Safe Learning of PDDL Domains with Conditional Effects

Argaman Mordoch[1], **Roni Stern**[1], **Enrico Scala**[2], **Brendan Juba**[3]

[1]Ben Gurion University
[2]Università degli Studi di Brescia
[3]Washington University in St. Louis
mordocha@post.bgu.ac.il, sternron@bgu.ac.il, enricos83@gmail.com, bjuba@wustl.edu

Abstract

Influential domain-independent planners have been developed to solve various types of planning problems. These planners often require a model of the acting agent's actions, given in some planning domain description language. Yet obtaining such an action model is a notoriously challenging task. This task is even more challenging in mission-critical domains, where a trial-and-error approach to learning how to act is not an option. In such domains, the action model used to generate plans must be *safe* in the sense that plans generated with it must be applicable and achieve their goals. Previous works on safe action model learning addressed domains with non-conditional effects. In this work, we propose the algorithm Safe Action Model Learning of Conditional Effects (Conditional-SAM) that can learn action models with conditional effects. We show that the challenge of learning conditional effects may require an exponential number of samples and present a case study to demonstrate the capabilities of Conditional-SAM.

Introduction

Planning is the fundamental task of choosing the right actions to achieve the desired outcome. An *automated domain-independent planner* refers to an Artificial Intelligence (AI) algorithm capable of solving a wide range of planning problems (Ghallab, Nau, and Traverso 2016). Developing a domain-independent planner is a long-term goal of AI research. Researchers developed many domain-independent planners for various types of planning problems. Such planners include Fast Downward (Helmert 2006), Fast Forward (Hoffmann 2001), COLIN (Coles et al. 2009) and more. These planners often require a model of the acting agent's actions, given in some domain description language (e.g., the Planning Domain Definition Language (**PDDL**) (Ghallab et al. 1998)). Defining an agent's *action model* to solve real-world problems is extremely hard. Researchers acknowledged this modeling challenge and algorithms for learning agents' action models from observations have been proposed (Cresswell and Gregory 2011; Aineto, Celorrio, and Onaindia 2019; Yang, Wu, and Jiang 2007; Juba, Le, and Stern 2021). Since the learned model may differ from the domain's actual action model, using it to plan raises two challenges: *safety* and *completeness*. The *safety* risk is that the learned model may generate a plan that cannot be applied in the domain or may not reach a state that satisfies the problem goals. The *completeness* risk is that the learned model may be too restrictive to generate plans for solving solvable problems. This work focuses on safety, a crucial property in mission-critical domains where a trial-and-error approach for learning how to act or online replanning are not viable options. In such domains, the action model used to generate plans must be *safe*, i.e., create plans that must be applicable and achieve their goals. The Safe Action Model Learning (SAM) family of algorithms (Stern and Juba 2017; Juba, Le, and Stern 2021; Juba and Stern 2022) addresses the challenge of learning safe action models for planning. But, these algorithms do not learn planning action models where actions might include *conditional effects*. A conditional effect is an effect that occurs only when a specific condition holds, which is not necessary to the action's execution. Some researchers claim that conditional effects can be compiled out of the agents' action models, but in practice, supporting them is important. Nebel (2000) claimed that conditional effects cannot be compiled away without a polynomial increase in the plan size. Compiling them away requires adding additional actions for each conditional effect because conditional effects enable the execution of multiple effects in parallel.

Moreover, we do not control the domain model used to generate the trajectories in the learning task, and this model may contain conditional effects. Learning these conditional effects is challenging since they might not be observed upon action execution. Even if they are observed, determining the cause for these effects may require many observations. Indeed, we show that for some domains, the task inherently requires an exponential number of example trajectories (Theorem 3).

Previous works on action model learning with conditional effects (Oates and Cohen 1996; Zhuo et al. 2010) made no safety guarantees for the learned models. This work addresses this gap by exploring the problem of learning safe action models for PDDL (Ghallab et al. 1998) domains with conditional effects. Specifically, we focus on PDDL domains with conjunctive preconditions and antecedents without existential quantifiers. For domains that satisfy these requirements, we introduce the Safe Action Model Learn-

ing of Conditional Effect algorithm, i.e., Conditional-SAM, which outputs an action model such that plans generated using it are guaranteed to be successful. We show that Conditional-SAM learns the correct action model with an asymptotically optimal number of trajectories when the size of the antecedents for the conditional effects is restricted, which is the only case where the problem is tractable. We also present an extension of Conditional-SAM that supports *lifted domains* and *universally quantified conditional effects*.

Together with our theoretical analysis of the sample complexity of learning conditional effects, we also demonstrate the usefulness of our algorithm's capabilities in practice. We focus on a domain from the international planning competition and show that with a small number of input observations, Conditional-SAM learns an action model that can be used, combined with a sound and complete planner to solve every test set problem.

Preliminaries

We recall the terminology and statement of our problems.

The Planning Problem

We focus on planning problems in domains where action outcomes are deterministic, the states are fully observable and contain Boolean variables only. Such problems are commonly modeled using a fragment of the ADL (Action Description Language) (Pednault 1989) and formulated in PDDL (Planning Domain Definition Language) (Ghallab et al. 1998), which consists of a domain and a problem description. A PDDL domain is defined by a tuple $D = \langle F, C, A \rangle$ where F is a finite set of Boolean variables, referred to as fluents (or predicates); C is a set of domain constants (might be empty); and A is a set of actions. An action model M for a set of actions $\mathcal{A}$ is a pair of functions pre_M and eff_M that map every action in $\mathcal{A}$ to its preconditions and effects.

Recall that a *literal* refers to either a fluent or its negation. A *state* of the world is a set of literals that includes, for every fluent f, either f or $\neg f$. The preconditions of an action a, $pre_M(a)$ is a set of literals that must be satisfied before applying a. If no preconditions are specified, the action is always executable. The effects of an action a, $eff_M(a)$ specify the outcome of applying a. A *conditional effect* is tuple $\langle c, e \rangle$ where c is the antecedent (condition), and e is the result (effect); both c and e are conjunctions of literals. The semantics of a conditional effect $\langle c, e \rangle$ is that if the antecedent c holds before the action is applied, then the result e occurs. Conditional effects can be viewed as a generalization of effects that are not conditional by specifying the trivial antecedent *true* to such conditions. The outcome of applying a to a state s according to action model M, denoted $a_M(s)$, is a new state in which every literal has its value as in state s except those literals changed by the actions' effects. Formally, the next state $s' = a_M(s)$ satisfies the following rules:

1. $\forall (c, e) \in eff_M(a) : (c \wedge s \nvdash \bot) \rightarrow e \in s'$
2. $\forall l \in s, \nexists (c, e) \in eff_M(a) :$
   $((c \wedge s \nvdash \bot) \wedge (\neg l \in e)) \rightarrow l \in s'$

A PDDL problem is defined by a tuple $P = \langle I, G, D \rangle$ where I is the initial state, containing all the literals that represent the state of all objects in the problem; G is the set of fluents that define the goal state, and D is the domain description. An action a is *applicable* in a state s if s satisfies the literals in $pre(a)$. Applying a in s, denoted $a(s)$, results in a state that differs from s only according to the assignments in $eff(a)$. A *plan* $\Pi = \langle a_1, a_2, ...a_n \rangle$ is a sequence of actions. We say that a plan Π is *valid* if a_1 is applicable in I and $G \subseteq a_n(a_{n-1}(...(a_1(I))))$.

The Learning Problem

There are different approaches to learning action models. In this work, we focus on learning action models from observations of previously executed plans represented as a set of *trajectories*.

Definition 1 (Trajectory)**.** *A trajectory $T = \langle s_0, a_1, s_1, \ldots a_n, s_n \rangle$ is an alternating sequence of states $(s_0, \ldots, s_n)$ and actions $(a_1, \ldots, a_n)$ that starts and ends with a state.*

The trajectory created by applying Π to a state s is the sequence $\langle s_0, a_1, \ldots, a_{|\Pi|}, s_{|\Pi|} \rangle$ such that $s_0 = s$ and for all $0 < i \leq |\Pi|$, $s_i = a_i(s_{i-1})$. In prior work (Wang 1994, 1995; Walsh and Littman 2008; Stern and Juba 2017; Arora et al. 2018; Aineto, Celorrio, and Onaindia 2019) a trajectory $\langle s_0, a_1, \ldots, a_{|\Pi|}, s_{|\Pi|} \rangle$ is often represented as a set of *action triplets* $\{ \langle s_{i-1}, a_i, s_i \rangle \}_{i=1}^{|\Pi|}$.

This work focuses on learning action models for actions with conditional effects. Conditional effects complicate the learning process. Given an action model with $|L|$ literals, the number of possible antecedents for every conditional effect grows *exponentially* in the number of literals $|L|$. Thus, if $|L|$ is too large, the problem becomes intractable.

Safe Action Models We follow prior work by Juba et al. (2021) and require that the learned action model is *safe*, where safe here means that any plan generated with the learned action model is also executable with the real, unknown action model.

Definition 2 (Safe Action Model (Juba, Le, and Stern 2021))**.** *An action model M' is safe with respect to an action model M if for every state s and action a it holds that*

$$pre_{M'}(a) \subseteq s \rightarrow (pre_M(a) \subseteq s \wedge a_{M'}(s) = a_M(s)) \quad (1)$$

In other words, M' is safe w.r.t. M if for every state s and action a, if a is applicable in s according to M' then (1) it is also applicable in s according to M, and (2) applying a in s results in exactly the same state according to both M and M'.

Approach

We present our approach to safely learn conditional effects.

Assumptions

We make the following simplifying assumptions:

1. Actions' preconditions and conditional effects' antecedents are conjunctions of literals.

2. For each literal l', there is *at most* one antecedent that results in achieving l'. Formally, for each literal l'

$$((c,e) \in \textit{eff}(a) : l' \in e) \rightarrow (\nexists (c',e') \in \textit{eff}(a) : l' \in e' \wedge c \neq c') \quad (2)$$

The first assumption is made by many action model learning algorithms (Aineto, Celorrio, and Onaindia 2019; Juba, Le, and Stern 2021; Yang, Wu, and Jiang 2007) and does not extensively reduce the number of suitable domains. The second assumption is needed to guarantee the complexity properties made later in the paper.

In addition, we assume that the maximal number of literals in an antecedent is at most n, which is a fixed parameter known in advance. We prove later that without this assumption, learning conditional effects is intractable. From a practical point of view, the last assumption requires a human modeler to specify the maximal number of literals in an antecedent for the domain, which is still significantly easier than manually defining the entire action model.

Conditional-SAM Inductive Rules

Conditional-SAM learns an action model by applying the following rules:

Definition 3 (Conditional-SAM Inductive Rules)**.** *For every action triplet $\langle s, a, s' \rangle \in \mathcal{T}$ it holds that:*

1. *For every literal $l \notin s$, then $l \notin \textit{pre}(a)$*
2. *For every literal $l' \notin s'$ then $\nexists (c,e) \in \textit{eff}(a)$ where $(c \wedge s \nvdash \bot) \wedge (l' \in e)$*
3. *For every literal $l' \in s' \setminus s$ then $\exists (c,e) \in \textit{eff}(a) : (c \wedge s \nvdash \bot) \wedge (l' \in e)$*
4. *For every literal $l' \in s' \setminus s$ and set of literals c such that $c \wedge s \vdash \bot$, then $\nexists (c',e) \in \textit{eff}(a)$ such that $c \subseteq c'$ and $l' \in e$.*

The first rule indicates which literals cannot be preconditions for an action. The second rule indicates which literals cannot be considered a conditional effect of an observed action. The third rule indicates which literals must be effects of an observed action. Every literal l' observed in $s' \setminus s$ is guaranteed to be an effect of a and there exists $(c,e) \in \textit{eff}(a)$ such that c is consistent with s and $l' \in e$. Note that the first rule also exists in the original SAM learning algorithm. The second and third rules correspond to similar rules in the original SAM learning but extend them to support conditional effects. The fourth induction rule is derived from Assumptions 1 and 2. From Assumption 1, we know that the antecedents for conditional effects consist of conjunctions of literals. Thus, for each conjunction of literals c that is not satisfied in s, then every conjunction c' that contains c will also not be satisfied in s. Furthermore, Assumption 2 states that there are no disjunctive conditional effects. Thus, if we observe l' as the result of the action, any conjunction of literals c that is not satisfied in s cannot be the antecedent of the conditional effect for l'.

Example 1. *Consider a domain with 3 fluents f_1, f_2, and f_3, where the size of antecedents is bounded by 1 (i.e., n=1), and assume to represent a state by a Boolean vector of size 3. Now, assume we observed an action triplet $\langle (T,T,F), a, (F,T,F) \rangle$. Using the first inductive rule, we infer that $\neg f_1$, $\neg f_2$, and f_3 are not preconditions of a. By using the second inductive rule, a cannot have an effect (c,e) such that c is consistent with $f_1 \wedge f_2 \wedge \neg f_3$ and the result is either f_1, $\neg f_2$, or f_3. Since $n = 1$, this rules out the conditional effects where c is one of the following sets $\{true\}, \{f_1\}, \{f_2\}, \{\neg f_3\}$ and e is either $\{f_1\}$ or $\{\neg f_2\}$ or $\{f_3\}$, e.g., $(c,e) = (\{f_1\}, \{\neg f_2\})$. According to the third inductive rule either $\neg f_1$ is a non-conditional effect of a or $(\{f_1\}, \{\neg f_1\})$, $(\{f_2\}, \{\neg f_1\})$, or $(\{\neg f_3\}, \{\neg f_1\})$ are conditional effects of a. Finally, according to the fourth inductive rule $(\{\neg f_2\}, \{\neg f_1\})$ and $(\{f_3\}, \{\neg f_1\})$ cannot be conditional effects.*

Conditional-SAM Algorithm

Let $A(\mathcal{T}), L(\mathcal{T})$ be the set of actions and literals observed in the trajectories $\mathcal{T}$. The Conditional-SAM algorithm maintains three data structures: *pre*, *PosAnte*, and *MustBeResult*. *pre* is the set of literals considered as the preconditions of the action. It is initialized to all the literals $l \in L(\mathcal{T})$. Conditional-SAM removes unnecessary preconditions according to Rule 1 in Def. 3. *PosAnte* maintains, for every action a and literal l, a set of conjunctions of literals, representing the conjunction of literals that may be antecedents of a conditional effect of a, which results in l. This data structure is initialized to include every subset of literals of size n or less. Conditional-SAM removes elements from *PosAnte* using Rules 2 and 4 in Def. 3. *MustBeResult* maintains, for every action a, the set of literals observed to be the result of the action a. This data structure is initialized as an empty set and populated using Rule 3 in Def. 3.

We present the main functionality of Conditional-SAM in Algorithm 1. For each action $a \in A(\mathcal{T})$, Conditional-SAM starts by initializing *pre*, *PosAnte* and *MustBeResult* as specified above. Conditional-SAM then applies the inductive rules and compiles the action model for each action triplet $\langle s, a, s' \rangle \in \mathcal{T}$ to update and populate *pre*, *PosAnte*, and *MustBeResult* as specified above.

After populating *pre*, *PosAnte*, and *MustBeResult*, Conditional-SAM compiles the complete action model using the function *LearnConditions*. The function *LearnConditions* starts by initializing $\textit{eff}(a)$ for each action $a \in A(\mathcal{T})$ to an empty set and $\textit{pre}^*(a)$ to be the observed preconditions of the action. Then, for every literal l that can be an effect of the action, the function creates the data structure PA, which contains all subsets of possible antecedents in *PosAnte* which are disjoint from $\textit{pre}(a)$.

Using PA, the function creates *NotAnte*, which is the conjunction of the negations of the sets of literals in PA. In contrast to *NotAnte*, the data structure *Ante* is the conjunction of all the possible antecedents in PA.

After creating the above conjunctions, the function verifies whether $l \in \textit{MustBeResult}(a)$. If so, the tuple $\langle \textit{Ante}, l \rangle$ is added to $\textit{eff}(a)$. If PA consists of more than a single set of antecedents, then there is an ambiguity on which antecedent is the cause of l. Thus, *LearnConditions* adds the or clause, $(l \vee \textit{NotAnte} \vee \textit{Ante})$, composed of three parts as follows: First, allowing the action to be applicable if the result, l, is observed in the pre-state. Second, the action is permitted if

none of the antecedents hold in the pre-state. Lastly, a is applicable if all the antecedents hold in the pre-state. If one of the above holds, the action will be executed.

In case l was not observed as a result of the action, the function adds $(l \vee \textit{NotAnte})$ to $\textit{pre}^*(a)$. Since we have yet to observe l as a result of the action, we cannot guarantee that it is *not* an effect. Since $\langle \textit{Ante}, l \rangle \notin \textit{eff}(a)$, to maintain the action's safety, we want to guarantee that l will not appear in the post-state. Thus, we only permit the execution of the action in states that guarantee that l will not be the result of a. Note that if $\textit{NotAnte} \subseteq pre(a)$, then l will never result from applying a. Finally, *LearnConditions* returns the learned preconditions and effects of the observed actions.

Algorithm 1: Conditional-SAM Algorithm

Input: $\mathcal{T}, n$
Output: A safe action model.
for $a \in A(\mathcal{T})$ **do**
  $\textit{pre}(a) \leftarrow L(\mathcal{T})$
  $\textit{MustBeResult}(a) \leftarrow \emptyset$
  $\textit{PosAnte}(l, a) \leftarrow \bigcup_{i=1}^{n} L(\mathcal{T})_i \cup \{\textit{true}\}$
end for
for $\langle s, a, s' \rangle \in \mathcal{T}$ **do**
  for l s.t. $l \notin s$ **do**
    $pre(a) \leftarrow pre(a) \setminus \{l\}$ ▷ Rule 1
  end for
  for $l \in s' \setminus s$ **do** ▷ Rule 3
    $\textit{MustBeResult}(a) \leftarrow \textit{MustBeResult}(a) \cup \{l\}$
  end for
  for $l' \notin s'$ and $c \in \textit{PosAnte}(l', a)$ s.t. $(c \wedge s \nvdash \bot)$ **do**
    $\textit{PosAnte}(l', a) \leftarrow \textit{PosAnte}(l', a) \setminus c$
  end for ▷ Rule 2
  for $l' \in s' \setminus s$ and $c \in \textit{PosAnte}(l', a)$ s.t. $c \wedge s \vdash \bot$ **do** ▷ Rule 4
    $\textit{PosAnte}(l', a) \leftarrow \textit{PosAnte}(l', a) \setminus c$
  end for
end for
return LearnConditions$(\textit{pre}, \textit{MustBeResult}, \textit{PosAnte})$

Example 2. *Given a domain with 3 literals and an action a where $\textit{pre}(a) = \emptyset$, $l_1 \in \textit{MustBeResult}(a)$, $\textit{PosAnte}(a, l_1) = \{\{l_2\}, \{l_3\}\}$, $\textit{PosAnte}(a, l_2) = \emptyset$, and $\textit{PosAnte}(a, l_3) = \emptyset$. The resulting preconditions and effects after applying LearnConditions are $\textit{pre}^*(a) = (l_1) \vee (\neg l_2 \wedge \neg l_3) \vee (l_2 \wedge l_3)$ and $\textit{eff}(a) = (l_2 \wedge l_3, l_1)$.*

Note that while we assume that all the actions in the real-world model do not support disjunctive antecedents (Assumption 2), the safe action model we create does allow such actions. This highlights that the safe action model is not necessarily equivalent to the real action model, but as we show later, it is guaranteed to be safe with respect to it.

Theorem 1. *The action model M' learned by Conditional-SAM is safe w.r.t the action model that generated the input trajectories $\mathcal{T}$*

Proof. Conditional-SAM learns a superset of the original actions model's preconditions. Thus, for each action a and state s such that a is applicable according to M', it is guaranteed to be applicable according to M^*.

Algorithm 2: Conditional Effects Learning Function

Input: *pre*, *MustBeResult*, *PosAnte*
Output: *pre* and *eff* for all actions.
for $a \in A(\mathcal{T})$ **do**
  $\textit{eff}(a) \leftarrow \emptyset$
  $\textit{pre}^*(a) \leftarrow \textit{pre}(a)$
  for $l \in L(\mathcal{T}) \setminus \textit{pre}(a)$ where $\textit{PosAnte}(l, a) \neq \emptyset$ **do**
    $PA \leftarrow \{c \in \textit{PosAnte}(l, a) | (\textit{pre}(a) \cap c) = \emptyset\}$
    $\textit{NotAnte} \leftarrow \bigwedge_{c \in PA} \neg c$
    $\textit{Ante} \leftarrow \bigwedge_{c \in PA} c$
    if $l \in \textit{MustBeResult}(a)$ **then**
      Add to $\textit{eff}(a)$: $(\textit{Ante}, l)$
      if PA is not a single clause **then**
        Add to $\textit{pre}^*(a)$: $(l \vee \textit{NotAnte} \vee \textit{Ante})$
      end if
    else
      Add to $\textit{pre}^*(a)$: $(l \vee \textit{NotAnte})$
    end if
  end for
end for
return $\langle \textit{pre}^*, \textit{eff} \rangle$

Given an action a and a state s in which a is applicable according to M', the resulting state $s' = a_{M'}(s)$, is equivalent to $s'^* = a^*_M(s)$, we prove this by contradiction. Assume that $s' \neq s'^*$. There can be two possibilities: (1) $\exists l \in s'$ such that $l \notin s'^*$, or (2) $\exists l \notin s'$ such that $l \in s'^*$. Since a is applicable in s according to M', it is also applicable according to M^*. Since Conditional-SAM only adds effects observed in the trajectories, according to rule 3, there cannot be a literal l such that (1) holds. If $l \in s'^*$ but $l \notin s'$, then Conditional-SAM did not observe l as a result of a and thus did not add it as an effect. According to line 16 one of $(l \vee \textit{NotAnte})$ hold in s. If $l \in s$, then $l \in s'$ according to M' (since a does not remove it), which contradicts (2). Similarly, if $\textit{NotAnte} \subseteq s$, then the antecedent of l according to M^* is negated in s thus $l \notin s'^*$ which also contradicts (2). Thus Conditional-SAM learns a safe action model with respect to M^*. □

Theoretical Analysis

In this section, we theoretically analyze the complexity of the Conditional-SAM algorithm, proving that under a fixed antecedent size (n) its space, runtime, and sample complexity are tractable, and show that our sample complexity bound is tight.

Space and Runtime Complexity

Consider the space complexity of Conditional-SAM. For every action $a \in A$ and every literal $l \in L$, Conditional-SAM maintains the data structures $\textit{pre}(a)$, $\textit{PosAnte}(l, a)$ and $\textit{MustBeResult}(a)$. The size of $\textit{pre}(a)$ is at most $|L|$. The size of $\textit{MustBeResult}(a)$ is also at most $|L|$. The size of $\textit{PosAnte}(l, a)$ is observed when it is initialized, containing every conjunction of literals of size at most n, including the empty set (representing the antecedent *true*). Thus, the size of $\textit{PosAnte}(l, a)$ is at most $\sum_{i=0}^{n} \binom{|L|}{i} \leq \left(\frac{|L| \cdot e}{n}\right)^n$. We note that the space complexity of *LearnConditions* is linear in the

size of *PosAnte*. Consequentially, the space complexity is

$$|A||L| + |A||L| + |A||L| \sum_{i=0}^{n} \binom{|L|}{i} \in O\left(|A||L|^{n+1} \left(\frac{e}{n}\right)^n\right) \tag{3}$$

Recall that n is a fixed constant — the maximal number of literals in an antecedent.

Next, we analyze the runtime complexity of Conditional-SAM. The initialization process requires the same runtime complexity as its space complexity, i.e., $O(|A||L|^{n+1}\left(\frac{e}{n}\right)^n)$. Then, Conditional-SAM iterates over all action triplets and applies the inductive rules in Def. 3. This requires $O(|\mathcal{T}||L|^{n+1}\left(\frac{e}{n}\right)^n)$, since as discussed above, the size of *PosAnte*(l,a) is at most $|L|^n\left(\frac{e}{n}\right)^n$.

Finally, in the *LearnConditions* function, the runtime complexity is bounded by the most intensive computational part, which is the part that creates the restrictive conditions. The complexity of this part is linear in *PosAnte*(l,a). Thus the total runtime complexity of *LearnConditions* is bounded by $O\left(|A||L|^n\left(\frac{e}{n}\right)^n)\right)$ Thus, the total runtime complexity of the algorithm is $O\left(|A||L|^n\left(\frac{e}{n}\right)^n) + |\mathcal{T}||L|^{n+1}\left(\frac{e}{n}\right)^n)\right)$.

Sample Complexity

Theorem 2. *Let $\mathcal{D}$ be a distribution over pairs $\langle P, \Pi\rangle$ where P is a problem from a fixed domain D and Π is a plan solving P. Given $m \geq \frac{1}{\epsilon}\left(\ln(3)|F||A| + 2\ln(2)|F||A|\left(\frac{2|F|e}{n}\right)^n + \ln\frac{1}{\delta}\right)$ trajectories (where e is the base of the natural logarithm) obtained by executing Π for m independent draws from $\mathcal{D}$, Conditional-SAM returns an action model M' such that with probability $1-\delta$, for a new P drawn from $\mathcal{D}$, the probability that there exists a plan in M' solving P is at least $1-\epsilon$.*

Proof. In view of Theorem 1, it suffices to show that for a pair $\langle P, \Pi\rangle$ drawn from $\mathcal{D}$, the preconditions of Π in M' are satisfied for each step of the execution of Π in the real action model M^*; indeed, the states obtained by M' and M^* are identical, so Π will then also solve P in M'.

Recall that Conditional-SAM passes the sets *pre*, *MustBeResult*, and *PosAnte* for each action a and, in the case of *PosAnte*, for each literal l to Algorithm 2. A literal l only appears in *pre*(a) for an action a if $\neg l$ has never been observed in the pre-state when action a was taken. Similarly, a clause $\neg c$ may appear as (a subclause of) some clause of the precondition *pre** of a in M' if c remains in the antecedents set *PosAnte*(e,a) of some candidate effect $e \in$ *MustBeResult*(a) for which more than one such candidate remains, or for which $e \notin$ *MustBeResult*(a) and c is in *PosAnte*(e,a). Note that if $\neg c$ is falsified in a state s (prohibiting a in s in M'), $s \subseteq c$. Hence, if the execution of Π in M^* would result in a being taken in s resulting in s', Conditional-SAM would remove c from *PosAnte*(e,a) for all $e \notin s'$, and c from *PosAnte*(e,a) if $c \not\subseteq s$ and $e \in s' \setminus s$.

We now claim that the probability that Conditional-SAM obtains a set of antecedents *PosAnte*(l,a) and set of preconditions *pre*(a) that prohibits the execution of Π with probability greater than ϵ is at most δ: a is only prohibited by *pre** in s if (1) $l \in s$ for some $\neg l \in pre(a)$; if (2) $s \subseteq c$ for some $c \in$ *PosAnte*(l,a) where $l \notin$ *MustBeResult*(a) and $\neg l \in s$; or, if (3) $l \in$ *MustBeResult*(a), $\neg l \in s$, $s \subseteq c$ for some $c \in$ *PosAnte*(l,a), and $s \not\subseteq c'$ for some (other) $c' \in$ *PosAnte*(l,a). When the execution of Π includes taking such an action a in such a prohibited state s, in the first case we see Conditional-SAM removes the falsified $\neg l$ from *pre*(a); for every effect e of a in s, any $c \not\subseteq s$ are removed from *PosAnte*(e,a) so cases (2) and (3) cannot occur; and for every literal $\tilde{e}$ that is not an effect of a in s, since $\tilde{e} \notin s'$, all $(c,\tilde{e})$ for $c \subseteq s$ are removed from *PosAnte*$(\tilde{e},a)$, so neither case (2) nor (3) can occur. Thus, we see that either at least one literal is deleted from *pre* or at least one c is deleted from some *PosAnte*(e,a) when such an (s,a,s') occurs in the trajectory, so that a is permitted by *pre*$^*(a)$ in s subsequently. Since literals are only deleted from *pre* and clauses are only deleted from *PosAnte*, Conditional-SAM then cannot return the eliminated collection of preconditions and antecedents sets.

Quantitatively, for any collection of preconditions and antecedents sets for which such a problem and plan would be obtained from $\mathcal{D}$ with probability greater than ϵ, Conditional-SAM can only return the corresponding collection with probability at most $(1-\epsilon)^m$ when it is given m examples drawn independently from $\mathcal{D}$. Observe that there are $3^{|F|}$ possible sets *pre* for each $a \in A$, and $2^{\sum_{k=0}^{n} 2^k\binom{|F|}{k}}$ possible sets *PosAnte* for each l and a. Thus, there are at most

$$3^{|F||A|}2^{2|F||A|\sum_{k=0}^{n}2^k\binom{|F|}{k}} \leq e^{\ln(3)|F||A| + 2\ln(2)|F||A|\left(\frac{2|F|e}{n}\right)^n}$$

possible collections of pre and *PosAnte*. Since $(1-\epsilon)^m \leq e^{-m\epsilon}$, taking a union bound over all possible collections of *pre* and *PosAnte* that prohibit the execution of the associated plan with probability at least ϵ, we find that for the given m, the total probability of Conditional-SAM obtaining such a collection of preconditions and antecedents sets is at most δ. Thus, with probability $1-\delta$, the action model indeed permits executing the plans associated with problems drawn from $\mathcal{D}$ with probability at least $1-\epsilon$ as needed. □

Conditional-SAM therefore enjoys approximate completeness with high probability so long as the number of training trajectories is sufficiently large. The one unsatisfying aspect of our bound is that the number of trajectories is exponential in the size of the antecedents of the conditions in the conditional effects we consider. Unfortunately, we find that this is unavoidable and our bound is asymptotically optimal (for any fixed n) for safe action model learning for domains with conditional effects:

Theorem 3. *Any learning algorithm that is guaranteed to return a safe action model must be given at least $m \geq \Omega(\frac{1}{\epsilon}(|F||A|(|F|/3n)^n| + \log\frac{1}{\delta}))$ samples to be able to guarantee that with probability at least $1-\delta$ the learned model permits a plan solving Π drawn from $\mathcal{D}$ with probability at least $1-\epsilon$ for $0 < \epsilon, \delta < 1/4$.*

Proof. For any $p \geq 3|A|$, consider a domain in which there is a no-op action with no effects, and for each other action $a_i \in A$ there is a fluent f_i called a *goal fluent* that is

the effect of exactly one action, and this is the only effect. The domain includes an additional set of $(p - |A|)/2$ fluents called *flag fluents*, and $(p - |A|)/2$ fluents (so there are $|A| + 2(p - |A|)/2 = p$ fluents in total) called *forbidden fluents*.

Now consider the following distributions on problems and plans. The initial state of every problem sampled from $\mathcal{D}$ has all goal fluents set to false, all but one (uniformly random) forbidden fluent true, and exactly n of the flag fluents (uniformly at random) true. With probability $1 - 4\epsilon$, the goal is empty. Otherwise, the goal includes a single goal fluent, chosen uniformly at random, that should be set to true. All other goal fluents, as well as the one forbidden fluent, must be set to false. The corresponding Π always consists of a plan with a single action; for the empty goal, the agent takes the no-op action, and otherwise the agent takes the action corresponding to the f_i goal fluent to be set true in the goal.

For any problem with a non-empty goal that we did not observe in the training set, the action model that is obtained from the true action model by adding the forbidden fluent as a conditional effect of the corresponding goal action with the flag fluents as the condition, is consistent with the training set. Indeed, either the action appears with a different set of flags so that one of the flag fluents in this condition is falsified (and the corresponding effect does not occur), a different forbidden fluent is false (so the relevant forbidden fluent is already true and the effect is not observed), or else the action differs from the one we need to achieve this goal, and then the effect is identical to the true action model. Therefore, no safe action model can permit taking the action needed to achieve the goal, and all other actions would reach a state in which some incorrect goal fluent is set to true and cannot be subsequently set to false.

Since the no-op goal only comprises $1 - 4\epsilon$ probability in the goal distribution, we need to observe at least a $3/4$ fraction of the possible goals for a safe action model to attain probability $1-\epsilon$. But, there are $|A|$ goals, $(p-|A|)/3 \geq p/3$ forbidden fluents, and $\binom{(p-|A|)/3}{n} \geq \left(\frac{p}{3n}\right)^n$ sets of flags, and in expectation, a sample of size m only contains $4\epsilon m$ examples of these pairs of goals and flag settings. We, therefore, need $\Omega\left(\frac{1}{\epsilon}(|F|/3)^n|F||A|\right)$ examples; likewise, to even observe any of the nonempty goals with probability $1 - \delta$, we need $\Omega\left(\frac{1}{\epsilon}\log\frac{1}{\delta}\right)$ examples, giving the claimed bound. □

Lifted Domains and Universal Quantifiers

It is common to describe PDDL domains and problems in a *lifted* manner. In lifted domains, actions and fluents are defined by a tuple, e.g., $\langle \textit{name}(a), \textit{params}(a)\rangle$, where the parameters $\textit{params}(a)$ have *types*, e.g., *truck* and *location*, giving a lifted action or fluent like so: *move(?t - truck, ?from - location, ?to - location)* and *at(?t - truck, ?loc - location)*. A state is a conjunction of *grounded fluents*, which are pairs of the form $\langle l, b_l\rangle$ where l is a fluent, and b_l is a function that maps parameters of l to concrete objects. A plan is a sequence of *grounded actions*, which are pairs in the form $\langle a, b_a\rangle$ where a is an action and b_a maps action parameters to objects. A trajectory is an alternating sequence of states and grounded actions.

Generally, the parameters in an action's preconditions and effects are bound to the action's parameters. Thus, preconditions and effects of an action in a lifted domain are *parameter-bound literals*. A parameter-bound literal for an action a is a pair (l, b_{la}) where l is a literal and b_{la} is a function that maps every parameter of l to a parameter in a. Let $\textit{bindings}(a)$ be the function that returns all parameter-bound literals that can be bound to a. For a grounded action $a_G = \langle a, b_a\rangle$ and parameter-bound literal $l \in \textit{bindings}(a)$, we define $g(a_G, l)$ to be the grounded literal resulting from assigning the objects in the parameters of a_G to the parameters of l. Given a conjunction of parameter-bound literals c, $g(a_G, c)$ returns the corresponding conjunction of grounded literals c_G such that $\forall l \in c : g(a_G, l) \in c_G$. Similarly, for a pair of conjunctions of parameter-bound literals (c, e) we define $g(a_G, c, e)$ to be the pair (c_G, e_G) that are the corresponding conjunctions of grounded literals. SAM learning has already been extended to learn lifted classical planning domains (Juba, Le, and Stern 2021) without conditional effects. Conditional-SAM may be extended to lifted domains similarly, based on the following extension to the Conditional-SAM inductive rules (Def. 3).

Definition 4 (Lifted Conditional-SAM Inductive Rules)**.** *For every action triplet $\langle s, a_G = \langle a, b_a\rangle, s'\rangle \in \mathcal{T}$,*

1. *For every $l \in \textit{bindings}(a)$ s.t. $g(a_G, l) \notin s$, $l \notin \textit{pre}(a)$*
2. *For every $l' \in \textit{bindings}(a)$ s.t. $g(a_G, l') \notin s'$, $\nexists (c, e) \in \textit{eff}(a)$ where $(g(a_G, c) \wedge s \nvdash \bot) \wedge (l' \in e)$*
3. *For every $l' \in \textit{bindings}(a)$, if $g(a_G, l') \in s' \setminus s$ then $\exists (c, e) \in \textit{eff}(a)$, where $(g(a_G, c) \wedge s \nvdash \bot) \wedge (l' \in e)$*
4. *For every $l' \in \textit{bindings}(a)$ and set of literals $c \subseteq \textit{bindings}(a)$ if $g(a_G, l') \in s' \setminus s$ and $g(a_G, c) \wedge s \vdash \bot$ then $\nexists (c', e) \in \textit{eff}(a)$ such that $c \subseteq c'$ and $l' \in e$*

Some PDDL domains and planners support *domain constants* and *universal quantifiers*, which allow actions' preconditions and effects to include additional parameters that are not bound to the actions' parameters. Domain constants are objects defined in the PDDL domain; they substantially extend the set of objects defined in a PDDL problem when linked with a particular domain. These objects can be bound to any parameter of a precondition or effect. For example, in the *Logistics* domain, we can define the constant $base$ that represents the base location of the trucks. Support for constants is relatively simple: the $\textit{bindings}(a)$ function will now return an extended set of parameter-bound literal that includes binding the literal parameters to domain constants.

Universally quantified effects define one or more *universally quantified variables (UQV)* that may be bound to any parameter of a literal used in them. We note that the result of a universally quantified conditional effect *must* include at least one UQV. Otherwise, if only the antecedents include UQVs, then we can interpret such antecedents as disjunctive universal preconditions. To clarify the formulation of universal effects, suppose that in our Logistics domain, we add the functionality to unload all packages in the truck to their current location. Figure 1 presents the *unload-multi* action schema to implement this functionality.

Not all planners support universal effects, but universal effects are common in benchmark domains that include con-

```
(:action unload-multi
:parameters (?truck-truck ?loc-location)
:precondition (at ?truck ?loc)
:effect
    (forall (?p - package)
        (when (in ?p ?truck)
        (and (not (in ?p ?truck)) (at ?p ?loc)))))
```

Figure 1: Example *unload-multi* action with universal effects.

ditional effects. We briefly describe how Conditional-SAM can be extended to support universal effects.

In general, the number of UQVs a universal effect may define is exponential in the arity of the domain fluents. Still, universal effects with more than two UQV are rare. Thus, we will assume the number of UQVs in a universal effect is a known fixed constant k. To support universal effects, the *bindings*(a) function is modified to also return parameter-bound literals that bind one or more literal parameters to UQVs that may be used in action a's effects. Similarly, the $g(a_G, l)$ function is modified such that if l is a parameter-bound literal that includes UQVs then $g(a_G, l)$ returns a *set* of grounded literals matching the grounded action's parameters combined with the UQVs. In addition, $g(a_G, c, e)$ returns a set of matching pairs (c_G, e_G) if either c or e include one or more UQVs. We now present the changes in the inductive rules to support universally quantified variables.

Definition 5 (Conditional-SAM Inductive Rules with UQVs)**.** *For every action triplet* $\langle s, a_G = \langle a, b_a \rangle, s' \rangle \in \mathcal{T}$*:*

1. *For every* $l \in$ *bindings*(a) *such that* $\exists l_G \in g(a_G, l)$ *where* $l_G \notin s$*, then* $l \notin$ *pre*(a)
2. *For every* $l' \in$ *bindings*(a) *such that* $\exists l'_G \in g(a_G, l')$ *and* $l'_G \notin s'$ *then* $\nexists (c, e) \in$ *eff*(a) *such that* $\exists (c_G, e_G) \in g(a_G, c, e)$ *where* $(c_G \wedge s \nvdash \bot) \wedge (l'_G \in e_G)$
3. *For every* $l' \in$ *bindings*(a) *if* $\exists l'_G \in g(a_G, l')$ *such that* $l'_G \in s' \setminus s$ *then* $\exists (c, e) \in$ *eff*(a)*, where* $\exists (c_G, e_G) \in g(a_G, c, e)$ *such that* $c_G \wedge s \nvdash \bot \wedge l'_G \in e_G$
4. *For every* $l' \in$ *bindings*(a) *and* $c \subseteq$ *bindings*(a) *if* $\exists (l'_g, c_G) \in g(a_G, c, l')$ *such that* $l'_G \in s' \setminus s$ *and* $c_G \wedge s \vdash \bot$ *then* $\nexists (c', e) \in$ *eff*(a) *such that* $c \subseteq c'$ *and* $l' \in e$

The rest of the Conditional-SAM algorithm remains essentially the same, where *MustBeResult* and *PosAnte* may now contain parameter-bound literals that include UQVs.

Case Study

To empirically show the functionality of Conditional-SAM, we implemented the algorithm and conducted experiments on a single domain that contains conditional effects. The experimented domain is an ADL version of the *Satellite* domain available in the classic IPC (Long and Fox 2003) benchmark. We sampled a set of trajectories in the domain, ran Conditional-SAM on the training trajectories, and attempted to solve the test set problems in that domain using the learned action model. To solve the test set problems, we used the state-of-the-art solver Fast-Downward (Helmert 2006). We then validated the resulting plans using VAL (Howey, Long, and Fox 2004). We followed a 5-fold cross-validation methodology by repeating each experiment 5 times, sampling different trajectories for learning and testing. The experiments were run on a Linux machine with eight cores and 16 GB of RAM.

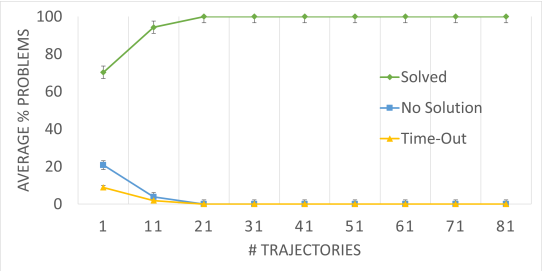

Figure 2: Fraction of problems solved for the Satellite domain.

In Figure 2, we present the average fraction of the problems solved as a function of the number of input trajectories used to learn the action model. We observe that when Conditional-SAM received a single trajectory as its input, the learned action model was able to solve 70% of the test set problems, 21% had no solution, and for 9%, the solver timed out. When Conditional-SAM used 31 trajectories, all of the test set problems were solved.

Related Work

Several prior works learn action models from trajectories. The Action-Relation Modelling System (ARMS) (Yang, Wu, and Jiang 2007) algorithm learns a PDDL description of action models by extracting a set of weighted constraints, from the given trajectories, over the preconditions and effects of the available actions. The Simultaneous Learning and Filtering (SLAF) (Amir and Chang 2008) algorithm is a different algorithm for learning action models designed for partially observable deterministic domains. The Learning Object-Centred Models (LOCM, LOCM2) (Cresswell, McCluskey, and West 2013; Cresswell and Gregory 2011) is another action model learning algorithm that analyzes plan sequences, where each action appears as an action name and arguments in the form of a vector of object names. FAMA (Aineto, Celorrio, and Onaindia 2019) is a state-of-the-art algorithm that improves the performance of LOCM. It can learn action models when the observability of the actions is minimal. FAMA can learn from gapped action sequences of actions, and in the extreme, FAMA can even learn when only given the initial and the final states as input.

The algorithms presented above learn action models that supply no guarantee that the actions learned are applicable according to the agent's actual action model definition. Contrary to these algorithms, the SAM family of algorithms is designed to learn action models in a setting where execution failures must be avoided (2017; 2021; 2022). To this end, SAM generates a conservative action model. This approach produces *sound* but may be *incomplete*.

To the best of our knowledge, there is no work with the main focus of learning safe action models with conditional

effects. In (Oates and Cohen 1996), the authors created an algorithm that can learn planning operators for STRIPS (Fikes and Nilsson 1971) by interacting with the environments and performing random actions and using search techniques to learn the context-dependent operators. This approach uses random walks which are costly in case that the agent cannot recover from failures. Furthermore, the resulting action model generated is grounded while our approach learns a lifted PDDL domain. In (Zhuo et al. 2010), the authors' main focus was learning action models with quantifiers and implications. The authors also proved that their algorithm could learn simple conditional effects with only one antecedent. Since the authors' goal is to *reduce* the domain compilation time for domain experts, the domains their algorithm outputs may be incomplete or even wrong. Similarly, SLAFS (Shahaf and Amir 2006) learns action models with conditional effects that are **consistent** with its observations. This means that these algorithms do not work in mission-critical settings.

Conclusions and Future Work

In this work, we presented Conditional-SAM, an algorithm that can learn action models for domains that include conditional and universal effects. We showed that Conditional-SAM learns a safe action model w.r.t the real unknown action model and presented sample complexity theory for the algorithm. In a preliminary case study, we showed that Conditional-SAM learns an action model that can solve all test set problems with a small number of input trajectories.

As future works, we have two main objectives: first, we aim to explore methods to improve the algorithm's scalability and support domains with more expressive conditional effects that might contain unbounded disjunctive antecedents or even include numeric conditions and effects. Second, we aim to extend our experimental evaluation of Conditional-SAM over a large set of planning problems.

References

Aineto, D.; Celorrio, S. J.; and Onaindia, E. 2019. Learning action models with minimal observability. *Artificial Intelligence*, 275: 104–137.

Amir, E.; and Chang, A. 2008. Learning partially observable deterministic action models. *Journal of Artificial Intelligence Research*, 33: 349–402.

Arora, A.; Fiorino, H.; Pellier, D.; Etivier, M.; and Pesty, S. 2018. A review of learning planning action models. *Knowledge Engineering Review*, 33.

Coles, A.; Coles, A.; Fox, M.; and Long, D. 2009. Temporal planning in domains with linear processes. In *International Joint Conference on Artificial Intelligence (IJCAI)*.

Cresswell, S.; and Gregory, P. 2011. Generalised domain model acquisition from action traces. In *International Conference on Automated Planning and Scheduling (ICAPS)*, 42–49.

Cresswell, S.; McCluskey, T.; and West, M. 2013. Acquiring planning domain models using LOCM. *The Knowledge Engineering Review*, 28(2): 195–213.

Fikes, R. E.; and Nilsson, N. J. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3-4): 189–208.

Ghallab, M.; Howe, A.; Knoblock, C.; McDermott, D.; Ram, A.; Veloso, M.; Weld, D.; and Wilkins, D. 1998. PDDL – The Planning Domain Definition Language. *Technical Report, Tech. Rep.*

Ghallab, M.; Nau, D.; and Traverso, P. 2016. *Automated planning and acting*. Cambridge University Press.

Helmert, M. 2006. The Fast Downward planning system. *Journal of Artificial Intelligence Research*, 26: 191–246.

Hoffmann, J. 2001. FF: The fast-forward planning system. *AI magazine*, 22(3): 57–57.

Howey, R.; Long, D.; and Fox, M. 2004. VAL: Automatic plan validation, continuous effects and mixed initiative planning using PDDL. In *16th IEEE International Conference on Tools with Artificial Intelligence*, 294–301. IEEE.

Juba, B.; Le, H. S.; and Stern, R. 2021. Safe Learning of Lifted Action Models. In *International Conference on Principles of Knowledge Representation and Reasoning (KR)*, 379–389.

Juba, B.; and Stern, R. 2022. Learning Probably Approximately Complete and Safe Action Models for Stochastic Worlds. In *AAAI Conference on Artificial Intelligence*.

Long, D.; and Fox, M. 2003. The 3rd international planning competition: Results and analysis. *Journal of Artificial Intelligence Research*, 20: 1–59.

Nebel, B. 2000. On the Compilability and Expressive Power of Propositional Planning Formalisms. *J. Artif. Intell. Res.*, 12: 271–315.

Oates, T.; and Cohen, P. R. 1996. Learning planning operators with conditional and probabilistic effects. In *Proceedings of the AAAI Spring Symposium on Planning with Incomplete Information for Robot Problems*, 86–94.

Pednault, E. P. 1989. Adl: Exploring the middle ground between strips and the situation calculus. In *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning (KR'89)*, 324–332.

Shahaf, D.; and Amir, E. 2006. Learning partially observable action schemas. In *Proceedings of the National Conference on Artificial Intelligence*, volume 21, 913. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999.

Stern, R.; and Juba, B. 2017. Efficient, Safe, and Probably Approximately Complete Learning of Action Models. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 4405–4411.

Walsh, T. J.; and Littman, M. L. 2008. Efficient learning of action schemas and web-service descriptions. In *AAAI Conference on Artificial Intelligence (AAAI)*, volume 8, 714–719.

Wang, X. 1994. Learning planning operators by observation and practice. In *Second International Conference on Artificial Intelligence Planning Systems (AIPS)*, 335–340.

Wang, X. 1995. Learning by observation and practice: an incremental approach for planning operator acquisition. In *International Conference on Machine Learning (ICML)*, 549–557.

Yang, Q.; Wu, K.; and Jiang, Y. 2007. Learning action models from plan examples using weighted MAX-SAT. *Artificial Intelligence*, 171(2-3): 107–143.

Zhuo, H. H.; Yang, Q.; Hu, D. H.; and Li, L. 2010. Learning complex action models with quantifiers and logical implications. *Artificial Intelligence*, 174(18): 1540–1569.