An Approximation of Action Theories of $\mathcal{AL}$ and Its Application to Conformant Planning

Tran Cao Son[1], Phan Huy Tu[1], Michael Gelfond[2], and A. Ricardo Morales[2]

[1] Department of Computer Science, New Mexico State University,
Las Cruces, NM 88003, USA
{tson, tphan}@cs.nmsu.edu
[2] Department of Computer Science, Texas Tech University, Lubbock, TX 79409, USA
{mgelfond, ricardo}@cs.ttu.edu

Abstract. In this paper we generalize the notion of approximation of action theories introduced in [13,26]. We introduce a logic programming based method for constructing approximation of action theories of $\mathcal{AL}$ and prove its soundness. We describe an approximation based conformant planner and compare its performance with other state-of-the-art conformant planners.

1 Introduction and Motivation

Static causal laws (a.k.a. *state constraints* or *axioms*) constitute an important part of every dynamic domain. Unlike an effect of an action, a static causal law represents a relationship between fluents. For example,

(a) In the travel domain, the static causal law "*one person cannot be at A if he is at B*" states that $at(B)$ is false if $at(A)$ is true;

(b) In the block world domain, the static causal law "*block A is above block B if A is on B*" says that $above(A, B)$ is true if $on(A, B)$ is true;

Static causal laws can cause actions to have *indirect effects*. For example, the action of putting the block A atop the block B, denoted by $put(A, B)$, causes $on(A, B)$ to be true. The static causal law (b) implies that $above(A, B)$ is also true, i.e., $above(A, B)$ is an indirect effect of $put(A, B)$. The problem of determining such indirect effects is known as the *ramification problem* in the area of reasoning about action and change (RAC).

In the last decade, several solutions to the ramification problem have been proposed. Each of these solutions extends a framework for RAC to allow static causal laws [2,18,22,23,20,15,17]. While being intensively studied by the RAC's research community, static causal laws have rarely been *directly* considered by the 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 (or static causal laws in our notation) [14], most of the planning domains used in the recent planning competitions [1,19,11] do not include axioms. The main reason for this practice is that it is widely believed that axioms can be compiled into actions' effect propositions; thus, making the representation of and reasoning about axioms become unnecessary in planning. This is partly true due to the fact that PDDL only allows non-recursive axioms. In a recent paper [29], it is proved

that adding axioms to the planning language not only improves the readability and elegance of the representation but also increases the expressiveness of the language. It is also shown that the addition of a component to handle axioms in a planner can indeed improve the performance of the planner.

The main difficulty in planning in domains with static causal laws lies directly in defining and computing the successor states. In general, domains with static causal laws are nondeterministic; for example, in a theory with a single action a and three fluents f, g, and h with the property that execution of a causes f to become true and the two static causal laws

(i) if f is true and g is false then h must be true; and,
(ii) if f is true and h is false then g must be true.

Intuitively, the execution of a in a state where f, h, and g are false will yield two possible states. In one state, f and g are true and h is false. In the other one, f and h are true and g is false. This nondeterminism leads to the fact that the execution of an action sequence can generate different trajectories. Thus, exact planning[1] is similar to conformant planning, an approach to dealing with incomplete information in planning. It is also worth noticing that the complexity of conformant planning (Σ_2^P) is much higher than planning in deterministic domains (**NP**-complete) [3,30]. It is also pointed out in [3] that approximations of the transition function between states can help reduce the complexity of the planning problem.

In this paper, we further investigate the notion of approximations of action theories introduced in [26,13]. We define an approximation for action theories of $\mathcal{AL}$. The key difference between the newly developed approximation and those proposed in [26,13] is that it is applicable for action descriptions with arbitrary static causal laws: while the approximation proposed in [13] is only for specific type of state constraints, the approximations in [26] are defined for action descriptions with sensing actions but without state constraints. We use a logic program in defining the approximation.

The paper is organized as follows. In the next section, we review the basics of the language $\mathcal{AL}$. Afterward, we define an approximation of $\mathcal{AL}$ action theories. We then proceed with the description of a logic programming based conformant planner which makes use of the approximation. We then compare the performance of our planner with some conformant planners which are closely related to our planner.

2   Syntax and Semantics of $\mathcal{AL}$

We consider domains which can be represented by a transition diagram whose nodes are possible states of the domain and whose arcs are actions that take the domain from one state to another. Paths of the diagram correspond to possible trajectories of the system. We limit our attention to transition diagrams which can be defined by action descriptions of the action language $\mathcal{AL}$ from [4]. The signature Σ of an action description of $\mathcal{AL}$ consists of two disjoint, non-empty sets of symbols: the set **F** of fluents, and

---

[1] By exact planning we mean the problem of finding a polynomial-bounded length sequence of actions that can achieve the goal at the end of every possible trajectory generated by the action sequence.

the set $\mathbf{A}$ of *elementary actions*. By an *action* we mean a non-empty set a of elementary actions. Informally we interpret an execution of a as a simultaneous execution of its components. For simplicity we identify an elementary action e with $\{e\}$. By *fluent literals* we mean fluents and their negations. By $\bar{l}$ we denote the fluent literal complementary to l. A set S of fluent literals is called *complete* if, for any $f \in \mathbf{F}$, $f \in S$ or $\neg f \in S$. An action description $\mathcal{D}$ of $\mathcal{AL}$ is a collection of statements of the form:

$$e \textbf{ causes } l \textbf{ if } p \tag{1}$$

$$l \textbf{ if } p \tag{2}$$

$$\textbf{impossible } a \textbf{ if } p \tag{3}$$

where e is an elementary action, a is an action, l is a fluent literal, and p is a set of fluent literals from the signature Σ of $\mathcal{D}$. The set p is often referred to as the *precondition* of the corresponding statement. When it 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 p then l will hold in any resulting state. Statement (2), called a *static causal law*, says that any state satisfying p must satisfy l. Statement (3) is an *impossibility condition*. It says that action a cannot be performed in a state satisfying p. We next define the transition diagram, $T(\mathcal{D})$ specified by an action description $\mathcal{D}$ of $\mathcal{AL}$.

A set of literals S is *closed* under a static causal law (2) if $l \in S$ whenever $p \subseteq S$. By $Cn(S)$, we denote the smallest set of literals that contains S and is closed under the static causal laws of $\mathcal{D}$. A *state* σ of $T(\mathcal{D})$ is a complete, consistent set of literals closed under the static causal laws of $\mathcal{D}$. An action b is said to be *prohibited* in σ if $\mathcal{D}$ contains an impossibility condition (3) such that $p \subseteq \sigma$ and $a \subseteq b$. $E(a, \sigma)$ stands for the set of all fluent literals l for which there is a causal law (1) in $\mathcal{D}$ such that $p \subseteq \sigma$ and $e \in a$. Elements of $E(a, \sigma)$ are called *direct effects* of the execution of a in σ.

Definition 1 ([21]). For an action a and two states σ_1 and σ_2, a transition $\langle \sigma_1, a, \sigma_2 \rangle \in T(\mathcal{D})$ iff a is not prohibited in σ_1 and $\sigma_2 = Cn(E(a, \sigma_1) \cup (\sigma_1 \cap \sigma_2))$.

An alternate sequence of states and actions, $M = \langle \sigma_0, a_0, \sigma_1, \ldots, a_{n-1}, \sigma_n \rangle$, is a *path* in a transition diagram $T(\mathcal{D})$ if $\langle \sigma_i, a_i, \sigma_{i+1} \rangle \in T(\mathcal{D})$ for $0 \leq i < n$. M is called a *model* of the chain of events $\alpha = \langle a_0, \ldots, a_{n-1} \rangle$; σ_0 (resp. σ_n) is referred to as the *initial state* (resp. *final state*) of M; M *entails* a set of fluent literals s, written as $M \models s$, if $s \subseteq \sigma_n$. We sometime write $\langle \sigma_0, \alpha, \sigma_n \rangle \in T(\mathcal{D})$ to denote that there exists a model of α whose initial state and final state is σ_0 and σ_n, respectively. An action description $\mathcal{D}$ is called *deterministic* if for any state σ_1 and action a there is at most one successor state σ_2 such that $\langle \sigma_1, a, \sigma_2 \rangle \in T(\mathcal{D})$. Note that if $\mathcal{D}$ is deterministic there can be at most one model for α given the initial state σ_0 and final state σ_n. We denote this model by $\sigma_n = \alpha(\sigma_0)$. Notice that in the presence of static causal laws, action theories can be nondeterministic. As an example, the second theory in the introduction can be described by the action description $\mathcal{D}_0$ consisting of the following statements:
$$\mathcal{D}_0 = \big\{\, a \textbf{ causes } f \qquad g \textbf{ if } f, \neg h \qquad h \textbf{ if } f, \neg g \,\big\}$$
Observe that $T(\mathcal{D}_0)$ includes the transitions $\langle \{\neg f, \neg h, \neg g\}, a, \{f, h, \neg g\} \rangle$ and $\langle \{\neg f, \neg h, \neg g\}, a, \{f, g, \neg h\} \rangle$. Hence, $\mathcal{D}_0$ is non-deterministic.

An action a is *executable* in state σ_1 if there is a state σ_2 such that $\langle \sigma_1, a, \sigma_2 \rangle \in T(\mathcal{D})$; a chain of events $\alpha = \langle a_1, \ldots, a_{n-1} \rangle$ is executable in a state σ if there exists a

path $\langle \sigma, \alpha, \sigma' \rangle$ in $T(\mathcal{D})$ for some σ'; $\mathcal{D}$ is called *consistent* if for any state σ_1 and action a which is not prohibited in σ_1 there exists at least one successor state σ_2 such that $\langle \sigma_1, a, \sigma_2 \rangle \in T(\mathcal{D})$.

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

Normally an agent does not have complete information about its current state. Instead its knowledge is limited to the current *partial state* – a consistent collection of fluent literals closed under the static causal laws of the agent's action description $\mathcal{D}$. In what follows partial states and states are denoted by (possibly indexed) letters s and σ respectively.

A state σ that includes a partial state s is called a *completion* of s. By $comp(s)$ we denote the set of all completions of s. An action a is *safe* in s if it is executable in every completion of s. A chain of events $\alpha = \langle a_0, \ldots, a_{n-1} \rangle$ is *safe* in s if (i) a_0 is safe in s; and (ii) for every state σ' such that $\langle \sigma, a_0, \sigma' \rangle \in T(\mathcal{D})$ for some $\sigma \in comp(s)$, $\langle a_1, \ldots, a_{n-1} \rangle$ is safe in σ'.

For many of its reasoning tasks the agent may need to know the effects of its actions which are determined by the fluents from s (as opposed to the actual completion of s). In [26] the authors suggest to model such knowledge by a transition function which approximates the transition diagram $T(\mathcal{D})$ for deterministic action theories with sensing actions. We will next generalize this notion to action theories in $\mathcal{AL}$. Even though approximations can be non-deterministic, in this paper we will be interested only in deterministic approximations.

Definition 2 (Approximation). $T'(\mathcal{D})$ is an *approximation* of $T(\mathcal{D})$ if

1. States of $T'(\mathcal{D})$ are partial states of $T(\mathcal{D})$.
2. If $\langle s, a, s' \rangle \in T'(\mathcal{D})$ then for every $\sigma \in comp(s)$,
    (a) a is executable in σ and,
    (b) $s' \subseteq \sigma'$ for every σ' such that $\langle \sigma, a, \sigma' \rangle \in T(\mathcal{D})$.

An approximation $T'(\mathcal{D})$ is *deterministic* if for each partial state s and action a, there exists at most one s' such that $\langle s, a, s' \rangle \in T'(\mathcal{D})$. The next observation shows that an approximation must be sound.

Observation 1. *Let $T'(\mathcal{D})$ be an approximation of $T(\mathcal{D})$. Then, for every chain of events α if $\langle s, \alpha, s' \rangle \in T'(\mathcal{D})$ then for every $\sigma \in comp(s)$, (a) α is executable in σ; and (b) $s' \subseteq \sigma'$ for every σ' such that $\langle \sigma, \alpha, \sigma' \rangle \in T(\mathcal{D})$.*

In what follows we describe a method for constructing approximations of action theories of $\mathcal{AL}$. In our approach, the transitions in $T'(\mathcal{D})$ will be defined by a logic program $\pi(\mathcal{D})$ called the *cautious encoding* of $\mathcal{D}$. The signature of $\pi(\mathcal{D})$ includes terms corresponding to fluent literals and actions of $\mathcal{D}$, as well as non-negative integers used to represent time steps. For convenience, we often write $\pi(\mathcal{D}, n)$ to denote the program $\pi(\mathcal{D})$ where the time constants take values between 0 and n. Atoms of $\pi(\mathcal{D})$ are formed by the following (sorted) predicate symbols:

– $h(l, T)$ is true if literal l holds at time-step T;
– $o(e, T)$ is true if action e occurs at time-step T;

- $dc(l, T)$ is true if literal l is a direct effect of an action that occurs at time $T-1$; and
- $ph(l, T)$ is true if literal l possibly holds at time T.

The program also contains a set of auxiliary predicates, including $time$, $fluent$, and $action$, for enumerating constants of sorts time, fluent, and action respectively; $literal$ and $contrary$ for defining literals and complementary literals, respectively[2].

In our representation, letters T, F, L, and A (possibly indexed) are used to represent variables of sorts time, fluent, literal, and action correspondingly. Morever, we also use some shorthands: if a is an action then $o(a, T) = \{o(e, T) : e \in a\}$. For a set of fluent literals p, and a predicate symbol $\rho \in \{h, dc, ph\}$, $\rho(p, T) = \{\rho(l, T) : l \in p\}$ and $not\ \rho(p, T) = \{not\ \rho(l, T) : l \in p\}$. For a fluent f, by $\bar{l}$ we mean $\neg f$ if $l = f$ and f if $l = \neg f$. Literals $\bar{l}$ and l are called contrary literals. For a set of literals p, $\bar{p} = \{\bar{l} : l \in p\}$. The set of rules of $\pi(\mathcal{D})$ consists of those encoding the laws in $\mathcal{D}$, those encoding the inertial axioms, and some auxiliary rules. We next describe these subsets of rules:

1. For each dynamic causal law (1) in $\mathcal{D}$, the rules

$$h(l, T+1) \leftarrow o(e, T), h(p, T) \qquad (4)$$
$$dc(l, T+1) \leftarrow o(e, T), h(p, T) \qquad (5)$$

   belong to $\pi(\mathcal{D})$. The first rule states that l holds at $T + 1$ if e occurs at T and the condition p holds at T. The second rule indicates that l is a direct effect of the execution of e. Since the state at the time moment T might be incomplete, we add to $\pi(\mathcal{D})$ the rule

$$ph(l, T+1) \leftarrow o(e, T), not\ h(\bar{p}, T) \qquad (6)$$

   which says that l might hold at $T+1$ if e occurs at T and the precondition p possibly holds at T.

2. For each static causal law (2) in $\mathcal{D}$, $\pi(\mathcal{D})$ contains the two rules:

$$h(l, T) \leftarrow h(p, T) \qquad (7)$$
$$ph(l, T) \leftarrow ph(p, T) \qquad (8)$$

   These rules basically state that if p holds (or possibly holds) at T then so does l.

3. For each impossibility condition (3) in $\mathcal{D}$, we add to $\pi(\mathcal{D})$ the following rule:

$$\leftarrow o(a, T), not\ h(\bar{p}, T) \qquad (9)$$

   This rule states that a cannot occur if the condition p possibly holds.

4. The inertial law is encoded as follows:

$$ph(L, T+1) \leftarrow not\ h(\bar{L}, T), not\ dc(\bar{L}, T+1) \qquad (10)$$
$$h(L, T) \leftarrow not\ ph(\bar{L}, T), T \neq 0 \qquad (11)$$

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

---

[2] Some adjustment to this syntax is needed if one wants to use some of the existing answer set solvers. For instance, since Cmodels does not allow $h(\neg f, T)$ we may replace it with, say, $h(neg(f), T)$. Besides, to simplify our representation, we make use of choice rules introduced in [24].

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

$$\leftarrow h(F,T), h(\neg F, T) \tag{12}$$
$$literal(F) \leftarrow fluent(F) \tag{13}$$
$$literal(\neg F) \leftarrow fluent(F) \tag{14}$$
$$contrary(F, \neg F) \leftarrow fluent(F) \tag{15}$$
$$contrary(\neg F, F) \leftarrow fluent(F) \tag{16}$$

The first constraint guarantees that two contrary literals cannot hold at the same time. The last four rules are used to define fluent literals and complementary literals.

At this point, it is worthwhile to provide the intuition behind the of atoms $dc(l, T)$, $h(l, T)$, and $ph(l, T)$. Let a be an action and s be a partial state. Consider an "one-step" program $\Pi = \pi(\mathcal{D}, 1) \cup \{h(l, 0) \mid l \in s\} \cup \{o(a, 0)\}$.

Observe that Definition 1 implies that a literal l belongs to a possible next state if

1. it is an direct effect of a, i.e., $l \in E(a, \sigma_1)$;
2. it holds by inertial, i.e., $l \in (\sigma_1 \cap \sigma_2)$; or,
3. it is an indirect effects of a, i.e., $l \in \sigma_2 \setminus (E(a, \sigma_1) \cup (\sigma_1 \cap \sigma_2))$. In other words, it is *caused* by a static causal law.

Let S_1, S_2, and S_3 denote the three sets of literals corresponding to the above three cases with respect to the partial state s. Since s might be incomplete, these three sets cannot be computed in full. Our approach is to conservatively estimate the next partial state by

(i) underestimate S_1 by considering only what *definitely will hold given s*. This set is encoded by the set of atoms of the form $dc(l, 1)$ and is computed by the rule (5);

(ii) overestimate the negation of S_2 by considering what *can possibly hold in the next state*. This set is encoded by the set of atoms of the form $ph(l, 1)$ and is computed by the rules (6) and (8); and

(iii) underestimate S_3 by considering only what *definitely will hold* and what *cannot possibly change* in the construction of the next state. This is encoded by the rules (10)-(11) and (7).

Definition 3. *Let $T^{lp}(\mathcal{D})$ be a transition diagram such that $\langle s, a, s' \rangle \in T^{lp}(\mathcal{D})$ iff s is a partial state and $s' = \{l \mid h(l, 1) \in \mathcal{A}\}$ where $\mathcal{A}$ is the answer set of $\pi(\mathcal{D}, 1) \cup h(s, 0) \cup \{o(a, 0)\}$.*

The following theorem[3] show that $T^{lp}(\mathcal{D})$ is sound with respect to $T(\mathcal{D})$.

Theorem 1 (Soundness). *If $\mathcal{D}$ is consistent then $T^{lp}(\mathcal{D})$ is a deterministic approximation of $T(\mathcal{D})$.*

4   Approximation Based Conformant Planners

We will now turn our attention to the conformant planning problem in action theories of $\mathcal{AL}$. We begin with the definition of a planning problem.

---

[3] Proofs of theorems are omitted to save space.

Definition 4. A *planning problem* is a tuple $\langle \mathcal{D}, s^0, s^f \rangle$ where s^0 and s^f are partial states of $\mathcal{D}$.

Partial states s^0 and s^f characterize possible initial situations and the goal respectively.

Definition 5. A chain of events $\alpha = \langle a_0, \ldots, a_{n-1} \rangle$ is a *solution* to a planning problem $\mathcal{P} = \langle \mathcal{D}, s^0, s^f \rangle$ if α is safe in s^0, and for every model M of α with a possible initial state $\sigma_0 \in comp(s^0)$, $M \models s^f$.

We often refer to α as a plan for s^f. If s^0 is a state and action description $\mathcal{D}$ is deterministic then α is a "classical" plan, otherwise it is a *conformant plan*. We next illustrate these definitions using the well-known bomb-in-the-toilet example.

Example 1 (Bomb in the toilet). There is a finite set of toilets and a finite set of packages. One of the packages contains a bomb. The bomb can be disarmed by dunking the package that contains it in a toilet. Dunking a package clogs the toilet. Flushing a toilet unclogs it. Packages can only be dunked in unclogged toilets, one package per toilet. The objective is to find a plan to disarm the bomb. This domain can be modeled by the action description $\mathcal{D}_1$ which consists of the following laws:

$dunk(P, E)$ **causes** $\neg armed(P)$	**impossible** $dunk(P, E)$ **if** $clogged(E)$
$dunk(P, E)$ **causes** $clogged(E)$	**impossible** $\{dunk(P, E), flush(E)\}$
$flush(E)$ **causes** $\neg clogged(E)$	**impossible** $\{dunk(P_1, E), dunk(P_2, E)\}$
	impossible $\{dunk(P, E_1), dunk(P, E_2)\}$

E and P are variables for toilets and packages respectively; E_1 and E_2 stand for different toilets and P_1 and P_2 stand for different packages. Note that the last three statements specify physical impossibilities of some concurrent actions and the domain does not have a static causal law.

Let n and m denote the number of packages and toilets respectively. A planning problem in this domain, denoted by $BMTC(n, m)$, is often given by $\langle \mathcal{D}_1, s^0, s^f \rangle$ where s^0 is a (possibly empty) collection of literals of the form $\neg armed(P)$, where P denotes some package. The goal s^f contains $\{\neg armed(1), \ldots, \neg armed(n)\}$.

Consider the problem $BMTC(2, 1)$. We can easily check that if σ is a state containing $\neg clogged(1)$ then $\langle \sigma, dunk(1, 1), \sigma' \rangle$ is a transition in $T(\mathcal{D}_1)$ where $\sigma' = (\sigma \setminus \{armed(1), \neg clogged(1)\}) \cup \{\neg armed(1), clogged(1)\}$. Furthermore,

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

is safe in the partial state $\emptyset$ and α is a solution to the problem $BMTC(2, 1)$. □

It is not difficult to show that there is a close relationship between conformant plans and paths of an approximation $T'(\mathcal{D})$ of $T(\mathcal{D})$. Because of the soundness of an approximation, it follows from Observation 1 that if $\langle s, \alpha, s' \rangle \in T'(\mathcal{D})$, $s \subseteq s^0$, and $s^f \subseteq s'$ then α is a safe solution in s^0 of the planning problem $\langle \mathcal{D}, s^0, s^f \rangle$.

Since $T^{lp}(\mathcal{D})$ is an approximation of $T(\mathcal{D})$, we can use the program $\pi(\mathcal{D})$ to compute safe solutions of the planning problem $\mathcal{P} = \langle \mathcal{D}, s^0, s^f \rangle$. Furthermore, because $T^{lp}(\mathcal{D})$ is deterministic and computing the next state can be done in polynomial time, we can show that the complexity of the conformant planning problem with respect to $T^{lp}(\mathcal{D})$ is reduced to **NP**-complete (comparing to Σ_2^P, see [30]).

We will next describe the program $\pi(\mathcal{P})$ for this purpose. Like $\pi(\mathcal{D})$, the signature of $\pi(\mathcal{P})$ includes terms corresponding to fluent literals and actions of $\mathcal{D}$. We add to $\pi(\mathcal{P})$ a constant, $length$, which represents the plan length, i.e., time steps can take value in the interval $[0, length]$. We also write $\pi(\mathcal{P}, n)$ to denote the program $\pi(\mathcal{P})$ with $length$ equal to n. $\pi(\mathcal{P})$ consists of $\pi(\mathcal{D})$ and the following rules:

1. *Rules encoding the initial state:* for each $l \in s^0$, we add to $\pi(\mathcal{P})$ the rule:

$$h(l, 0) \leftarrow \qquad (17)$$

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

$$\leftarrow not\ h(l, length)$$

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

3. *Action generation rule:* as in other ASP-planners, $\pi(\mathcal{P})$ contains the rule for generating action occurrences:

$$1\{o(A, T) : action(A)\} \leftarrow T < length \qquad (18)$$

   which says that at each moment of time T, some action must occur[4].

With the help of Theorem 1, we can prove the correctness of the planner $\pi(\mathcal{P})$.

Theorem 2. *Let $\mathcal{A}$ be an answer set of $\pi(\mathcal{P}, n)$. It holds that*

- *for every $0 \leq i < n$, if $a_i = \{e \mid o(e, i) \in \mathcal{A}\}$ then a_i is an action which is not prohibited in $\{l \mid h(l, i) \in \mathcal{A}\}$; and*
- *$\alpha = \langle a_0, \ldots, a_{n-1} \rangle$ is a solution to $\mathcal{P}$.*

This theorem allows us to use $\pi(\mathcal{P})$ for computing minimal plans of $\mathcal{P}$. This is done by sequentially computing the answer sets of $\pi(\mathcal{P}, 0), \pi(\mathcal{P}, 1), \ldots$. In the next section, we will describe our experiments with $\pi(\mathcal{P})$. From now on, we will refer to $\pi(\mathcal{P})$ as CPASP[5]. Before going on, we would like to mention that $\pi(\mathcal{P})$ is not complete. One of the main reasons for the incompleteness of $\pi(\mathcal{P})$ lies in its limited capability in reasoning-by-cases. The next example demonstrates this issue.

Example 2. Consider the action description $\mathcal{D}_2$ consisting of two dynamic causal laws
$$a\ \textbf{causes}\ f\ \textbf{if}\ g \qquad a\ \textbf{causes}\ f\ \textbf{if}\ \neg g$$
Intuitively, we have that a is a conformant plan achieving f from $\emptyset$ because either g or $\neg g$ is true in any state belonging to $comp(\emptyset)$. Yet, it is easy to verify that a cannot be generated by CPASP due to the fact that $\pi(\mathcal{D}_2, 1) \cup h(\emptyset, 0) \cup \{o(a, 0)\}$ has a unique answer set containing no atom of the form $h(l, 1)$.   □

The next example shows that it is not only conditional effects but also static causal laws can cause T^{lp} to be incomplete.

---

[4] If we wish to find a sequential plan, the only thing needed to do is to change the left side of the rule to $1\{o(A, T) : action(A)\}1$.

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

Example 3. Consider the action description $\mathcal{D}_3$ consisting of the following laws

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

We can check that a is a solution to the problem $\mathcal{P}_3 = \langle \mathcal{D}_3, \{\neg f, \neg g\}, \{g\} \rangle$ since a causes f to hold and the two static causal laws guarantee that if f holds then so does g. Yet, neither $h(h, 1)$ nor $h(\neg h, 1)$ will belong to any answer set of $\pi(\mathcal{P}_3, 1)$ due to the rules (10) and (11). As such, $\pi(\mathcal{P}_3, 1)$ does not return an answer set, i.e., a cannot be found using $T^{lp}(\mathcal{D}_3)$. □

5 Experiments

We ran CPASP on both SMODELS and Cmodels [16]. In general, Cmodels yields better performance. The results reported in this paper are the times obtained using Cmodels. Since most answer set solvers do not scale up well to programs that require large grounded representation, we also implemented the approximation in a C++ planner, called CPA^{ph}([28]). CPA^{ph} employs a best-first search strategy with the number of fulfilled subgoals as its heuristic function. Unlike CPASP, the current version of CPA^{ph} does not compute concurrent plans. However, CPA^{ph} allows disjunctions to be specified in the initial state description, while CPASP does not. Thus, CPASP cannot solve conformant planning benchmarks in the literature where the initial state specification contains disjunctions. We consider this as one of the weaknesses of CPASP.

We compare CPASP (and CPA^{ph}) with three other conformant planners CMBP[9], DLV^k[12], and $\mathcal{C}$-PLAN[8] because these planners do allow static causal laws and are similar in spirit of 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). While the latter two allow concurrent planning, the former does not. A comparison between DLV^K and other planners like SGP [25] and GPT [5] can be found in [12]. For a comparison between CPA^{ph} and other state-of-the-art conformant planners like Conformant-FF [6], KACMBP [10], and POND [7], we refer the reader to [28].

We prepared two test suites: one contains sequential, conformant planning benchmarks and the other contains concurrent, conformant planning benchmarks.

The first test suite includes two typical planning domains, the well-known Bomb-in-the-toilet and the $Ring$ domains [10]. In the former, we consider two variants, $BMT(n, p)$ and $BMTC(n, p)$, where n and p are the numbers of packages and toilets respectively. The first one is without clogging and the second one is with clogging. The uncertainty in the initial state is that we do not know whether or not packages are disarmed. In the Ring domain, one can move in a cyclic fashion (either forward or backward) around a n-room building to lock windows. Each room has a window and the window can be locked only if it is closed. Initially, the robot is in the first room and it does not know the state (open/closed) of the windows. The goal is to have all windows locked. A possible conformant plan is to perform a sequence of actions *forward, close, lock* repeatedly. In this domain, we tested with $n \in \{2, 4, 6, 8, 10\}$.

These domains, however, do not contain many static causal laws. Therefore, we introduce two new domains, called $Domino$ and $Gaspipe$. The former is very simple. We have n dominos standing on a line in such a way that if one of them falls then the domino on its right also falls. There is a ball hanging close to the leftmost one. Touching

the ball causes the first domino to fall. Initially, the states of dominos are unknown. The goal is to have the rightmost one to fall. The solution is obviously to touch the ball. In this domain, we tested with $n \in \{100, 200, 500, 100, 2000, 5000, 10000\}$.

The $Gaspipe$ domain is a little more complicated. We need 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 of the pipeline and the burner is on the right-most. The pipe line contains sections that connect with each other by valves. The state of pipe sections can be either pressured or unpressured. Opening a valve causes the section on its right side to be pressured if the section on its left is pressured. Moreover, to be safe, 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 unpressured. There are two kinds of static causal laws. The first one is that if a valve is open and the section on its left is pressured then the section on its right will pressured. Otherwise (either the valve is closed or the section on the left is unpressured), the pipe on the right side is unpressured. The burner will start a flame if the pipe connecting to it is pressured. The gas tank is always pressured. The uncertainty we introduce with the initial situation is that the states of valves are unknown. A possible conformant plan will be closing all valves but the first one (that is, the one that connects to the gas tank), in the right-to-left order and then opening them in the reverse order. We tested with $n \in \{3, 5, 7, 9, 11\}$.

The last domain in the first test suite is the $Cleaner$ domain. It is a modified version of the Ring domain. The difference is that instead of locking the window, the robot has to clean objects. Each room has p objects to be cleaned. Initially, the robot is at the first room and does not know whether or not objects are cleaned. The goal is to have all objects cleaned. While the Domino and Gaspipe domains expose a richness in static causal laws, the Cleaner domain provides a high degree of uncertainty in the initial state. We tested the domain with 6 problems where $n \in \{2, 5\}$ and $p \in \{10, 50, 100\}$ respectively.

The second test suite includes benchmarks for concurrent, conformant planning. It contains four domains. The BMT^p and $BMTC^p$ domains are variants of BMT and $BMTC$ in the first test suite in which dunking different packages into different toilets at the same time is allowed. The $Gaspipe^p$ is a modification of $Gaspipe$ in which closing multiple valves at the time are allowed. In addition, one can 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 $Cleaner$ domain is relaxed to allow cleaning multiple objects in the same room at the same time. The relaxed version is denoted by $Cleaner^p$. The testing problems in the second test suite are the same as those in the first test suite.

All experiments were made on a 2.4 GHz CPU, 768MB RAM machine, running Slackware 10.0 operating system. Time limit is set to half an hour. The testing results for two test suites are shown in Tables 1a) and 1b) respectively. We did not test $\mathcal{C}$-PLAN in the sequential planning benchmarks since it is supposed to use for concurrent planning[6]. Times are shown in seconds; "PL", "TO", "MEM", "NA" indicate the length of the plan found by the planner, that the planner ran out of time, that the planner ran out of memory, and that the planner returns a message indicating that no plan can be found[7], respectively. Since both DLV^K and CPASP require as an input parameter the length of

---

[6] The authors told us that $\mathcal{C}$-PLAN was not intended for searching sequential plans.

[7] We did contact the authors' of the planner for help and are waiting for a response.

Table 1. Comparison between CPASP, CPA^{ph}, CMBP DLV^K, and $\mathcal{C}$-PLAN in sequential DLV^K, and $\mathcal{C}$-PLAN in sequential a) Sequential Benchmarks b) Concurrent Benchmarks

Domains	CMBP		DLV^K		CPASP		CPA^{ph}	
Problems	PL	Time	PL	Time	PL	Time	PL	Time
$BMT(2,2)$	2	0.03	2	0.046	2	0.209	2	0.000
$BMT(4,2)$	4	0.167	4	0.555	4	0.418	4	0.002
$BMT(6,2)$	6	0.206	6	216.557	6	0.775	6	0.005
$BMT(8,4)$	8	0.633		TO	8	6.734	8	0.021
$BMT(10,4)$	10	1.5		TO	10	890.064	10	0.038
$BMTC(2,2)$	2	0.166	2	0.121	2	0.222	2	0.001
$BMTC(4,2)$	6	0.269	6	72.442	6	0.712	6	0.004
$BMTC(6,2)$	10	0.749		TO	8	2.728	10	0.010
$BMTC(8,4)$		TO		TO		TO	12	0.031
$BMTC(10,4)$		TO		TO		TO	16	0.054
$Gaspipe(3)$		NA	5	0.132	5	1.349	7	0.026
$Gaspipe(5)$		NA	9	0.425	9	2.226	22	0.481
$Gaspipe(7)$		NA	13	42.625	13	6.186	86	8.464
$Gaspipe(9)$		NA		TO	17	39.323	261	45.910
$Gaspipe(11)$		NA		TO	21	868.102	1327	529.469
$Cleaner(2,2)$	5	0.1	5	0.104	5	0.496	5	0.002
$Cleaner(2,5)$	11	0.617	11	214.696	11	3.88	11	0.012
$Cleaner(2,10)$		TO		TO		TO	21	0.060
$Cleaner(4,2)$	11	0.13	11	14.82	11	2.094	11	0.014
$Cleaner(4,5)$		TO		TO		TO	23	0.082
$Cleaner(4,10)$		TO		TO		TO	43	0.434
$Cleaner(6,2)$	17	4.1		TO	17	224.391	17	0.054
$Cleaner(6,5)$		TO		TO		TO	35	0.311
$Cleaner(6,10)$		TO		TO		TO	65	1.623
$Ring(2)$	5	0.01		0.201	5	0.911	5	0.003
$Ring(4)$	11	0.116		0.638	11	2.738	12	0.025
$Ring(6)$	17	0.5		TO	17	18.852	18	0.088
$Ring(8)$		TO		TO	23	669.321	24	0.242
$Ring(10)$		TO		TO		TO	30	0.542
$Domino(100)$	1	0.26	1	0.1	1	0.216	1	0.026
$Domino(200)$	1	1.79	1	0.352	1	0.285	1	0.099
$Domino(500)$	1	7.92	1	2.401	1	0.747	1	0.568
$Domino(1000)$	1	13.2	1	13.104	1	1.236	1	2.313
$Domino(2000)$	1	66.6	1	62.421	1	2.414	1	9.209
$Domino(5000)$	1	559.467		MEM	1	6.076	1	67.619
$Domino(10000)$		TO		MEM	1	12.584	1	350.129

a)

Domains	$\mathcal{C}$-PLAN		DLV^K		CPASP	
Problems	PL	Time	PL	Time	PL	Time
$BMT^P(2,2)$	1	0.078	1	0.074	1	0.116
$BMT^P(4,2)$	2	0.052	2	0.094	2	0.268
$BMT^P(6,2)$	3	1.812	3	3.065	3	0.346
$BMT^P(8,4)$	2	4.32	2	10.529	2	0.248
$BMT^P(10,4)$		TO		TO	3	1.911
$BMTC^P(2,2)$	1	0.057	1	0.059	1	0.13
$BMTC^P(4,2)$	3	0.076	3	0.908	3	0.3
$BMTC^P(6,2)$	5	7.519	5	333.278	5	0.672
$BMTC^P(8,4)$		TO		TO	3	0.508
$BMTC^P(10,4)$		TO		TO	5	1192.458
$Gaspipe^P(3)$		TO	4	0.088	4	0.402
$Gaspipe^P(5)$		TO	6	0.173	6	0.759
$Gaspipe^P(7)$		TO	8	0.441	8	1.221
$Gaspipe^P(9)$		TO	10	17.449	10	3.175
$Gaspipe^P(11)$		TO		TO	12	8.832
$Cleaner^P(2,2)$	3	0.052	3	0.076	3	0.265
$Cleaner^P(2,5)$	3	0.121	3	0.066	3	0.3
$Cleaner^P(2,10)$	3	0.06	3	0.076	3	0.309
$Cleaner^P(4,2)$	7	0.068	7	0.196	7	0.773
$Cleaner^P(4,5)$	7	0.09	7	0.809	7	0.931
$Cleaner^P(4,10)$	7	0.131	7	237.637	7	1.164
$Cleaner^P(6,2)$	11	0.116	11	4.475	11	1.982
$Cleaner^P(6,5)$	11	0.195	11	986.731	11	2.947
$Cleaner^P(6,10)$	11	0.357		TO	11	3.737

b)

a plan to search for, we ran them by incrementally increasing the plan length, starting from 1[8], until a plan is found.

As can be seen in Table 1a), in the BMT and $BMTC$ domains, CMBP outperforms both DLV^K and CPASP in most problems. However, its performance is not competitive with CPA^{ph} which can solve the $BMTC(10,4)$ with only less than one tenth of a second (In fact, CPA^{ph} can scale up to larger problems, e.g., with 100 packages and 100 toilets, within the time limit). CPASP in general has better performance than DLV^K in these domains. As an example, DLV^K took more than three minutes to solve the $BMT(6,2)$, while it took only 0.775 seconds for CPASP to solve the same problem. Within the time limit, CPASP is able to solve more problems than DLV^K.

CPASP seems to work well with domains rich in static causal laws like $Domino$ and $Gaspipe$. In the $Domino$ domain, CPASP outperforms all the other planners in most of instances. It took only 2.414 seconds to solve $Domino(2000)$, while both DLV^K and CMBP took more than one minute. Although CPA^{ph} can solve all the instances in this domain, its performance is in general worse than CPASP's. In the $Gaspipe$ domain, CPASP and CPA^{ph} are competitive with each other and outperform the other two. The $Cleaner$ domain turns out to be quite hard for the tested planners except CPA^{ph}. We believe that the high degree of uncertainty in the initial state is the main reason for this performance gain of CPA^{ph} comparing to others since it does not consider *all possible cases* in searching for a solution.

CPASP is outperformed by both CMBP and DLV^K in some small instances in the $Ring$ domain. However, it can solve the $Ring(8)$, while CMBP and DLV^K cannot.

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

Again, CPA^{ph} is the best. This shows that CPASP can be competitive with the tested conformant planners in some sequential planning benchmarks.

Table 1b) shows that CPASP also has a fairly good performance in concurrent planning problems. It outperforms both DLV^{K} and $\mathcal{C}$-PLAN in most instances in the BMT^p, $BMTC^p$, and $Gaspipe^p$ domains. DLV^{K} is better than $\mathcal{C}$-PLAN in the $Gaspipe^p$ domain. On the contrary, $\mathcal{C}$-PLAN is very good at the $Cleaner^p$ domain. To solve $Cleaner$ $(6, 10)$, $\mathcal{C}$-PLAN took only 0.357 seconds , whereas DLV^{K} ran out of time and CPASP needs 3.737 seconds.

6   Conclusion and Future Work

We present a logic programming based approximation for $\mathcal{AL}$ action descriptions and apply it to conformant planning. We describe two conformant planners, CPASP and CPA^{ph}, whose key reasoning part is for computing the approximation. Our initial experiments show that with an appropriate approximation, logic programming based conformant planners can be built to deal with problems rich in static causal laws and incomplete information about the initial state. In other words, a careful study in approximated reasoning may pay off well in the development of practical planners.

As an approximation can only guarantee soundness, it will be interesting to characterize situations when an approximation (e.g. $T^{lp}(\mathcal{D})$) can yield completeness. For example, if $T^{lp}(\mathcal{D})$ can generate all conformant plans of length 1 and whenever $\langle a_1, \ldots, a_n \rangle$ is a solution to $\langle \mathcal{D}, s, s^f \rangle$, $\langle a_2, \ldots, a_n \rangle$ is a solution to $\langle \mathcal{D}, s', s^f \rangle$ where $s' = \bigcap_{\exists \sigma \in comp(s).\langle \sigma, a_1, \sigma' \rangle \in T(\mathcal{D})} \sigma'$, then $T^{lp}(\mathcal{D})$ is complete. It can be shown that the first condition can be met when $\mathcal{D}$ does not contain *(i)* a static causal law; *(ii)* a pair of dynamic causal laws of the form a **causes** f **if** p and a **causes** f **if** p' with $p' \cap \bar{p} \neq \emptyset$; and *(iii)* a pair of impossibility conditions of the form **impossible** a **if** p and **impossible** a **if** p' with $p' \cap \bar{p} \neq \emptyset$. Identifying sufficient conditions for the completeness of $T^{lp}(\mathcal{D})$ will be our main concern in the near future.

At this point, we would like to mention that to verify that our approach can deal with a broad spectrum of planning problems, we tested CPASP and CPA^{ph} with several benchmarks problems [1] including the instances of the Blocks World domain tested in [12] and did not encounter a problem that the two planners cannot solve. This shows that our approach can deal with a large class of practical planning problems. Finally, we would like to point out that the use of logic programming allows us to easily exploit control knowledge (e.g., "*do not dunk a package unless it is armed*") in improving the quality of a plan or to specify complex initial (incomplete)-states (see e.g., [13,27]).

References

1. F. Bacchus. The AIPS'00 Planning Competition. *AI Magazine*, 22(3), 2001.
2. C. Baral. Reasoning about Actions : Non-deterministic effects, Constraints and Qualification. In *IJCAI'95*, 2017–2023.

3. 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.

4. 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.

5. B. Bonet and H. Geffner. GPT: a tool for planning with uncertainty and partial information. In *IJCAI-01 Workshop on Planning with Uncertainty and Partial Information*, 82–87, 2001.

6. R. Brafman and J. Hoffmann. Conformant planning via heuristic forward search: A new approach. In *(ICAPS-04)*, 355–364.

7. D. Bryce and S. Kambhampati. Heuristic Guidance Measures for Conformant Planning. In *(ICAPS-04)*, 365–375, 2004.

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

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

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

11. 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/.

12. T. Eiter, W. Faber, N. Leone, G. Pfeifer, and A. Polleres. A Logic Programming Approach to Knowledge State Planning, II: The DLV$^\mathcal{K}$ System. *AIJ*, 144(1-2):157–211, 2003.

13. M. Gelfond and R. Morales. Encoding conformant planning in a-prolog. DRT'04.

14. M. Ghallab et al. PDDL — the Planning Domain Definition Language. Version 1.2. Technical Report CVC TR98003/DCS TR1165, Yale Center for Comp, Vis and Ctrl, 1998.

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

16. Y. Lierler and M. Maratea. Cmodels-2: SAT-based Answer Set Solver Enhanced to Non-tight Programs. In LPNMR'04, 346–350, LNCS 2923, 2004.

17. V. Lifschitz. On the Logic of Causal Explanation (Research Note). *AIJ*, 96(2):451–465.

18. F. Lin. Embracing causality in specifying the indirect effects of actions. IJCAI'95, 1985–93.

19. D. Long and M. Fox. The 3rd International Planning Competition: Results and Analysis. JAIR, 20:1–59, 2003.

20. N. McCain & H. Turner. A causal theory of ramifications and qualifications. IJCAI'95, 1978–1984.

21. N. McCain & M. Turner. Causal theories of action and change. AAAI'97, 460–467.

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

23. M. Shanahan. The ramification problem in the event calculus. In IJCAI'99, 140–146, 1999.

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

25. D. Smith and D. Weld. Conformant graphplan. In *Proceedings of AAAI 98*, 1998.

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

27. T.C. Son, C. Baral, N. Tran, and S. McIlraith. Domain-Dependent Knowledge in Answer Set Planning. To appear in TOCL.

28. T.C. Son, P.H. Tu, M. Gelfond, and R. Morales. Conformant Planning for Domains with Constraints — A New Approach. To Appear in AAAI'05.

29. S. Thiebaux, J. Hoffmann, and B. Nebel. In Defense of PDDL Axioms. IJCAI'03.

30. H. Turner. Polynomial-length planning spans the polynomial hierarchy. JELIA'02, 111–124.