\bigcap_{s \in S} s) \cap \sigma$ but $l \notin \delta \cap \sigma$. This implies that (i) $l \in \sigma$, and (ii) $l \in s \setminus \delta$ for every $s \in S$. The latter implies that $l \triangleleft (s \setminus \delta)$ for every $s \in S$. By the definition of $\gg$, $S \not\gg_\sigma \delta$. This is a contradiction.

C.2 Proof of Proposition 3

(1) Assume that a is not safe in δ. That means there exists an impossibility condition

$$\textbf{impossible } b \textbf{ if } \psi$$

such that $b \subseteq a$ and ψ possibly holds in δ, i.e.,

$$\neg\psi \cap \delta = \emptyset \tag{C.15}$$

By the definition of $\gg$, $S \gg_\sigma \delta$ implies that there exists a state $s \in S$ such that $b \not\lhd (s \setminus \delta)$. Because a is executable in s, ψ does not hold in s, i.e., $\psi \not\subseteq s$. Because s is a complete set of fluent literals, it follows that

$$\neg\psi \cap s \neq \emptyset \tag{C.16}$$

By (C.15) and (C.16), there exists a fluent literal $l \in (s \setminus \delta)$ such that $l \in \neg\psi$. By the definition of dependencies, we have $b \triangleleft l$ and this is a contradiction because $b \not\lhd (s \setminus \delta)$.

(2) Let $S' = Res(a, S)$.

Consider an arbitrary state $s \in S$. Because a is executable in S, it follows from the previous item that a is safe in δ. By Lemma 9, Proposition 1, and the definition of $T^{lp}(\mathcal{D})$, it follows that there exists a (unique) partial state δ' such that $\langle \delta, a, \delta' \rangle \in T^{lp}(\mathcal{D})$. We need to show that $S' \gg_\sigma \delta'$.

Suppose otherwise, that is, $S' \not\gg_\sigma \delta'$. Then, there are two possible cases (note that because $\delta \subseteq s$ for every $s \in S$, by Theorem 3, $\delta' \subseteq s'$ for every $s' \in S$):

(a) there exists a fluent literal $l \in \sigma$ such that $l \lhd (s' \setminus \delta')$ for every $s' \in S'$.

Let l be such a fluent literal. Consider an arbitrary state $s \in S$ and let $\langle s, a, s' \rangle$ be a transition in $T(\mathcal{D})$. Furthermore, let $g \in s' \setminus \delta'$ such that $l \lhd g$. By Proposition 6, because $g \in s' \setminus \delta'$, there must be a fluent literal $h \in s \setminus \delta$ such that $g \lhd h$. Because of the transitivity of $\lhd$, we have $l \lhd h$. This implies that

$$l \lhd (s \setminus \delta) \tag{C.17}$$

Because s can be any arbitrary state in S, (C.17) implies that $S \not\gg_\sigma \delta$. This is a contradiction.

(b) there exists an action b such that $b \lhd (s' \setminus \delta')$ for every $s' \in S'$.

Consider an arbitrary state $s \in S$ and let $\langle s, b, s' \rangle$ be a transition in $T(\mathcal{D})$. Furthermore, let $l \in s' \setminus \delta'$ be a fluent literal such that $b \lhd l$. By Proposition 6, because $l \in s' \setminus \delta'$, there exists a fluent literal g in $s \setminus \delta$ such that $l \lhd g$. By the definition of dependencies, it follows that $b \lhd g \in (s \setminus \delta)$. Hence, we have

$$b \lhd (s \setminus \delta) \tag{C.18}$$

Because s can be any arbitrary state in S, (C.18) implies that $S \not\gg_\sigma \delta$. This is a contradiction.

C.3 Proof of Proposition 4

Let $\alpha = \langle a_0, a_1, \ldots, a_{n-1} \rangle$. For $i \geq 0$, let $\alpha[i]$ denote the chain of the initial i events of α, i.e., $\alpha[i] = \langle a_0, a_1, \ldots, a_{i-1} \rangle$. We will prove the proposition by induction on the length n of α.

(1) Base case: $n = 0$.

Item 1 is trivial. Item 2 is true because $Res(\alpha, S) = S \gg_\sigma \delta$, $\langle \delta, \langle\rangle, \delta \rangle \in T^{lp}(\mathcal{D})$ and $T^{lp}(\mathcal{D})$ is deterministic.

(2) Inductive Step: Suppose the proposition is true for $n \leq k$. We need to show that it is true for $n = k + 1$.

Let $S_i = Res(\alpha[i], S)$ and let δ_i be the partial state such that $\langle \delta, \alpha[i], \delta_i \rangle \in T^{lp}(\mathcal{D})$. Clearly to prove the inductive step, we only need to show that

(a) a_k is safe in δ_k, and

(b) $S_{k+1} \gg_\sigma \delta_{k+1}$

By the inductive hypothesis, we have $S_k \gg_\sigma \delta_k$. By Proposition 3, it follows that a_k is safe in δ_k and $S_{k+1} \gg_\sigma \delta_{k+1}$.

C.4 Proof of Theorem 6

This theorem follows directly from Proposition 4 and the definition of a simple planning problem (Definition 14). If $\mathcal{P}$ has no solution then it is trivial. Suppose that $\mathcal{P}$ has a solution, say, $\alpha = \langle a_0, \ldots, a_{n-1} \rangle$. We will show that $\pi(\mathcal{P}, n)$ is consistent.

Because α is a solution of $\mathcal{P}$, there exists a sequence of sets of states $\langle S \rangle_{i=0}^{n}$ such that

(1) $S_0 = comp(\delta^0)$
(2) $S_i = Res(a_{i-1}, S_{i-1})$ for $i \geq 1$
(3) $\delta^f \subseteq s$ for every $s \in S_n$.

According to Proposition (4), there exists a sequence of partial states $\langle \delta_i \rangle_{i=0}^{n}$ such that $\delta_0 = \delta^0$ and $S_n \gg_{\delta^f} \delta_n$. By Proposition 2, we have $\delta^f \subseteq \delta_n$.

Let us construct a sequence of sets of atoms D_i as follows:

$$D_0 = h(\delta^0, 0) \cup o(a_0, 0) \cup \neg o(A \setminus a_0, 0)$$

for $1 \leq i \leq n-1$,

$$\begin{aligned} D_i = {} & h(l, \delta_i) \cup o(a_i, i) \cup \neg o(A \setminus a_i, i) \\ & \{de(l, i) \mid l \in de(a_{i-1}, \delta_{i-1})\} \cup \{ph(l, i) \mid l \in ph(a_{i-1}, \delta_{i-1})\} \end{aligned}$$

and

$$\begin{aligned} D_n = {} & \{h(l, n) \mid l \in \delta_n\} \cup \\ & \{de(l, n) \mid l \in de(a_{n-1}, \delta_{n-1})\} \cup \{ph(l, n) \mid l \in ph(a_{n-1}, \delta_{n-1})\} \end{aligned}$$

It is easy to see that for $0 \leq i \leq n$, D_i is an answer set of μ_i (defined previously in Appendix B). By Lemma 12, $C_1 = \bigcup_{i=0}^{n} D_i$ is an answer set of $\pi_1(\mathcal{P}, n)$. As a result, by Lemma 11, $C = C_1 \cup U$ is an answer set of $\pi_0(\mathcal{P}, n)$.

We will show that C is also an answer set of $\pi(\mathcal{P}, n)$. Because $\pi(\mathcal{P}, n)$ is the program $\pi_0(\mathcal{P}, n)$ with additional constraints (6), (8), (19), and (21), all we need to do is to show that C does not violate any of these constraints. For (6) and (8), it is trivial because δ_i is a partial state and$\langle \delta_{i-1}, a_{i-1}, \delta_i \rangle \in T^{lp}(\mathcal{D})$. Constraint (19) is satisfied by C because $\delta^f \subseteq \delta_n$. Constraint (21) is satisfied because a_i is an action, i.e., a non-empty set of elementary actions.

Hence, C is an answer set of $\pi(\mathcal{P}, n)$. This means that the program $\pi(\mathcal{P}, n)$ is

consistent. As a result, the theorem holds.

D Proof of Theorem 8

We begin with a lemma about the operator $Cl_{\mathcal{D}}$ that will be used in the proof.

Given an action theory $\mathcal{D}$, for a set of fluent literals σ, let

$$\Lambda(\sigma) = \sigma \cup \{l \mid \exists[l \textbf{ if } \psi] \in \mathcal{D} \text{ such that } \psi \subseteq \sigma\} \tag{D.1}$$

Let $\Lambda^0(\sigma) = \sigma$ and $\Lambda^{i+1}(\sigma) = \Lambda(\Lambda^i(\sigma))$ for $i \geq 0$. Since, by the definition of Λ, for any set of fluent literals σ' we have $\sigma' \subseteq \Lambda(\sigma')$, the sequence $\langle \Lambda^i(\sigma) \rangle_{i=0}^{\infty}$ is monotonic with respect to the set inclusion operation. In addition, $\langle \Lambda^i(\sigma) \rangle_{i=0}^{\infty}$ is bounded by the set of fluent literals. Thus, there exists $\sigma^{\texttt{limit}}$ such that

$$\sigma_{\mathcal{D}}^{\texttt{limit}} = \bigcup_{i=0}^{\infty} \Lambda^i(\sigma) \tag{D.2}$$

Furthermore, $\sigma_{\mathcal{D}}^{\texttt{limit}}$ is unique and satisfies all static causal laws in $\mathcal{D}$.

Lemma 16 *For any set of fluent literals σ, we have $\sigma_{\mathcal{D}}^{\texttt{limit}} = Cl_{\mathcal{D}}(\sigma)$.*

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

$$\sigma_{\mathcal{D}}^{\texttt{limit}} \subseteq Cl_{\mathcal{D}}(\sigma)$$

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

$$Cl_{\mathcal{D}}(\sigma) \subseteq \sigma_{\mathcal{D}}^{\texttt{limit}}$$

Accordingly, we have

$$\sigma_{\mathcal{D}}^{\texttt{limit}} = Cl_{\mathcal{D}}(\sigma)$$

□

Lemma 17 *For every set of fluent literals σ,* Closure($\mathcal{D}$,σ) $= Cl_{\mathcal{D}}(\sigma)$.

Proof. It is easy to see that the function Closure($\mathcal{D}$,σ) is a straightforward computation of $\sigma_{\mathcal{D}}^{\texttt{limit}}$ (Equations (D.2) and (D.1)). Hence, by Lemma 16, we have Closure($\mathcal{D}$,σ) $= Cl_{\mathcal{D}}(\sigma)$. □

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

Lemma 18 *Let $i \geq 0$ be an arbitrary integer, and x be a binary predicate symbol. For any set σ of fluent literals, the following program*

$$\begin{aligned} x(l,i) \leftarrow & \qquad (l \in \sigma) \\ x(l,i) \leftarrow x(\psi,i) & \qquad ([l \textbf{ if } \psi] \in \mathcal{D}) \end{aligned}$$

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

Proof. By the definition of an answer set of a positive program, it is easy to see that the above program has the unique answer set $\{x(l,i) \mid l \in \sigma_{\mathcal{D}}^{\mathtt{limit}}\} = \{x(l,i) \mid l \in Cl_{\mathcal{D}}(\sigma)\}$ (see Lemma 16). □

Let a be an action and δ be a partial state such that $\Pi(a,\delta)$ is consistent. By Proposition 1, $\Pi(a,\delta)$ has a unique answer set, say B. Let lit denote the set of all fluent literals, i.e., $lit = \mathbf{F} \cup \neg\mathbf{F}$. We define

$$de(a,\delta) = \{l \mid l \text{ is a direct effect of } a \text{ in } \delta\} \tag{D.3}$$

$$pde(a,\delta) = \{l \mid l \text{ is a possible direct effect of } a \text{ in } \delta\} \tag{D.4}$$

$$ph(a,\delta) = Cl_{\mathcal{D}}((pde(a,\delta) \cup (lit \setminus \neg\delta)) \setminus de(a,\delta)) \tag{D.5}$$

Then, we have the following lemma.

Lemma 19 *For any fluent literal l, $h(l,1) \in B$ iff $l \in Cl_{\mathcal{D}}(de(a,\delta) \cup (lit \setminus \neg ph(a,\delta)))$.*

Proof. Because B is an answer set of $\Pi(a,\delta)$, it is also an answer set of $\Pi_0(a,\delta)$ (recall that $\Pi_0(a,\delta)$ is the program obtained from $\Pi(a,\delta)$ by removing constraints).

According to Lemma 6, there exists an answer set B_1 of the program $\Pi_3(a,\delta)$ (rules (A.32)–(A.38)) such that (A.39) and (A.40) hold.

It is easy to see that $X = \{ph(l,1) \mid l \in lit\}$ is a splitting set of $\Pi_3(a,\delta)$. The bottom part of $\Pi_3(a,\delta)$ with respect to X is the following set of rules

$$\begin{aligned} ph(l,1) \leftarrow & \\ & (l \text{ is a possible direct effect of } a \text{ in } \delta, \neg l \text{ is not a direct effect of } a) \\ ph(L,1) \leftarrow & \\ & (L \text{ possibly holds in } \delta, \neg L \text{ is not a direct effect of } a) \\ ph(l,1) \leftarrow & \, ph(\psi,1) \\ & ([l \textbf{ if } \psi] \in \mathcal{D}) \end{aligned}$$

which can be rewritten to (see (D.3) and (D.4) for the definition of $de(a,\delta)$

and $pde(a, \delta)$)

$$
\begin{aligned}
ph(l, 1) \leftarrow & \\
& (l \in (pde(a, \delta) \setminus \neg de(a, \delta)) \cup (lit \setminus \neg(\delta \cup de(a, \delta))) \\
ph(l, 1) \leftarrow & \, ph(\psi, 1) \\
& ([l \textbf{ if } \psi] \in \mathcal{D})
\end{aligned}
$$

By Lemma 17, this program has a unique answer set (see (D.5) for the definition of $ph(a, \delta)$)

$$M = \{ph(l, 1) \mid l \in ph(a, \delta)\}$$

Hence, the evaluation of the top part of $\Pi_3(a, \delta)$ with respect to M is the following set of rules:

$$
\begin{aligned}
h(l, 1) \leftarrow & \\
& (l \in de(a, \delta)) \\
h(l, 1) \leftarrow & \, h(\psi, 1) \\
& ([l \textbf{ if } \psi] \in \mathcal{D}) \\
h(L, 1) \leftarrow & \\
& (\neg l \notin ph(a, \delta))
\end{aligned}
$$

Again, by Lemma 17, this program has a unique answer set (note that $(\neg l \notin ph(a, \delta)) \Leftrightarrow (l \in (lit \setminus \neg ph(a, \delta)))$):

$$N = \{h(l, 1) \mid l \in Cl_{\mathcal{D}}(de(a, \delta) \cup (lit \setminus \neg ph(a, \delta)))\}$$

By the splitting set theorem, we have $B_1 = M \cup N$. Hence by (A.40), we have

$$(h(l, 1) \in B) \Leftrightarrow (l \in Cl_{\mathcal{D}}(de(a, \delta) \cup (lit \setminus \neg ph(a, \delta))))$$

Hence, the lemma holds. □

We now show that Theorem 8 holds. By Lemma 18, it is easy to see that $\textsc{Res}(\mathcal{D}, a, \delta) = Cl_{\mathcal{D}}(de(a, \delta) \cup (lit \setminus \neg ph(a, \delta)))$. By Lemma 19, it follows that Theorem 8 holds.