Safe Learning of Lifted Action Models

Brendan Juba[1], **Hai S. Le**[1], **Roni Stern**[2,3]
[1]**Washington University in St. Louis, 1 Brookings Dr., St. Louis, MO, USA, {bjuba,hsle}@wustl.edu**
[2]**Palo Alto Research Center, 3333 Coyote Hill Rd., Palo Alto, CA, USA, rstern@parc.com**
[3]**Ben Gurion University of the Negev, Be'er Sheva, Israel, 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 use learning, but learning approaches frequently do not provide guarantees of safety: variously, actions may fail or may not lead to the desired outcome. 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 demonstrating that our approach scales favorably in practice.

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. The best-known formulation is Reinforcement Learning (RL). In RL, an agent collects observations about the world by performing actions and observing their outcomes. The RL agent then uses these observations to decide how to act in the future. RL techniques have proven to be effective in a variety of domains, especially for low-level control tasks. However, RL generally does not consider the possibility of "failing," except insofar as the reward is sub-optimal. Similarly, most offline approaches that aim to learn a world model from past observations allow generating failing actions (Amir and Chang 2008).[1] In some domains, this is not a problem and the agent simply incorporates such experiences and updates its internal model to improve future executions. In other domains, however, execution failure *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 having a domain model, is called *safe model-free planning* (Stern and Juba 2017). Instead of a domain model, in safe model-free planning 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 bounded the probability of encountering such problems given a number of trajectories quasi-linear in the number of actions.

Their positive result is limited to *grounded* domain models, that is, domains that are not defined by *lifted*, i.e., parameterized, actions and fluents. It is possible to generate a grounded domain model from a given lifted domain model and problem. But, the size of the resulting grounded domain model can be arbitrarily larger than the 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. This significantly limits the applicability of Stern and Juba's algorithm.

In this work, we generalize their approach and propose 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

[1]They also did not consider lifted action models as we do.

algorithm *Safe Action Model (SAM)* Learning.

Two versions of SAM learning are presented. The first may be used when the mapping from lifted fluents to grounded fluents can be inverted. 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 small planning benchmarks. Finally, we discuss a more general version of SAM learning, for the case where the mapping from lifted fluents to grounded fluents cannot be inverted.

Background and Problem Definition

We define a classical planning **domain** by a tuple $\langle T, O, \mathcal{F}, \mathcal{A}, M \rangle$ where T is a set of types, O is a set of objects, $\mathcal{F}$ is a set of lifted fluents, $\mathcal{A}$ is a set of lifted actions, and M is an action model for $\mathcal{A}$.

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.

Lifted and Grounded Literals

A *lifted fluent* F is a relation over a list of types. These types are called the *parameters* of F and denoted by *params*(F). For example, in the logistics domain $at(?truck, ?location)$ is a lifted fluent that represents some truck ($?truck$) is at some location ($?location$). 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 same 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 relation over the objects in the image of b that match the relation over the corresponding parameters. We call this relation a *grounded fluent* or simply a fluent, and denote it by f. In our logistics example, for $F = at(?truck, ?location)$ and $b = \{?truck : truck1, ?location : loc1\}$ the corresponding grounded fluent f is $at(truck1, loc1)$. A *state* of the world is a set of grounded fluents. The term *literal* refers to either a fluent or its negation. The definitions of binding, lifted, and grounded fluent transfer naturally to literals.

Lifted and Grounded Actions

A lifted action $A \in \mathcal{A}$ is a pair $\langle$*name*, *params*$\rangle$ where *name* is a string and *params* is a list of types, denoted *name*(A) and *params*(A), respectively. 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 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 $\langle A, b \rangle$ 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:

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

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

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

The trajectory created by applying π to a state s is the sequence $\langle s_0, a_1, \dots, a_{|\pi|}, s_{|\pi|} \rangle$ 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, 1995; Stern and Juba 2017; Walsh and Littman 2008), it is common to represent a trajectory $\langle s_0, a_1, \dots, a_{|\pi|}, s_{|\pi|} \rangle$ as a set of triples $\left\{ \langle s_{i-1}, a_i, s_i \rangle \right\}_{i=1}^{|\pi|}$. 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 action a. $\mathcal{T}(A)$ is similarly defined for all action triplets that contain actions that are groundings of A.

A classical planning **problem** is a tuple $\langle D, s_I, s_g \rangle$ where D is a classical planning domain, s_I is the start state, i.e., the state of the world before planning, and s_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' that contains all the grounded literals in s_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 $s_g \subseteq s_G$). The *safe model-free planning* problem (Stern and Juba 2017) is defined as follows.

Definition 2 (Safe model-free planning)**.** *Let* $\Pi = \langle \langle T, O, \mathcal{F}, \mathcal{A}, M^* \rangle, s_I, s_g \rangle$ *be a classical planning problem and let* $\mathcal{T} = \{\mathcal{T}^1, \dots, \mathcal{T}^m\}$ *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 $\langle T, O, s_I, s_g, \mathcal{T} \rangle$ *and the desired output is a plan* π *that is a solution to* Π*. We denote this safe model-free planning problem as* $\Pi_{\mathcal{T}}$.

The main challenge in this problem is that the problem-solver – the agent – needs to find a sound plan for a planning problem but it is not given the set of fluents, actions, and action model of the domain ($\mathcal{F}$, $\mathcal{A}$, and M^*, respectively). We assume that 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.

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 recall their approach.

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:* $\forall l \in pre(a) \forall s : l \in s$
2. *All effects are satisfied:* $\forall l \in eff(a) \forall s' : l \in s'$
3. *Frame axioms hold:* $\forall (l \notin eff(a) \wedge l \notin s) \rightarrow l \notin s'$

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].* $\forall l \notin s' : l \notin eff(a)$
- *Rule 3 [must be an effect].* $\forall l \in s' \setminus s : l \in eff(a)$

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.

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*

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

Definition 4 says that if a is applicable in s according to a safe action model (M'), then (1) a is also applicable in s according to the action model M, and (2) the state resulting by applying a to s is the same 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* algorithm (Stern and Juba 2017) for 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 implement this algorithm, Stern and Juba (2017) proposed an algorithm for learning a safe action model, that we refer to as the *Safe Action-Model (SAM) Learning* algorithm.

SAM Learning for Grounded Domains

SAM Learning works as follows. First, it assumes every action a has all literals as its preconditions and no literals as its effects. Then, it iterates over every action triplet in $\mathcal{T}(a)$ and applies the rules in Observation 1 to remove incorrect preconditions and to add effects.

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

The main limitation of using a safe action model to generate plans is that it may be more restrictive than the real action model (M^*). That is, there may be states in which an action a is applicable according to M, but not according to the safe action model. Consequently, there may be planning problems that are solvable with M, but not with the safe action model. Thus, if an action model M' is safe w.r.t. another action model M then in some sense it is *weaker*. Next, we complement Theorem 1 by showing that every safe action model that is consistent with the given trajectories must be weaker than the action model returned by SAM learning.

Clearly, in the fully observable deterministic world of classical planning, every action model that is not consistent with the given set of trajectories is false. Moreover, there exists a trajectory in the domain in which using such a false action model will yield a failure. Thus, the set of consistent action models must contain the real action model.

Theorem 2 (SAM Learning is complete)**.** *Let* M_{SAM} *be the action model created by SAM learning given the set of trajectories* $\mathcal{T}$*. 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* M_{SAM}.

Proof. Consider an action model M' that is consistent with $\mathcal{T}$ and safe w.r.t. M^*. Let a be an action and s be a state such that a is applicable in s according to M', i.e., $pre_{M'}(a) \in s$. Since M' is safe w.r.t. M^*, $\Big(pre_{M^*}(a) \subseteq s \wedge a_{M'}(s) = a_{M^*}(s) \Big)$ By construction of M_{SAM}, if a literal l is a precondition of a according to M_{SAM}, 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 $pre_{M'}(a) \subseteq pre_{M_{SAM}}(a)$, and thus a is applicable in s according to M_{SAM}, i.e., $pre_{M_{SAM}}(a) \in s$. Since M_{SAM} is safe, $a_{M_{SAM}}(s) = a_{M^*}(s) = a'_M(s)$. □

Theorem 2 says that every action-model learning algorithm is bound to either return an unsafe action model or

return a action model model that is weaker than the action model returned by SAM learning. However, the action model returned by SAM may still be weaker than the real action model, and 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 SAM 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). However, in a lifted domain, the real action model is assumed to be defined only over lifted actions. This enables us to generalize SAM 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.

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 SAM learning to lifted domains. To describe our SAM learning algorithm for lifted domains, we denote by $\mathit{bindings}(b_A, b_L)$ the set of all parameter bindings $b_{L,A}$ that satisfy the following

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

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].* $\forall \langle L, b_L \rangle \notin s$:

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

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

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

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

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

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

Definition 5 (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.,

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

This simplifies the inference rules given in Observation 2. In particular, the "an effect" rule (Rule 1) becomes

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

Algorithm 1: Safe Action-Model (SAM) Learning

Input : $\Pi_{\mathcal{T}} = \langle T, O, s_I, s_g, \mathcal{T} \rangle$
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 $\mathit{eff}(A) \leftarrow \emptyset$
4 $\mathit{pre}(A) \leftarrow$ all parameter-bound literals
5 **foreach** $(s, \langle A, b_A \rangle, s') \in \mathcal{T}(A)$ **do**
6 **foreach** $\langle L, b_{L,A} \rangle \in \mathit{pre}(A)$ **do**
7 **if** $\langle L, b_A \circ b_{L,A} \rangle \notin s$ **then**
8 Remove $\langle L, b_{L,A} \rangle$ from $\mathit{pre}(A)$
9 **foreach** $\langle L, b_L \rangle \in s' \setminus s$ **do**
10 $b_{L,A} \leftarrow \langle L, (b_A)^{-1} \circ b_L \rangle)$
11 Add $\langle L, b_{L,A} \rangle$ to $\mathit{eff}(A)$
12 **return** $(\mathit{pre}, \mathit{eff})$

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 2 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 1 in Observation 2). Note that Rule 3 in Observation 2 is not needed since we initialize the set of effects of every action to be an empty set.

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

Proof. We first show by induction on the iterations of the loop in lines 5–11 that on every iteration

$$\mathit{pre}_{M^*}(A) \subseteq \mathit{pre}(A) \text{ and } \mathit{eff}(A) \subseteq \mathit{eff}_{M^*}(A) \tag{9}$$

where M^* is the correct action model. Prior to the first iteration, the preconditions of all lifted actions A are all possible parameter-bound literals, so $\mathit{pre}(A)$ must be a subset of $\mathit{pre}_{M^*}(A)$. (Note that this includes every parameter-bounded fluent *and* its negation.) Similarly, the effects are

[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 in our simple logistics example.

set to $\emptyset$, which is surely a subset of $\mathit{eff}_{M^*}(A)$. The changes made to $\mathit{pre}(A)$ and $\mathit{eff}(A)$ in subsequent iterations are encapsulated in lines 8 and 11 in Algorithm 1. Line 8 is a direct application of Rule 1 ("not a precondition") from Observation 2, and thus $\mathit{pre}(A)$ is still a subset of pre_{M^*}. Similarly, line 11 is an application of Rule 3 ("an effect") in the same observation, given that $\mathit{bindings}(b_A, b_L)$ consists of a single parameter binding due to the injective binding assumption. This completes the induction.

Let $(\mathit{pre}, \mathit{eff})$ be the action model returned by Algorithm 1 (line 12). From the induction above (Eq. 9) it immediately follows that for every grounded action $\langle A, b_A \rangle$ and state s, if $\mathit{pre}(\langle A, b_A \rangle) \subseteq s$ then $\mathit{pre}_{M^*}(\langle A, b_A \rangle) \subseteq s$. From the induction above, all the parameter-bound literals in $\mathit{eff}(A)$ are indeed effects of A. Finally, consider any parameter-bound literal $\langle L, b_{L,A} \rangle$ that is an effect of A but is absent from $\mathit{eff}(A)$, i.e., every $\langle L, b_{L,A} \rangle \in \mathit{eff}_{M^*}(A) \setminus \mathit{eff}(A)$. By construction of eff, this can only occur if this parameter-bound literal was true in all pre-states of groundings of A in all the available trajectories. Consequently, $\langle L, b_{L,A} \rangle$ must be in $\mathit{pre}(A)$. Therefore, every grounded literal in the post-state of applying $\langle A, b_A \rangle$ in s (i.e., $\langle A, b_A \rangle_{M^*}(s)$) is either in $\mathit{eff}(\langle A, b_A \rangle)$ or $\mathit{pre}(\langle A, b_A \rangle)$. □

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 in the location and the package is on the track, respectively. There are three possible actions: *Move*, *Load*, and *Unload*. Table 1 lists the parameters, preconditions, and effects of these actions. 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 correct 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 correct 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 correct 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), SAM Learning for grounded domains will require at least 10 trajectories to learn an action model with all of the actions.

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] Let $arity(F, t)$ and $arity(A, t)$ be the number of type-t parameters of the lifted fluent F and action A.

Theorem 4. *Under the injective 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 is drawn from* $\mathcal{P}_D$ *that is not solvable with* M_{SAM} *with probability at most* ϵ*.*

Theorem 4 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. Indeed, there are $arity(A, t)^{arity(F,t)}$ ways to bind the parameters of F of type t to the parameters of A, and hence $\prod_{t \in T} arity(A, t)^{arity(F,t)}$ ways of binding all of the parameters of F to parameters of A. The preconditions and effects are sets of these parameter-bound fluents. For example, in 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 4.

Definition 6 (Adequate)**.** *An action model* M *is* ϵ-adequate *if, with probability at most* ϵ*, a trajectory* T *sampled from* $\mathcal{T}_D$

[3] The planner need not be deterministic.

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 4) is ϵ-adequate with probability at least $1 - \delta$.*

Proof. Consider any action model M_B that may be returned by SAM Learning but is not ϵ adequate. By definition, the probability of drawing a trajectory from $\mathcal{T}_D$ that is inconsistent with M_B is at least ϵ. Thus, the probability of drawing m samples that are consistent with M_B is at most

$$(1 - \epsilon)^m \leq e^{-m \cdot \epsilon}. \tag{10}$$

M_B can only be returned if this occurs. For our choice of m,

$$e^{-m \cdot \epsilon} \leq e^{-(\ln 3L + \ln \frac{1}{\delta})} = \frac{\delta}{3^L} \tag{11}$$

where

$$L = 2 \sum_{\substack{F \in \mathcal{F} \\ A \in \mathcal{A}}} \prod_{t \in T} arity(A, t)^{arity(F,t)}$$

Let B be the set of action models that are not ϵ-adequate. By a union bound over B, the probability that SAM Learning will return an action model that is not ϵ-adequate is at most $\frac{|B|\delta}{3^L}$. For each parameter-bound fluent, each precondition or effect will either contain that fluent, or its negation, or neither of them. Hence, the number of possible action models is 3^L. Since B is a set of action models, we have that the size of B is at most 3^L. Therefore, the probability that SAM Learning will return an action model that is not ϵ-adequate is at most δ. □

Proof of Theorem 4. Let M be an action model returned by SAM Learning given m samples. Thus, M is a safe action model (Theorem 3) 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 □

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 1 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 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 $L(x)$. Next, we describe Extended SAM Learning, which is able to capture the above form of inference and is applicable to cases where the injective action-binding assumption does not hold.

Extended SAM Learning

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

Safe Action Model Constraints $CNF_{pre}(A)$ 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, $CNF_{eff}(A)$ 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, $CNF_{pre}(A)$ and $CNF_{eff}(A)$ 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 CNF_{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 ¬IsEff($\langle L, b_{L,A} \rangle$) to CNF_{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 CNF_{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 both CNFs by applying unit propagation and removing subsumed clauses.

Proxy Actions The main challenge in creating a safe action model from the generated CNFs is the disjunction in CNF_{eff}, which represents uncertainty w.r.t to the effects of

[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.

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. 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 non-subsumed non-unit clauses. (The number of proxy actions is thus exponential in the number of non-unit clauses.) In this proxy action, we identify all of the parameters that appear together in a clause in the subset; if multiple clauses share any variables, we identify all of the parameters across the clauses. 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 clauses 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. Algorithm 2 lists the complete pseudocode of E-SAM learning.

Theoretical Properties The E-SAM Learning action model satisfies the same properties as the original model (assuming injective bindings), captured in Theorems 3 and 2:

Theorem 5. *The E-SAM Learning action model is safe.*

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 3. □

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 cut inferences (a.k.a., resolution inferences) from these clauses. By the completeness of resolution for prime implicates (see, e.g., (Brachman and Levesque 2004, Chapter 13, Exercise 1)), all of the prime implicates of CNF_{eff} can be derived by applications of cut. In turn, therefore, unit propagation can also derive all of the prime implicates of CNF_{eff}. □

Theorem 6. *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.*

Algorithm 2: Extended SAM Learning

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

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

Proof. Let M' be an action model that is consistent with $\mathcal{T}$ and safe w.r.t. M^*, and consider any transition $\langle s, \langle A, b\rangle, s'\rangle$ permitted by M'. Consider the set S of literals in $s' \setminus s$ that do not correspond to unit clauses in the CNF created by E-SAM Learning, and the set $\bar{S}$ of literals that are the groundings under b of the effects in the non-unit clauses created by E-SAM learning that are not in $s' \setminus s$. Recall, Observation 2 characterizes the set action models consistent with $\mathcal{T}$ and by Lemma 2, no sub-clause of the CNF created by E-SAM learning is entailed by the rules of Observation 2. Therefore, for every literal of every non-unit clause of this CNF, there exists an action model consistent with $\mathcal{T}$ in which that literal is the only satisfied literal of the clause. (Otherwise, a strictly smaller clause would be entailed.) Therefore, for each literal $l \in S$, since M' is safe w.r.t. M^*, all of the parameters of A in some clause for this effect must be bound to the objects necessary to obtain l as the corresponding effect. Thus, b must be consistent with at least one of the proxy actions A_{proxy}. Furthermore, since the literals in $\bar{S}$ may be effects of $\langle A, b\rangle$, if they are not in $s' \setminus s$, they must be in s, so the preconditions of A_{proxy} are satisfied as well. Since the E-SAM Learning action model is safe by Theorem 5, the post-state of A_{proxy} is therefore equal to that obtained by the true action model, which is in turn also equal to s' since M' is also safe. A is therefore an application of A_{proxy}, and we see that the use of A in M' is safe with respect to the set of proxy actions in the E-SAM Learning action model. □

Experiments

To evaluate the performance of SAM Learning for lifted domains (Algorithm 1), we performed experiments on two simple IPC domains: *N-Puzzle* (3×3, 3 predicates, 1 action) and *Blocksworld* (8 blocks, 5 predicates, 4 actions). For each domain, we generated 10 trajectories with 10 actions each by taking random actions (all distinct lifted actions of a domain appear at least once in each trajectory). Then, we ran

SAM Learning on these trajectories and obtained a safe action model. As a baseline, we used FAMA (Aineto, Celorrio, and Onaindia 2019), which is a modern algorithm for learning an action model from trajectories. Note that FAMA has no safety guarantee. For N-Puzzle, both methods correctly learned the action model after observing a single $\langle s, a, s' \rangle$ triple in all 10 trials. For Blocksworld, Table 2 lists the number of state-action-state triples needed to learn a correct action model over 10 runs, where each run processed the trajectories in a random order.

# $\langle s, a, s' \rangle$	6	7	8	9	$\cdots$	13	14	15	16	17
SAM	1	1	6	2	0	0	0	0	0	0
FAMA	0	0	0	0	0	3	1	3	1	2

Table 2: # trials in which SAM Learning and FAMA learned the true Blocksworld action model with a given # of triples.

In all cases, SAM learning was able to recover a correct action model using fewer (s, a, s') triplets than FAMA. Note that once SAM Learning finds a correct model, it will not add more literals to the preconditions or effects, since SAM only removes literals that are not satisfied in the pre-state and adds literals that switch values between pre and post-states. Meanwhile, FAMA might add irrelevant literals to the preconditons or effects as it processes more transitions.

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

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 (which are often identified with negative "reward") (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.

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 knowledge of 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. We will also examine its performance on more complicated IPC domains.

Acknowledgments

This research is partially funded by NSF award IIS-1908287 and BSF grant #2018684 to Roni Stern.

References

Aineto, D.; Celorrio, S.; and Onaindia, E. 2019. Learning action models with minimal observability. *Artificial Intelligence* 275. doi:10.1016/j.artint.2019.05.003.

Amir, E.; and Chang, A. 2008. Learning Partially Observable Deterministic Action Models. *J. Artif. Intell. Res. (JAIR)* 33: 349–402.

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

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.

McDermott, D. 2000. The 1998 AI Planning Systems Competition. *AI Magazine* 21(2): 13.

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.

Moldovan, T. M.; and Abbeel, P. 2012. Safe exploration in Markov decision processes. In *Proceedings of the 29th International Conference on Machine Learning*, 1451–1458.

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.

Wachi, A.; Sui, Y.; Yue, Y.; and Ono, M. 2018. Safe exploration and optimization of constrained mdps using gaussian processes. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

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

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

Wang, X. 1995. Learning by observation and practice: an incremental approach for planning operator acquisition. In *Proceedings of the Twelfth International Conference on International Conference on Machine Learning*, 549–557.