Safe Learning of Lifted Action Models*

Brendan Juba [1] , **Hai S. Le**[1] , **Roni Stern**[2,3]
[1]Washington University in St. Louis, USA
[2]Palo Alto Research Center, USA
[3]Ben Gurion University of the Negev, Israel
{bjuba, hsle}@wustl.edu, rstern@parc.com, sternron@post.bgu.ac.il

Abstract

Creating a domain model, even for classical, domain-independent planning, is a notoriously hard knowledge-engineering task. A natural approach to solve this problem is to learn a domain model from observations. However, model learning approaches frequently do not provide safety guarantees: the learned model may assume actions are applicable when they are not, and may incorrectly capture actions' effects. This may result in generating plans that will fail when executed. In some domains such failures are not acceptable, due to the cost of failure or inability to replan online after failure. In such settings, all learning must be done offline, based on some observations collected, e.g., by some other agents or a human. Through this learning, the task is to generate a plan that is guaranteed to be successful. This is called the model-free planning problem. Prior work proposed an algorithm for solving the model-free planning problem in classical planning. However, they were limited to learning grounded domains, and thus they could not scale. We generalize this prior work and propose the first safe model-free planning algorithm for lifted domains. We prove the correctness of our approach, and provide a statistical analysis showing that the number of trajectories needed to solve future problems with high probability is linear in the potential size of the domain model. We also present experiments on twelve IPC domains showing that our approach is able to learn the real action model in all cases with at most two trajectories.

1 Introduction

In classical domain-independent planning, a *domain model* is a model of the environment and how the acting agent can interact with it. The domain model is given in a formal planning description language such as STRIPS (Fikes and Nilsson 1971) or the Planning Domain Definition Language (PDDL) (McDermott et al. 1998). Domain-independent planning algorithms (planners) use the domain model to generate a plan for achieving a given goal condition from a given initial state. Creating a domain model, however, is a notoriously hard knowledge-engineering task.

To overcome this modeling problem, a variety of learning methods have been proposed. Model-free Reinforcement Learning (RL) avoids the need for a domain model by learning directly how to act by performing actions and observing their outcomes. Other learning approaches aim to learn a world model from past observations, and use that model to solve future planning problems (Amir and Chang 2008). Notably, Asai and Muise (2020) recently demonstrated this approach can even learn a PDDL model directly from (non-symbolic) images. However, all these approaches **permit the generation of failing actions**, i.e., actions that are either not applicable in the current state or do not achieve the intended effects. In some domains, this is acceptable and the agent simply incorporates such experiences and updates its internal model to improve future executions. In other domains, however, failing actions must be avoided and only safe actions are allowed. This occurs when execution failure is too costly, or the agent cannot replan due to limited computational capabilities. The problem of finding a safe plan, i.e., a plan that will not fail, without possessing a domain model, is called **safe model-free planning** (Stern and Juba 2017). In safe model-free planning, instead of a domain model the planning agent is given a set of trajectories from plans that were executed in the past in the same domain (e.g., by a different agent or a human).

Stern and Juba (2017) proposed a sound algorithm for safe model-free planning, i.e., an algorithm that generates plans that do not fail, provided that the environment is actually captured by a (grounded) STRIPS model. However, their algorithm is not complete, i.e., it may not return a plan for a solvable planning problem. Nevertheless, they proposed a PAC-style model of learning to plan, in which completeness may be relaxed to "approximate completeness" with respect to the distribution of problems observed during training. They thus bounded the probability of encountering problems their model cannot solve, given a number of trajectories quasi-linear in the number of actions. However, their positive result is limited to *grounded* domain models, that is, domains that are not defined by *lifted*, i.e., parameterized, actions and fluents. The size of a grounded domain model can be arbitrarily larger than its corresponding lifted domain model. In particular, a single lifted action can yield a number of grounded actions that grow polynomially with the number of objects in the domain, with the number of parameters of the lifted action as its exponent. In addition, learning a grounded domain model limits the generalization possible between different groundings of the same lifted domain. For example, a grounded action model for a blocksworld do-

*This is the KR 2021 proceedings version. See the associated technical report arXiv:2107.04169 [cs.AI] for full proofs.

main with 8 blocks cannot be used to solve problems for a blocksworld domain with 9 blocks. This significantly limits the applicability of Stern and Juba's algorithm.

In this work, we overcome these limitations by presenting an algorithm that efficiently solves safe model-free planning problems for lifted domains. The key component of this approach is an algorithm that learns a *safe action model*, which is a model of the agent's possible actions that is consistent with the underlying, unknown, domain model. We call this algorithm *Safe Action Model (SAM)* Learning.

Two versions of SAM learning are presented. The first may be used when each object is only ever bound to one action parameter at a time in the example trajectories. We prove that this version is sound, and when the actions and fluents have bounded arity, we can guarantee that the action model is sufficient with high probability after observing a number of trajectories that is linear in the possible size of the lifted model. Importantly, the number of trajectories needed depends only on the size of this lifted model, and is independent of the number of objects in the domain, in contrast to Stern and Juba's algorithm. We also observed efficient learning experimentally on twelve domains from the International Planning Competition (IPC) (McDermott 2000): SAM learning is able to learn the real action model for all cases with at most two trajectories. Finally, we discuss a more general version of SAM learning, for the case where multiple arguments are bound to the same object in some trajectories.

Our work also revisits the algorithm of Stern and Juba, and shows that it can be interpreted as solving a kind of knowledge-based learning task, similar to *inductive logic programming* (Muggleton and De Raedt 1994), using the STRIPS axioms as background knowledge. We show in particular that the obtained model is the action model with the largest possible set of feasible plans (i.e., least constrained) that can be proven safe with the given trajectories, and in this sense is the *strongest* safe action model. We show that our algorithms for lifted domains also enjoy this property.

2 Background and Problem Definition

Let O be a set of objects and let T be a set of types. Every object $o \in O$ is associated with a type $t \in T$ denoted *type*(o). For example, in the logistics domain from the International Planning Competition (IPC) (McDermott 2000) there are types *truck* and *location* and there may be objects t_1 and t_2 that represent two different trucks and two objects l_1 and l_2 that represent two different locations.

2.1 Lifted and Grounded Literals

A *lifted fluent* F is a pair $\langle$*name*, *params*$\rangle$ representing a relation over typed objects, where *name* is a symbol and *params* is a list of types. We denote the name of F and its parameters by *name*(F) and *params*(F) respectively, and $arity(F, t)$ denotes the number of type-t parameters. For example, in the logistics domain $at(?truck, ?location)$ is a lifted fluent that represents which trucks ($?truck$) are at which locations ($?location$). A *binding* of a lifted fluent F is a function b : *params*$(F) \rightarrow O$ mapping every parameter of F to an object in O of the indicated type. A *grounded fluent* f is a pair $\langle F, b\rangle$ where F is a lifted fluent and b is a binding for F. To *ground* a lifted fluent F with a binding b means to create a (Boolean-valued) fluent with a value determined by whether or not the objects in the image of b satisfy the relation associated with the lifted fluent. In our logistics example, for $F = at(?truck, ?location)$ and $b = \{?truck : truck1, ?location : loc1\}$ the corresponding grounded fluent f is $at(truck1, loc1)$, indicating whether $truck1$ is at $loc1$. The term *literal* refers to either a fluent or its negation. The definitions of binding, lifted, and grounded fluents transfer naturally to literals. A *state* of the world is a set of grounded literals that, for every grounded fluent, either includes that fluent or its negation.

2.2 Lifted and Grounded Actions

A lifted action $A \in \mathcal{A}$ is a pair $\langle$*name*, *params*$\rangle$ where *name* is a symbol and *params* is a list of types, denoted *name*(A) and *params*(A), respectively, and $arity(A, t)$ denotes the number of type-t parameters. The 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. To define the preconditions and effects of a lifted action, we first define the notion of a *parameter-bound literal*. A *parameter binding* of a lifted literal L and an action A is a function $b_{L,A}$: *params*$(L) \rightarrow$ *params*(A) that maps every parameter of L to a parameter in A. A *parameter-bound literal* l for the lifted action A is a pair of the form $\langle L, b_{L,A}\rangle$ where $b_{L,A}$ is a parameter binding of L and A. $pre_M(A)$ and $eff_M(A)$ are sets of parameter-bound literals for A.

A *binding* of a lifted action A is defined like a binding of a lifted fluent, i.e., a function b : *params*$(A) \rightarrow O$. A *grounded action* a is a tuple $\langle A, b_A\rangle$ where A is a lifted action and b_A is a binding of A. The preconditions of a grounded action a according to the action model M, denoted $pre_M(a)$, is the set of grounded literals created by taking every parameter-bound literal $\langle L, b_{L,A}\rangle \in pre_M(A)$ and grounding L with the binding $b_A \circ b_{L,A}$. The effects of a grounded action a, denoted $eff_M(a)$, are defined in a similar manner. The grounded action a can be applied in a state s iff $pre_M(a) \subseteq s$. The outcome of applying a to a state s according to action model M, denoted $a_M(s)$, is a new state that contains all literals in $eff_M(a)$ and all the literals in s such that their negation is not in $eff_M(a)$. Formally:

$$a_M(s) = \{l | (l \in s \wedge \neg l \notin eff_M(a)) \vee l \in eff_M(a)\} \quad (1)$$

We omit M from $a_M(s)$ when it is clear from the context. The outcome of applying a sequence of grounded actions $\pi = (a_1, \ldots a_n)$ to a state s is the state $s' = a_n(\cdots a_1(s)\cdots)$. A sequence of actions $a_1, \ldots, a_n$ can be applied to a state s if for every $i \in 1, \ldots, n$ the action a_i is applicable in the state $a_{i-1}(\cdots a_1(s)\cdots)$.

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 the literature on learning

action models (Wang 1994; Wang 1995; Walsh and Littman 2008; Stern and Juba 2017; Arora et al. 2018), it is common to represent a trajectory $\langle s_0, a_1, \ldots, a_{|\pi|}, s_{|\pi|} \rangle$ as a set of triples $\{ \langle s_{i-1}, a_i, s_i \rangle \}_{i=1}^{|\pi|}$. Each triple $\langle s_{i-1}, a_i, s_i \rangle$ is called an *action triplet*, and the states s_{i-1} and s_i are referred to as the pre- and post- state of action a_i. We denote by $\mathcal{T}(a)$ the set of all action triplets in the trajectories in $\mathcal{T}$ that include the grounded action a. $\mathcal{T}(A)$ is defined for all action triplets that contain actions that are groundings of the lifted action A.

2.3 Domains and Problems

A classical planning **domain** is defined by a tuple $\langle T, \mathcal{F}, \mathcal{A}, M \rangle$ where T is a set of types, $\mathcal{F}$ is a set of lifted fluents, $\mathcal{A}$ is a set of lifted actions, and M is an action model for $\mathcal{A}$. A classical planning **problem** is defined by a tuple $\langle D, O, s_I, G \rangle$ where D is a classical planning domain; O is a set of objects; s_I is the start state, i.e., the state of the world before planning; and G is a set of grounded literals that define when the goal has been found. A **solution** to a planning problem is a sequence of grounded actions that can be applied to s_I and if applied to s_I results in a state s_G that contains all the grounded literals in G. Such a sequence of grounded actions is called a *plan*. The trajectory of a plan starts with s_I and ends with a goal state s_G (where $G \subseteq s_G$). The *safe model-free planning* problem (Stern and Juba 2017) is defined as follows.

Definition 2 (Safe model-free planning)**.** *Let* $\Pi = \langle \langle T, \mathcal{F}, \mathcal{A}, M^* \rangle, O, s_I, G \rangle$ *be a classical planning problem and let* $\mathcal{T} = \{\mathcal{T}_1, \ldots, \mathcal{T}_m\}$ *be a set of trajectories for other planning problems in the same domain. The input to a safe model-free planning algorithm is the tuple* $\langle T, O, s_I, G, \mathcal{T} \rangle$ *and the desired output is a plan* π *that is a solution to* Π*. We denote this safe model-free planning problem as* $\Pi_{\mathcal{T}}$*.*

We refer to the action model M^* as the real action model. The trajectories in $\mathcal{T}$ share the same domain as Π, and thus they have been generated by applying actions from $\mathcal{A}$ and following the action model specified in M^*. However, these trajectories may start in states that are not from s_I, may end in states that do not satisfy G, and may consider a set of objects that is different from O. Safety is captured in Definition 2 by requiring that the output plan π is a **sound plan** for Π. That is, π is applicable and ends up reaching a state that satisfies the goal. The main challenge is that the problem-solver – the agent – needs to find a sound plan to Π but it is not given the set of fluents, actions, and action model of the domain ($\mathcal{F}$, $\mathcal{A}$, and M^*, respectively).

In this work, we make the following simplifying assumptions. Actions have deterministic effects, the agent has complete observability, and when the agent observes a grounded action $a = \langle A, b_a \rangle$, it is able to discern that a is the result of grounding A with b_a. Similarly, if it observes a state with a grounded fluent $f = \langle F, b_f \rangle$, it is able to discern that f is the result of grounding F with b_f. Also, we assume that actions' preconditions and effects are conjunctions of literals, as opposed to more complex logical statements, and we do not currently consider conditional effects of actions. These assumptions are reasonable when planning in digital/virtual environments, such as video games, or environments that have been instrumented with reliable sensors, such as warehouses designed to be navigated by robots (Li et al. 2020). Later, we will discuss approaches to relax these assumptions and apply our work to a broader range of environments.

3 Conservative Planning in Grounded Domains

Our approach for solving the model-free planning problem in lifted domains builds on the conservative planning approach proposed by Stern and Juba (2017) for grounded domains. Thus, we first describe their approach. This is done in a slightly different framing, which allows us to present a new theoretical property regarding the strength of the learned action model.

3.1 Inference Rules for Grounded Domains

In a grounded domain, a state is a set of literals, and so are the preconditions and effects of all actions. That is, there is no notion of lifted literals of actions.

First, we define the notion of a consistent action model following the semantics of classical planning.

Definition 3 (Consistent Action Model)**.** *An action model* M *is consistent with a set of trajectories* $\mathcal{T}$ *if for every action triplet* $\langle s, a, s' \rangle \in \mathcal{T}(a)$ *it holds that:*
1. All preconditions are satisfied: $\forall l \in pre(a) : l \in s$
2. All effects are satisfied: $\forall l \in \mathit{eff}(a) : l \in s'$
3. Frame axioms[1] *hold:* $\forall l : (l \notin \mathit{eff}(a) \wedge l \notin s) \rightarrow l \notin s'$

The contrapositives of the conditions in the above definition can be interpreted as inference rules as follows.

Observation 1 (Inference rules for grounded domains)**.** *For any action triplet* $\langle s, a, s' \rangle$ *it holds that:*
- *Rule 1 [not a precondition].* $\forall l \notin s : l \notin pre(a)$
- *Rule 2 [not an effect].* $\forall l \notin s' : l \notin \mathit{eff}(a)$
- *Rule 3 [must be an effect].* $\forall l \in s' \setminus s : l \in \mathit{eff}(a)$

So, Rule 1 states that a literal that is not in a pre-state cannot be a precondition. Rule 2 states that a literal that is not in a post-state cannot be an effect. Rule 3 states that a literal that is in the post-state but not in the pre-state, must be an effect. Since this is just a restatement of the definition of a consistent action model, these rules precisely characterize the action models that are consistent with a given set of traces.

In the fully observable deterministic world of classical planning, every action model that is not consistent with the given set of trajectories is false, and the set of consistent action models must contain the real action model. However, some of the consistent action models are different from the real action model, and plans generated with them may yield a failure, e.g., trying to apply an action in a state in which not all preconditions hold.

Definition 4 (Safe Action Model)**.** *An action model* M' *is safe with respect to an action model* M *iff for every state* s *and grounded action* a *it holds that*

$$pre_{M'}(a) \subseteq s \rightarrow \Big(pre_{M}(a) \subseteq s \wedge a_{M'}(s) = a_M(s) \Big) \qquad (2)$$

[1] This means literals only change as a result of action effects.

In words, Definition 4 says that if action model M' is safe w.r.t. M then for every state s and action a, if a is applicable in s according to M' then (1) a is also applicable in s according to M, and (2) applying a to s results in the same state according to both action models. We say that an action model is safe if it is a safe action model w.r.t. the real action model M^*.

Observe that any plan generated by a planner given a safe action model must also be a sound plan according to M^*. The *conservative planning* approach (Stern and Juba 2017) for safe model-free planning is based on this observation. In conservative planning, we first learn from the given set of trajectories, an action model M that is safe w.r.t. M^*, and then apply an off-the-shelf planner to generate plans using M. To learn such a safe action model, Stern and Juba (2017) proposed the following algorithm. First, assume every action a has all literals as its preconditions and no literals as its effects. Then, iterate over every action triplet in $\mathcal{T}(a)$ and apply the rules in Observation 1 to remove incorrect preconditions and to add effects. We refer to this algorithm hereafter as the *Safe Grounded Action-Model (SGAM) Learning* algorithm, and discuss its theoretical properties.

3.2 Theoretical Analysis

Theorem 1 (SGAM Learning is sound (Stern and Juba 2017))**.** *SGAM learning produces a safe action model.*

The main limitation of using a safe action model M_{safe} is that it may be *weaker* than the real action model (M^*), in the sense that there may be states in which an action a is applicable according to M^*, but not applicable according to M_{safe}. Consequently, there may be planning problems that are solvable with M^* but not with M_{safe}. This is stated in a more formal and general way below.

Definition 5 (Strength of Action Models)**.** *If there exists a trajectory that is consistent with M' but not with M, then we say that M is weaker than M'. If no such trajectory exists then we say that M is at least as strong as M'.*

If M is at least as strong as M' then given enough computation time, every planning problem that is solvable with M' is also solvable with M. Alternatively, if M' is weaker than M then there may be planning problems that cannot be solved using M' but can be solved using M. Next, we complement Theorem 1 by showing that the action model returned by SGAM learning is at least as strong as every safe action model that is consistent with the given trajectories.

Theorem 2 (The Strength of SGAM Learning)**.** *Let M_{SGAM} be the action model created by SGAM learning given the set of trajectories $\mathcal{T}$. M_{SGAM} is at least as strong as any action model M' that is safe and consistent with $\mathcal{T}$.*

Proof. Consider an action model M', which is safe and consistent with $\mathcal{T}$. Let a be an action and s be a state such that a is applicable in s according to M', i.e., $pre_{M'}(a) \subseteq s$. Since M' is safe w.r.t. M^*, then $pre_{M^*}(a) \subseteq s$ and $a_{M'}(s) = a_{M^*}(s)$. By construction of M_{SGAM}, if a literal l is a precondition of a according to M_{SGAM}, then it has appeared in the pre-state of all action triplets in $\mathcal{T}(a)$. Thus, there exists a consistent action model in which l is a precondition of a and this action model may be the real model. Therefore, since M' is safe it follows that $pre_{M_{SGAM}}(a) \subseteq pre_{M'}(a)$, and thus a is applicable in s according to M_{SGAM}, i.e., $pre_{M_{SGAM}}(a) \subseteq s$. Since M_{SGAM} is safe, $a_{M_{SGAM}}(s) = a_{M^*}(s) = a_{M'}(s)$. Thus, every trajectory consistent with M' will also be consistent with M_{SGAM}. □

While the action model returned by SGAM is at least as strong as any other safe action model, it may still be weaker than the real action model. Consequently, conservative planning for model-free planning is bound to be sound but incomplete—it generates plans that are sound but it may fail to generate plans for some solvable planning problems.

A statistical analysis showed that under some assumptions, the number of trajectories SGAM learning needs to learn a safe action model that can solve most problems is quasilinear in the number of actions in the domain (Stern and Juba 2017). However, the number of grounded actions in a *lifted domain* can be quite large: the number of grounded actions that are groundings of a single lifted action grows polynomially with the number of objects in the domain (exponentially in the number of parameters). On the other hand, in a lifted domain, the real action model is assumed to be defined by lifted actions. This enables us to generalize SGAM learning across multiple groundings of the same lifted action, eliminating the dependence on the number of objects in the number of trajectories needed to learn a useful safe action model. We describe this in the next section.

4 Conservative Planning for Lifted Domains

In this section, we describe a conservative planning approach for safe model-free planning in lifted domains, which is based on a novel generalization of SGAM learning to lifted domains. We refer this algorithm as simply SAM learning. To describe SAM learning, we denote by $bindings(b_A, b_L)$ the set of all parameter bindings $b_{L,A}$ that satisfy the following

$$b_A \circ b_{L,A} = b_L. \tag{3}$$

4.1 Inference Rules for Lifted Domains

The core of our algorithm is the following generalization of Observation 1, defining what observing an action triplet with a grounded action $\langle A, b_A \rangle$ entails for the lifted action A.

Observation 2. *For any action triplet $\langle s, \langle A, b_A \rangle, s' \rangle$*

- *Rule 1 [not a precondition].* $\forall \langle L, b_L \rangle \notin s$:

$$\forall b \in bindings(b_A, b_L) : \langle L, b \rangle \notin pre(A) \tag{4}$$

- *Rule 2 [not an effect].* $\forall \langle L, b_L \rangle \notin s'$*:*

$$\forall b \in bindings(b_A, b_L) : \langle L, b \rangle \notin eff(A) \tag{5}$$

- *Rule 3 [an effect].* $\forall \langle L, b_L \rangle \in s' \setminus s$:

$$\exists b \in bindings(b_A, b_L) : \langle L, b \rangle \in eff(A) \tag{6}$$

I.e., in ILP terminology, the grounded literal $\langle L, b_L \rangle$ is subsumed *by some $\langle L, b \rangle \in eff(A)$.*

For much of this paper, we make the following assumption:

Algorithm 1: Safe Action-Model (SAM) Learning

Input : $\Pi_{\mathcal{T}} = \langle T, O, s_I, G, \mathcal{T} \rangle$
Output: An action model that is safe w.r.t. the action model that generated $\mathcal{T}$

1 $\mathcal{A}' \leftarrow$ all lifted actions observed in $\mathcal{T}$
2 **foreach** *lifted action* $A \in \mathcal{A}'$ **do**
3 $\quad$ *eff*$(A) \leftarrow \emptyset$
4 $\quad$ *pre*$(A) \leftarrow$ all parameter-bound literals
5 $\quad$ **foreach** $(s, \langle A, b_A \rangle, s') \in \mathcal{T}(A)$ **do**
6 $\quad\quad$ **foreach** $\langle L, b_{L,A} \rangle \in$ *pre*(A) **do**
7 $\quad\quad\quad$ **if** $\langle L, b_A \circ b_{L,A} \rangle \notin s$ **then**
8 $\quad\quad\quad\quad$ Remove $\langle L, b_{L,A} \rangle$ from *pre*(A)
9 $\quad\quad$ **foreach** $\langle L, b_L \rangle \in s' \setminus s$ **do**
10 $\quad\quad\quad$ $b_{L,A} \leftarrow \left\langle L, (b_A)^{-1} \circ b_L \right\rangle)$
11 $\quad\quad\quad$ Add $\langle L, b_{L,A} \rangle$ to *eff*(A)
12 **return** (*pre*, *eff*)

Definition 6 (Injective Action Binding)**.** *In every grounded action $\langle A, b_A \rangle$, the binding b_A is an injective function, i.e., every parameter of A is mapped to a different object.*

Under this assumption, for every pair of bindings b_L and b_A there exists a unique $b_{L,A}$ that satisfies Eq. 3. This binding is obtained by inverting b_A, i.e.,

$$\mathit{bindings}(b_A, b_L) = \{(b_A)^{-1} \circ b_L\}. \tag{7}$$

where $(b_A)^{-1}$ maps an object o to the parameter of A that b_A maps to o. That is, each grounded literal and action that appears in a trajectory is essentially a renaming of the corresponding parameters in the lifted literals and actions by objects; the grounded literals and actions are *OI-subsumed* by the lifted literals and actions (De Raedt 2008, Section 5.5.1). Equation 7 simplifies the inference rules given in Observation 2. In particular, the "an effect" rule (Rule 3) becomes

$$\forall \langle L, b_L \rangle \in s' \setminus s : \left\langle L, (b_A)^{-1} \circ b_L \right\rangle \in \mathit{eff}(A). \tag{8}$$

4.2 SAM Learning for Lifted Domains

We now present our SAM Learning algorithm for lifted domains in Algorithm 1. For every lifted action A observed in some trajectory, we initially assume that A has no effects and all possible parameter-bound literals are its preconditions (line 4 in Algorithm 1).[2] Then, for every action triplet $(s, \langle A, b_A \rangle, s')$ with this lifted action, we remove from the preconditions of A every parameter-bound literal $\langle L, b_{L,A} \rangle$ that is not satisfied in the current pre-state (Rule 1 in Observation 2). Then, for every grounded literal $\langle L, b_L \rangle$ that holds in the post-state s' and not in s, we add a corresponding effect to A (Rule 3 in Observation 2). Note that Rule 2 in Observation 2 is not needed since we initialize the set of effects of every action to be an empty set.

Theorem 3. *Given a set of trajectories $\mathcal{T}$, SAM learning (Algorithm 1) runs in time*

$$\mathcal{O}\Big(\sum_{A \in \mathcal{A}} |\mathcal{T}(A)| \sum_{F \in \mathcal{F}} \prod_{t \in T} \mathit{arity}(A,t)^{\mathit{arity}(F,t)} \Big)$$

Proof. For every action $A \in \mathcal{A}$, SAM learning iterates over all action triplets in $\mathcal{T}(A)$ and, in the worst case, checks every possible parameter-bound literal $\langle L, b_{L,A} \rangle$ if it is not a precondition and if it is an effect. There are $\mathit{arity}(A,t)^{\mathit{arity}(L,t)}$ ways to bind the parameters of L of type t to the parameters of A, and hence $\prod_{t \in T} \mathit{arity}(A,t)^{\mathit{arity}(L,t)}$ parameter-bound literals with A and L. □

[2] It is possible to initialize the preconditions of every lifted action to the pre-state of one of the action triplets in which it is used.

Action	Params	Precond.	Effects
Move	?tr - truck ?from - location ?to - location	at(tr, from)	at(tr, to), not(at(tr, from))
Load	?pkg - package ?tr - truck ?loc - location	at(tr, loc) at(pkg, loc)	on(pkg, tr), not(at(pkg, loc))
Unload	?pkg - package ?tr - truck ?loc - location	at(tr, loc), on(pkg, tr)	not(on(pkg,tr), at(pkg, loc)

Table 1: The parameters, preconditions, and effects of the actions according to the real action model of our simple logistics example.

4.3 Safety Property

We extend the notion of a *safe action model* to lifted domains as follows. An action model M in a lifted domain is safe iff every grounded action defined by M satisfies Eq. 2. This definition preserves the property that a safe action model is an action model that enables generating plans that are guaranteed to be sound w.r.t. M^*. We show next that *SAM* Learning for lifted domains indeed returns a safe action model.

Theorem 4. *Given the injective action binding assumption, SAM Learning (Algorithm 1) creates a safe action model.*

Sketch of Proof. The main loop (lines 6–12) maintains the following invariant: $\mathit{pre}_{M^*}(A) \subseteq \mathit{pre}(A)$ and $\mathit{eff}(A) \subseteq \mathit{eff}_{M^*}(A)$. Thus, if an action is applicable according to M it is also applicable in M^*. Also, if we are missing an effect for a lifted action A, it means it always appeared in the pre-state of all action triplets for that action, and thus it must be in its preconditions. Thus, whenever A is applicable according to M it will yield the same post-state as it would according to M^*. The full proof appears in the technical report (Juba, Le, and Stern 2021).

4.4 An Example of SAM Learning

Consider the following simple logistics problem. There are five objects: one *truck* object (tr), one *package* object (pkg), and three *locations* objects (A, B, and C). *at(?truck, ?location)* and *on(?truck, ?package)* are lifted fluents representing that the truck is at the location and the package is on the truck, respectively. There are three possible actions: *Move*, *Load*, and *Unload*. Table 1 lists the parameters, preconditions, and effects of these actions in M^*. Now, assume we are given three trajectories T_1, T_2, and T_3. T_1 starts with the

truck and the package at location A, and performs two move actions: Move(tr, A, B) and Move(tr, B, C). T_2 starts in the same state, but performs Load(pkg, tr, A) and Move(tr, A, B). T_3 starts with the truck at location A and the package at location B, and performs Move(tr, A, B), Load(pkg, tr, B), Move(tr, B, C), and Unload(pkg, tr, C). Given only the first trajectory T_1, the action model returned by SAM Learning already contains the real action model for the lifted Move action, since the only grounded fluents that can be bound to the parameters of the grounded action Move(tr, A, B) are at(tr, A) and not(at(tr, B)) in the pre-state, and at(tr, B) and not(at(tr, A)) in the post-state. In contrast, SAM Learning for grounded domains will not know anything about the preconditions and effects of the grounded action Move(tr, B, C) unless it is also given the trajectory T_3. Similarly, given the second trajectory T_2, the action model returned by SAM Learning contains the real action model for the lifted Load action, since the only grounded fluents that can be bound to the parameters of the grounded action Load(pkg, tr, A) are at(tr, A), at(pkg, A), and not(on(pkg, tr)) in the pre-state and at(tr, A), not(at(pkg, A)), and on(pkg, tr)) in the post-state. In fact, given T_1, T_2, and T_3, SAM Learning is able to learn the real action model for this domain. Note that since there are 10 grounded actions in this domain (four Move actions and three Load and Unload actions), SGAM Learning will require at least 10 action triplets to learn an action model with all of the actions.

5 Sample Complexity Analysis

Planning with a safe action model is a sound approach for safe model-free planning, since every plan it outputs is a sound plan according to the real action model. However, it is not complete: a planning problem may be solvable with the real action model, but not the learned one. As in prior work on safe model-free planning (Stern and Juba 2017), we can bound the likelihood of facing such a problem as follows.

Let $\mathcal{P}_D$ be a probability distribution over solvable planning problems in a domain D. Let $\mathcal{T}_D$ be a probability distribution over pairs $\langle P, T\rangle$ given by drawing a problem P from $\mathcal{P}(D)$, using a sound and complete planner to generate a plan for P, and setting T to be the trajectory from following this plan.[3]

Theorem 5. *Under the injective action binding assumption, given* $m \geq \frac{1}{\epsilon}(2\ln 3 \sum_{\substack{F\in\mathcal{F}\\A\in\mathcal{A}}} \prod_{t\in T} arity(A,t)^{arity(F,t)} + \ln\frac{1}{\delta})$ *trajectories sampled from* $\mathcal{T}_D$*, with probability at least* $1-\delta$ *SAM learning for lifted domains (Algorithm 1) returns a safe action model* M_{SAM} *such that a problem drawn from* $\mathcal{P}_D$ *is not solvable with* M_{SAM} *with probability at most* ϵ*.*

Theorem 5 guarantees that with high probability ($\geq 1-\delta$) SAM Learning returns an action model that will only fail to solve a given problem with low probability ($\leq \epsilon$), given a number of example trajectories linear in the size of the models. For example, in the real action model of our simple logistics example with two binary fluents and three ternary actions, the load and unload actions have a single argument of each type; only the move action has two arguments of the same type (location). The only fluents that have location arguments are the at fluents, which have arity one with respect to locations. Thus, guaranteeing $\epsilon = \delta = 5\%$ requires only 324 trajectories. The rest of this section is devoted to establishing Theorem 5.

Definition 7 (Adequate)**.** *An action model M is* ϵ-adequate *if, with probability at most ϵ, a trajectory T sampled from $\mathcal{T}_D$ contains an action triplet $\langle s, a, s'\rangle$ where s does not satisfy $pre_M(a)$.*[4]

Lemma 1. *The action model returned by SAM Learning (Algorithm 1) given m trajectories (as specified in Theorem 5) is ϵ-adequate with probability at least $1-\delta$.*

Sketch of proof. Since the trajectories are are drawn independently, the probability that an action model that is *not* ϵ-adequate is consistent with the m trajectories we draw is bounded by $(1-\epsilon)^m \leq \frac{\delta}{\#\text{action models}}$. A union bound over all of the action models that are not ϵ-adequate completes the proof. A complete proof appears in the technical report (Juba, Le, and Stern 2021).

Proof of Theorem 5. Let M be an action model returned by SAM Learning given m samples. Thus, M is a safe action model (Theorem 4) and it is ϵ adequate (Lemma 1). Consider a problem P drawn from $\mathcal{P}(D)$, and its corresponding pair $\langle P, T\rangle$ from $\mathcal{T}(D)$. Since M is ϵ-adequate, with probability at least $1-\epsilon$, for every action triplet $\langle s, a, s'\rangle \in T$ a is applicable in s, that is, $pre_M(a) \subseteq s$. Since M is a safe action model, we have that $a_M(s) = a_{M^*}(s) = s'$. Thus, with probability at least $1-\epsilon$ the trajectory T is consistent with the learned action model M, and therefore P can be solved with M □

6 Multiple Action Bindings

When the injective action-binding assumption does not hold, multiple action parameters are bound to the same object and thus $(b_A)^{-1}$ is not defined. As a result, when SAM Learning infers an effect (Rule 3 in Observation 2) it cannot generalize it to be a unique effect of the corresponding lifted action, as done in line 10 in Algorithm 1. This poses a challenge to learning a safe action model, as the information that can be inferred from observing action triplets can be complex.

For example, consider a lifted action $A(x, y)$. Suppose x and y are associated with the same type and o is an object of that type. Given the action triplet $\langle\{\,\}, A(o, o), \{L(o)\}\rangle$, the agent can infer that $L(o)$ is an effect of the grounded action $A(o, o)$. However, the agent cannot accurately infer the effect of the lifted action $A(x, y)$: it can be either $\{L(x)\}$, $\{L(y)\}$, or both. Concretely, if o_1 and o_2 are two different objects from the same type as o, the agent cannot determine if applying $A(o_1, o_2)$ will result in a state with $\{L(o_1)\}$, $\{L(o_2)\}$, or $\{L(o_1), L(o_2)\}$. Consequently, any safe action model must not enable groundings of A that bind x and y to different objects, unless $L(x)$ and $L(y)$ both already hold.

Now, assume the agent is also given the action triplet $\langle\{L(o_1)\}, A(o_1, o_2), \{L(o_1)\}\rangle$. The pre- and post-state are

[3] The planner need not be deterministic.

[4] An action model may not contain any information about some action a. For the purpose of safe planning this is equivalent to an action model in which the precondition to a can never be satisfied.

the same, so in Algorithm 1 we cannot learn any new effects of A from this triplet. However, we can infer that $L(o_2)$ is not an effect of the grounded action in this triplet. Consequently, the parameter-bound literal $L(y)$ cannot be an effect of the lifted action A. Thus, this second action triplet does provide useful information: it allow us to infer that the lifted action $A(x, y)$ has a parameter-bound effect $L(x)$.

In a planning task, we might avoid the above by reformulating the domain to satisfy the injective action binding assumption. However, in a learning setup, we do not have control over how the domain is formulated and so the domain we are learning may indeed violate the injective action binding assumption, preventing the application of SAM learning. Next, we describe Extended SAM Learning, which addresses such cases by capturing the form of inference described above.

6.1 Extended SAM Learning

Extended SAM (E-SAM) learning works in two stages. First, it creates for every lifted action A a conjunction and a Conjunctive Normal Form (CNF) formula, denoted $\mathit{Conj}_{\mathit{pre}}(A)$ and $\mathit{CNF}_{\mathit{eff}}(A)$, that describe a set of constraints for a safe action model. Then E-SAM learning generates a safe action model based on these formulas.

Safe Action Model Constraints $\mathit{Conj}_{\mathit{pre}}(A)$ uses atoms of the form IsPre($\langle L, b_{L,A}\rangle$), which specify that $\langle L, b_{L,A}\rangle$ is a precondition L in a safe action model. Similarly, $\mathit{CNF}_{\mathit{eff}}(A)$ uses atoms of the form IsEff($\langle L, b_{L,A}\rangle$), which specify that $\langle L, b_{L,A}\rangle$ is an effect of L in a safe action model.

Initially, $\mathit{Conj}_{\mathit{pre}}(A)$ and $\mathit{CNF}_{\mathit{eff}}(A)$ represent that all possible parameter-bound literals are preconditions and there are no effects. Then, E-SAM learning iterates over every action triplet (s, a, s') in the given set of trajectories in which a is a grounding of A. For every such triplet, it applies the inference rules in Observation 2 as follows.

Every parameter-bound literal $\langle L, b_{L,A}\rangle$ such that $\langle L, b_A \circ b_{L,A}\rangle$ is not in the pre-state cannot be a precondition (Rule 1). So, we remove IsPre($\langle L, b_{L,A}\rangle$) from $\mathit{Conj}_{\mathit{pre}}$ for such parameter-bound literals. Similarly, every parameter-bound literal $\langle L, b_{L,A}\rangle$ such that $\langle L, b_A \circ b_{L,A}\rangle$ is not in the post-state cannot be an effect (Rule 2). So, we add $\neg$IsEff($\langle L, b_{L,A}\rangle$) to $\mathit{CNF}_{\mathit{eff}}$ for such parameter-bound literals. Finally, every grounded literal $\langle L, b_L\rangle$ in $s' \setminus s$ must be an effect. So, we add to $\mathit{CNF}_{\mathit{eff}}$ the disjunction over all parameter-bound literals $\langle L, b_A \circ b_{L,A}\rangle$ that satisfy $\langle L, b_A \circ b_{L,A}\rangle = \langle L, b_L\rangle$ (Rule 3). Once the given trajectories have been processed by the algorithm, we simplify the CNF by applying unit propagation and removing subsumed clauses.

Proxy Actions The main challenge in creating a safe action model from the generated formulas is the disjunction in $\mathit{CNF}_{\mathit{eff}}$, which represents uncertainty w.r.t to the effects of action. To address this, we create a safe action model with a set of proxy actions that ensure every action is only applicable when we know its effects. We achieve this by computing, for each possible subset of the parameter-bound literals in the formulas for a given action, most general unifiers for the literals; in our setting, such unifiers simply identify subsets of the action parameters. Alternatively, if the parameter-bound literal appears in the precondition, then the literal does not need to be included in the unifier, and we know that the corresponding effect will always hold. Hence, since at least one of the unified parameters occurs in the parameter binding of the effect in the true action model, so when the parameters in the set are all bound to the same object (or the literal appears in the precondition), we can guarantee that the corresponding effect literal holds in the post-state. In more detail, this is done as follows.

If an action has only unit clauses, we have a single action with the effects indicated by the positive literals. Otherwise, we create a proxy action for all subsets of the parameter-bound literals in non-subsumed non-unit clauses. (The number of proxy actions is thus exponential in the size of the formula of non-unit clauses.) In this proxy action, we identify all of the parameters that appear in the same position of the literals in the subset with the same fluents. Each proxy action has the following set of preconditions and effects: every unit clause in the CNF and every clause in the corresponding subset specifies an effect of the proxy action. For the subset of literals not chosen for this proxy action, the proxy action has the corresponding literals as additional preconditions, in addition to the preconditions of the original SAM Learning action model. Every plan generated by the action model created by the resulting action model is translated to a plan without proxy actions by replacing them with the actions for which they were created. Algorithms 2 and 3 list the complete pseudocode of E-SAM learning.

Theoretical Properties E-SAM Learning creates an action model that satisfies the same properties as the action model created by SAM learning under the injective action binding assumption, as captured in Theorems 4 and 2.

Theorem 6. *The E-SAM Learning action model is safe.*

Algorithm 2: Extended SAM Learning

Input : $\Pi_{\mathcal{T}} = \langle T, O, s_I, G, \mathcal{T}\rangle$
Output: (*pre*, *eff*) for a safe action model

$\mathcal{A}' \leftarrow$ all lifted actions observed in $\mathcal{T}$
foreach *lifted action* $A \in \mathcal{A}'$ **do**
  $(\mathit{Conj}_{\mathit{pre}}, \mathit{CNF}_{\mathit{eff}}) \leftarrow$ ExtractClauses(A, $\mathcal{T}(A)$)
  $\mathit{CNF}^1_{\mathit{eff}} \leftarrow$ all unit clauses in $\mathit{CNF}_{\mathit{eff}}$
  SurelyEff$\leftarrow \{l \mid \text{IsEff}(l) \in \mathit{CNF}^1_{\mathit{eff}}\}$
  SurelyPre$\leftarrow \{l \mid \text{IsPre}(l) \in \mathit{Conj}_{\mathit{pre}}\}$
  `/* Create proxy actions for non-unit effects clauses */`
  $\mathit{CNF}_{\mathit{eff}} \leftarrow \mathit{CNF}_{\mathit{eff}} \setminus \mathit{CNF}^1_{\mathit{eff}}$
  foreach $S \in$ *Powerset*($\mathit{CNF}_{\mathit{eff}}$) **do**
    $\mathit{pre}(A_S) \leftarrow$ SurelyPre; $\mathit{eff}(A_S) \leftarrow$ SurelyEff
    foreach $C_{\mathit{eff}} \in \mathit{CNF}_{\mathit{eff}} \setminus S$ **do**
      foreach *IsEff*$(l) \in C_{\mathit{eff}}$ **do**
        Add l to $\mathit{pre}(A_S)$
    MergeObjects$\big(S, \mathit{pre}(A_S), \mathit{eff}(A_S)\big)$
return (*pre*, *eff*)

Algorithm 3: ExtractClauses

Input : A, a lifted action
Input : $\mathcal{T}(A)$, action triplets that contain A
Output: ($Conj_{pre}$, CNF_{eff}), representing the constraints over $pre(A)$ and $eff(A)$

$CNF_{eff} \leftarrow \emptyset$; $Conj_{pre} \leftarrow \emptyset$
foreach *parameter-bound literal* $\langle L, b_{L,A} \rangle$ **do**
  Add IsPre($\langle L, b_{L,A} \rangle$) to $Conj_{pre}$
foreach $(s, \langle A, b_A \rangle, s') \in \mathcal{T}(A)$ **do**
  foreach *IsPre(*$\langle L, b_{L,A} \rangle \in Conj_{pre}$*)* **do**
    if $\langle L, b_A \circ b_{L,A} \rangle \notin s$ **then**
      Remove IsPre($\langle L, b_{L,A} \rangle$) from $Conj_{pre}$
  foreach $\langle L, b_L \rangle \in s' \setminus s$ **do**
    $C_{eff} \leftarrow \bot$
    foreach $b_{L,A} \in bindings(b_A, b_L)$ **do**
      $C_{eff} \leftarrow C_{eff} \vee$ IsEff($\langle L, b_{L,A} \rangle$)
    Add EffectsClause to CNF_{eff}
  foreach *parameter bound literal* $\langle L, b_{L,A} \rangle$ **do**
    $b_L \leftarrow \langle L, b_{L,A} \circ b_A \rangle$
    if $\langle L, b_L \rangle \notin s'$ **then**
      Add $\neg$IsEff($\langle L, b_{L,A} \rangle$) to CNF_{eff}
Minimize(CNF_{eff})
return ($Conj_{pre}$, CNF_{eff})

Proof. For each of the proxy actions, for every effect, at least one of the parameter-bound literals for the identified parameters is an effect of the true action. Furthermore, the preconditions ensure that the rest of the uncertain effects are already present in the pre-state. The post-state of the proxy action is thus identical to that of the true action when its precondition is satisfied. Likewise, the proxy actions have preconditions that are only stronger than the actual precondition. Eq. 2 therefore holds. The rest of the claim now follows from the argument in Theorem 4. □

Recall that a *prime implicate* is a clause that is entailed by a formula for which no subclause is also entailed. CNF_{eff} consists of precisely these prime implicates.

Lemma 2. *All prime implicates of CNF_{eff} are derived by unit propagation.*

Proof. Note that the clauses created by Rule 3 contain only positive literals, and negative literals are only created by Rule 1 and 2, which create unit clauses. Hence, unit propagation is sufficient to capture all possible resolution inferences from these clauses. By the completeness of resolution for prime implicates (e.g., (Brachman and Levesque 2004, Ch. 13, Exercise 1)), all of the prime implicates of CNF_{eff} can be derived by resolution. In turn, therefore, unit propagation can also derive all of the prime implicates of CNF_{eff}. □

Theorem 7. *Every action model M' that is consistent with $\mathcal{T}$ and safe w.r.t. the real action model M^* is also safe with respect to the extended SAM Learning action model.*

	# lifted fluents	# lifted actions	max arity fluents	max arity actions	max ground fluents	max ground actions
Blocks	5	4	2	2	182	182
Depot	6	5	4	2	75	450
Ferry	5	3	2	2	75	75
Floortile	10	7	2	4	40	64
Gripper	4	3	2	3	42	84
Hanoi	3	1	2	3	33	166
Npuzzle	3	1	2	3	80	80
Parking	5	4	2	3	182	2,184
Satellite	8	5	2	4	75	1875
Sokoban	4	2	3	5	288	564
Spanner	6	3	2	4	12	12
Transport	5	3	2	5	870	3600

Table 2: Statistics on the domains in our experiments.

Sketch of Proof. Observation 2 characterizes the set of action models consistent with $\mathcal{T}$. Since CNF_{eff} consists of the prime implicates of this set, M^* could be an action model that uniquely satisfies any one of the literals of any clause of CNF_{eff}; therefore, for M' to be safe w.r.t. this unknown M^*, it must bind the literals in accordance with one of our proxy actions. The argument is now similar to the proof of Theorem 2; a full proof appears in the technical report (Juba, Le, and Stern 2021).

Time complexity E-SAM learning can be split into two parts: a **learning** part, which extracts clauses about the preconditions and effects of the actions (line 3 in Alg. 2), and a **compilation** part that generates a PDDL encoding that can be used by off-the-shelf PDDL planners. The latter involves the creation of the proxy actions. The learning part of E-SAM learning can be implemented to run in polynomial time in the number of parameter-bound literals and total number of action triplets, similar to SAM learning. The compilation part, however, may run in exponential time due to the inability of PDDL to capture the uncertainty over actions' effects that has been learned (captured by CNF_{eff}). Future work may investigate avoiding this exponential step by instead compiling the learned knowledge to a domain encoding for a conformant planner (Bonet 2010).

7 Experiments

Next, we perform an experimental evaluation of SAM Learning over planning problems from twelve domains from the IPC (McDermott 2000). Table 2 lists the names of these domains, the number of lifted fluents and actions, the largest arity of these lifted fluents and actions, and the largest number of grounded fluents and actions in our dataset. We have chosen only domains in the IPC benchmarks in which the injective action binding assumption holds. For such domains, E-SAM Learning and SAM Learning behave the same.

For each domain, we generated problems using the problem generator provided in the IPC learning tracks and solved them using their true action models with the MADAGASCAR planner (Rintanen 2014) to obtain example trajectories. These trajectories were broken to action triplets and

Domain	# Objects	Trajectories		Triplets	
		SAM	FAMA	SAM	FAMA
Blocks	7 blocks	1	2	13	22
	8 blocks	1	2	16	29
	9 blocks	1	2	18	35
	10 blocks	1	2	22	40
	11 blocks	1	2	25	46
	12 blocks	1	2	28	53
	13 blocks	1	2	35	60
	14 blocks	1	2	42	72
Depot	1 truck, 2 places, 4 hoists, 10 crates	1	1	18	24
	1 truck, 2 places, 4 hoists, 15 crates	1	1	26	32
	2 trucks, 3 places, 5 hoists, 10 crates	1	1	22	28
	2 trucks, 3 places, 5 hoists, 15 crates	1	1	28	36
Ferry	2 locations, 8 cars	1	1	4	7
	3 locations, 10 cars	1	1	9	12
	4 locations, 12 cars	1	1	12	15
	5 locations, 15 cars	1	1	14	17
Floortile	3x3, 2 robots	1	2	13	22
	4x3, 2 robots	1	2	13	32
	4x4, 2 robots	1	2	16	40
	5x4, 2 robots	1	2	16	52
Gripper	2 rooms, 6 balls	1	1	4	8
	2 rooms, 10 balls	1	1	5	8
	3 rooms, 8 balls	1	1	5	8
	3 rooms, 14 balls	1	1	5	9
Hanoi	3 disks	1	1	3	3
	4 disks	1	1	3	3
	5 disks	1	1	3	3
	6 disks	1	1	3	3
Npuzzle	8 tiles	1	1	1	1
	15 tiles	1	1	1	1
	24 tiles	1	1	1	1
Parking	3 curbs, 4 cars	2	3	13	20
	5 curbs, 8 cars	2	4	32	52
	7 curbs, 12 cars	2	4	53	87
	8 curbs, 14 cars	2	3	72	98
Satellite	2 sats., 4 instrs., 4 modes, 8 dirs.	1	1	20	28
	4 sats., 4 instrs., 4 modes, 8 dirs.	1	1	20	28
	5 sats., 5 instrs., 5 modes, 10 dirs.	1	1	24	32
	5 sats., 5 instrs., 5 modes, 15 dirs.	1	1	26	34
Sokoban	5x5, 2 boxes	1	1	4	6
	7x7, 2 boxes	1	1	6	8
	8x8, 3 boxes	1	1	5	7
	9x9, 3 boxes	1	1	8	10
Spanner	10 spanners, 10 nuts, 2 locations	1	1	14	16
	10 spanners, 10 nuts, 4 locations	1	1	16	18
	10 spanners, 10 nuts, 6 locations	1	1	18	20
	11 spanners, 11 nuts, 2 locations	1	1	15	17
	11 spanners, 11 nuts, 4 locations	1	1	17	19
	11 spanners, 11 nuts, 6 locations	1	1	19	21
	12 spanners, 12 nuts, 2 locations	1	1	16	18
	12 spanners, 12 nuts, 4 locations	1	1	18	20
	12 spanners, 12 nuts, 6 locations	1	1	20	22
Transport	2 trucks, 5 packages, 10 locations	1	1	16	20
	2 trucks, 10 packages, 20 locations	1	1	18	22
	4 trucks, 10 packages, 20 locations	1	1	22	26
	4 trucks, 15 packages, 30 locations	1	1	24	30

Table 3: Number of trajectories and action triplets needed to learn the real action model in each domain.

given to the SAM Learning algorithm one at a time to obtain a safe action model. We halted this process when the learned action model was equivalent to the real model, and report the number of triplets and trajectories given to the algorithm. As a baseline, we performed this experiment also with FAMA (Aineto, Celorrio, and Onaindia 2019), which is a modern algorithm for learning action models from trajectories. Note that unlike SAM Learning, FAMA has no safety guarantee. In addition, SAM learning runs in time linear in the number of lifted actions, lifted literals, and trajectories, while FAMA runs an automated planner which has an exponential worst-case running time (as planning is PSPACE-complete). SAM learning is only exponential in the maximal number of parameters of each action and literal. Thus, SAM learning can easily scale to very large domains.

Table 3 lists the results of our experiments. The "# Objects" column lists the objects in the problem, and the values under "Trajectories" and "Triplets" are the number of trajectories and action triplets, respectively, required to learn the correct model. In all cases, both methods were able to recover the real action model. However, SAM Learning was able to find such a model using at most as many, and often significantly fewer triplets and trajectories. For example, for the Floortile problem with 2 robots and a 5×4 floor, SAM learned the correct model with only 16 action triplets while FAMA required 52 action triplets. In fact, in all domains except Parking, SAM Learning learned the correct model with a single trajectory. Note that once SAM Learning finds a correct model it will never change it, since SAM only removes literals that are not satisfied in the pre-state from the preconditions and adds literals that switch values between pre and post-states to the effects. Meanwhile, FAMA might add irrelevant literals or remove correct literals from the preconditons or effects as it processes more action triplets.

The code for SAM learning and our experiments is available at https://github.com/hsle/sam-learning.

8 Related Work

A variety of notions of *safety* have been considered in RL, for example capturing the ability to reliably return to a home state (Moldovan and Abbeel 2012) or avoiding undesirable states (Turchetta, Berkenkamp, and Krause 2016; Wachi et al. 2018) while learning about the environment. But, these approaches to safe exploration require some kind of strong prior knowledge, either in the form of beliefs about the transition model or knowledge that the safety levels follow a Gaussian process model. Such assumptions are reasonable in the low-level motion planning tasks where RL excels, but they do not suit the kind of discrete, high-level problems typically considered in domain-independent planning. In addition, in these works safety is soft constraint that an algorithm aims to maximize, while in our case safety is a hard constraint.

Our work is part of the growing literature on learning action models for domain-independent planning (Arora et al. 2018), which includes algorithms such as ARMS (Yang, Wu, and Jiang 2007), LOCM (Cresswell, McCluskey, and West 2013), LOCM2 (Cresswell and Gregory 2011), AMAN (Zhuo and Kambhampati 2013), and FAMA (Aineto, Celorrio, and Onaindia 2019). Similar to SAM learning, ARMS (Yang, Wu, and Jiang 2007) also defines rules to infer an action model from a given set of trajectories. Our third rule ("must be an effect") is somewhat

similar to their (I.1) rule. The other ARMS rules are different, and are designed to **explain** the observed trajectories in a succinct manner. Thus, the action model created by ARMS may be either under- or over-constrained for this purpose. LOCM (Cresswell, McCluskey, and West 2013) and LOCM2 (Cresswell and Gregory 2011) learn action models by identifying state transitions that are consistent with the observed trajectories. They do not require as input a set of possible lifted fluents or types, and learn from data the relation between objects and types, as well as the relation between actions and types. LOCM2 is a heuristic version of LOCM algorithm for learning action models. It does not support domains in which the injective action binding assumption does not hold, or domains where a deleted literal does not appear as a precondition. AMAN (Zhuo and Kambhampati 2013) is an action-model learning algorithm that is specifically designed to handle noisy observations. It constructs a graphical model and learns the statistical relationship between actions and possible state transitions. FAMA (Aineto, Celorrio, and Onaindia 2019) compiles the problem of finding an action model that is consistent with a set of trajectories to a planning problem. The solution to this planning problem is a sequence of "actions" that construct an action model. FAMA is more general than SAM or ESAM in the sense that it supports partial observability.

FAMA, as well as LOCM, LOCM2, and AMAN, aim to create an action model that **explains** the given trajectories. This can be viewed as a solving an *inductive logic programming* (Muggleton and De Raedt 1994) task. The action model they generated is only guaranteed to be **consistent** with the given set of observations. Our algorithms (SAM and ESAM) provide a stronger guarantee: the action model they create is **safe** with respect to the real action model (M^*). A safe action model (Definition 4) is, by definition, consistent with the given trajectories, but a consistent action model (Definition 3) may very well be unsafe. For example, consider an action model M in which the effects of all actions are correct (i.e., match the effects in M^*) and all actions have no preconditions. This action model is clearly unsafe, and plans generated with it may be unsound. Yet, such an action model is consistent with any trajectory generated according to the real action model (M^*). None of the above works provide a safety guarantee, and plans generated with the action models they generate may be unsound.[5]

9 Relaxing the Assumptions

Our learning of preconditions is very similar to Valiant's elimination algorithm (Valiant 1984) for learning conjunctions in supervised learning. Following his work, we can easily support preconditions that are k-CNFs (and not just a simple conjunction) by considering that all sets of possible clauses of size k as preconditions instead of a simple conjunction. This will increase the sample complexity bound in Theorem 5 by raising the first term to the k^{th} power and similarly increase the running time. Conditional effects can be similarly supported if we can bound the number of literals in their firing condition by some value k. In this case, the extension to SAM learning keeps track of all possible conditions with at most k literals that hold when an action is applied.

We believe that the algorithm can similarly be extended to handle independent, random noise, provided that either (a) all fluents are corrupted with the same probability or (b) the rate of corruption of each fluent is known. Indeed, this is an example of independent attribute noise, and extensions of Valiant's elimination algorithm to these settings were proposed by Goldman and Sloan (1995) (extending Shackelford and Volper 1988) and Decatur and Gennaro (1995), respectively. In the presence of such noise, however, the safety property must be weakened: indeed, since any combination of fluent settings may be observed, albeit with exponentially small probability, we can only expect to guarantee that the action model will be safe with high probability w.r.t. the noise. Likewise, we believe similar guarantees are possible in sufficiently benign partial information settings, following Michael (2010).

If the environment itself is far from deterministic, then clearly the STRIPS rules we learn would be inappropriate, and a different representation would be necessary. We note that if the environment is stochastic and there is noise of unknown rates that differ across fluents, then it seems to be information-theoretically impossible to learn a safe model (in our sense) even when an adequate set of deterministic rules exists, cf. the counterexample of Goldman and Sloan (1995): we cannot distinguish between a fluent that is just corrupted by observation noise from a fluent that is merely correlated with it.

10 Conclusion and Future Work

In this work, we presented the Safe Action Model Learning algorithm for lifted domains. SAM Learning for lifted domains is guaranteed to return an action model that produces sound plans, even without knowing the preconditions and effect of the actions in the domain. A theoretical analysis shows that the number of trajectories needed to learn an action model that will solve a given problem with high probability is linear in the potential size of the action model. This approach is suitable for most domains in current planning benchmarks, where the effects of actions are trivial unless the action parameters are bound to different objects. We also discussed how to adapt our algorithm to the case where this assumption does not hold. In the future, we aim to extend safe action-model learning to domains with partial observability and stochasticity.

Acknowledgements

This research is partially funded by NSF awards IIS-1908287, IIS-1939677, and CCF-1718380, and BSF grant #2018684 to Roni Stern.

[5]Note that the soundness and completeness of FAMA (Lemmas 1 and 2 there) do not refer to plans generated by the action model FAMA learns, but to the learning algorithm it self. That is, FAMA is sound in the sense that the action model it returns is consistent with the given trajectories, and it is complete in the sense that if a consistent action model exists then FAMA will find it. Indeed, FAMA may return an unsafe action model.

References

Aineto, D.; Celorrio, S.; 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. *J. Artif. Intell. Res. (JAIR)* 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.

Asai, M., and Muise, C. 2020. Learning neural-symbolic descriptive planning models via cube-space priors: The voyage home (to STRIPS). In *the International Joint Conference on Artificial Intelligence (IJCAI)*, 2676–2682.

Bonet, B. 2010. Conformant plans and beyond: Principles and complexity. *Artificial Intelligence* 174(3):245–269.

Brachman, R. J., and Levesque, H. J. 2004. *Knowledge Representation and Reasoning*. Elsevier.

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

De Raedt, L. 2008. *Logical and Relational Learning*. Springer.

Decatur, S. E., and Gennaro, R. 1995. On learning from noisy and incomplete examples. In *Eighth Conference on Computational Learning Theory (COLT)*, 353–360.

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.

Goldman, S. A., and Sloan, R. H. 1995. Can PAC learning algorithms tolerate random attribute noise? *Algorithmica* 14(1):70–84.

Juba, B.; Le, H. S.; and Stern, R. 2021. Safe learning of lifted action models. arXiv:2107.04169 [cs.AI].

Li, J.; Tinka, A.; Kiesel, S.; Durham, J. W.; Kumar, T. S.; and Koenig, S. 2020. Lifelong multi-agent path finding in large-scale warehouses. In *International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, 1898–1900.

McDermott, D.; Ghallab, M.; Howe, A.; Knoblock, C.; Ram, A.; Veloso, M.; Weld, D.; and Wilkins, D. 1998. PDDL-the planning domain definition language. Technical report, AIPS '98 - The Planning Competition Committee.

McDermott, D. 2000. The 1998 AI planning systems competition. *AI Magazine* 21(2):13.

Michael, L. 2010. Partial observability and learnability. *Artificial Intelligence* 174(11):639–669.

Moldovan, T. M., and Abbeel, P. 2012. Safe exploration in Markov decision processes. In *International Conference on Machine Learning (ICML)*, 1451–1458.

Muggleton, S., and De Raedt, L. 1994. Inductive logic programming: Theory and methods. *The Journal of Logic Programming* 19:629–679.

Rintanen, J. 2014. Madagascar: Scalable planning with SAT. In *8th International Planning Competition (IPC)*.

Shackelford, G., and Volper, D. 1988. Learning k-DNF with noise in the attributes. In *First Workshop on Computational Learning Theory (COLT)*, 97–103.

Stern, R., and Juba, B. 2017. Efficient, safe, and probably approximately complete learning of action models. In *the International Joint Conference on Artificial Intelligence (IJCAI)*, 4405–4411.

Turchetta, M.; Berkenkamp, F.; and Krause, A. 2016. Safe exploration in finite markov decision processes with gaussian processes. In *Advances in Neural Information Processing Systems*, 4312–4320.

Valiant, L. G. 1984. A theory of the learnable. *Commun. ACM* 27(11):1134–1142.

Wachi, A.; Sui, Y.; Yue, Y.; and Ono, M. 2018. Safe exploration and optimization of constrained MDPs using Gaussian processes. In *AAAI Conference on Artificial Intelligence (AAAI)*, 6548–6555.

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., and Kambhampati, S. 2013. Action-model acquisition from noisy plan traces. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2444–2450.