

One-shot learning: From domain knowledge to action models

Abstract

Most approaches to learning action planning models heavily rely on a significantly large volume of training samples or plan observations. In this paper, we adopt a different approach based on deductive learning from domain-specific knowledge, specifically from logic formulae that specify constraints about the possible states of a given domain. The minimal input observability required by our approach is a single example composed of a full initial state and a partial goal state. We will show that exploiting specific domain knowledge enable to constrain the space of possible action models as well as to complete partial observations, both of which turn out helpful to learn good-quality action models.

1 Introduction

The learning of action models in planning has been typically addressed with inductive learning data-intensive approaches. From the pioneer learning system ARMS [Yang *et al.*, 2007] to more recent ones [Mourão *et al.*, 2012; Zhuo and Kambhampati, 2013; Kucera and Barták, 2018], all of them require thousands of plan observations or training samples, i.e., sequences of actions as evidence of the execution of an observed agent, to obtain and validate an action model. These approaches return the statistically significant model that best explains the plan observations by minimizing some error metric. A model explains an observation if a plan containing the observed actions is computable with the model and the states induced by this plan also include the possibly partially observed states. The limitation of posing model validation as an optimization task over a testing set of observations is that it neither guarantees completeness (the model may not explain all the observations) nor correctness (the states induced by the execution of the plan generated with the model may contain contradictory information).

Differently, other approaches rely on symbolic-via learning. The Simultaneous Learning and Filtering (SLAF) approach [Amir and Chang, 2008] exploits logical inference and builds a complete explanation through a CNF formula that represents the initial belief state, and a plan observation. The

formula is updated with every action and state of the observation, thus representing all possible transition relations consistent with it. SLAF extracts all satisfying models of the learned formula with a SAT solver although the algorithm cannot effectively learn the preconditions of actions. A more recent approach addresses the learning of action models from plan observations as a planning task which searches the space of all possible action models [Aineto *et al.*, 2018]. A plan here is conceived as a series of steps that determine the preconditions and effects of the action models plus other steps that validate the formed actions in the observations. The advantage of this approach is that it only requires input samples of about a total of 50 actions.

This paper studies the impact of using mixed input data, i.e., automatically-collected plan observations and human-encoded domain-specific knowledge, in the learning of action models. Particularly, we aim to stress the extreme case of having a single observation sample and answer the question to whether the lack of training samples can be overcome with the supply of domain knowledge. The question is motivated by (a) the assumption that obtaining enough training observations is often difficult and costly, if not impossible in some domains [Zhuo, 2015]; (b) the fact that although the physics of the real-world domain being modeled are unknown, the user may know certain pieces of knowledge about the domain; and (c) the desire for correct action models that are usable beyond their applicability to a set of testing observations. To this end, we opted for checking our hypothesis in the framework proposed in [Aineto *et al.*, 2018] since this planning-based satisfiability approach allows us to configure additional constraints in the compilation scheme, it is able to work under a minimal set of observations and uses an off-the-shelf planner¹. Ultimately, we aim to compare the informational power of domain observations (information quantity) with the representational power of domain-specific knowledge (information quality). Complementarily, we restrict our attention to solely observations over fluents as in many applications the actual actions of an agent may not be observable [Sohrabi *et al.*, 2016].

Next section summarizes basic planning concepts and outlines the baseline learning approach [Aineto *et al.*, 2018].

¹We thank authors for providing us with the source files of their learning system.

Then we formalize our one-shot learning task with domain knowledge and subsequently we explain the task-solving process. Section 5 presents the experimental evaluation and last section concludes.

2 Background

We denote as F (fluents) the set of propositional state variables. A partial assignment of values to fluents is represented by L (literals). We adopt the *open world assumption* (what is not known to be true in a state is unknown) to implicitly represent the unobserved literals of a state. Hence, a state s includes positive literals (f) and negative literals ($\neg f$) and it is defined as a full assignment of values to fluents; $|s| = |F|$. We use $\mathcal{L}(F)$ to denote the set of all literal sets on F ; i.e. all partial assignments of values to fluents.

A *planning action* a has a precondition list $\text{pre}(a) \in \mathcal{L}(F)$ and a effect list $\text{eff}(a) \in \mathcal{L}(F)$. The semantics of an action a is specified with two functions: $\rho(s, a)$ denotes whether a is *applicable* in a state s and $\theta(s, a)$ denotes the *successor state* that results from applying a in a state s . Then, $\rho(s, a)$ holds iff $\text{pre}(a) \subseteq s$, i.e. if its preconditions hold in s . The result of executing an applicable action a in a state s is a new state $\theta(s, a) = \{s \setminus \neg\text{eff}(a) \cup \text{eff}(a)\}$, where $\neg\text{eff}(a)$ is the complement of $\text{eff}(a)$, which is subtracted from s so as to ensure that $\theta(s, a)$ remains a well-defined state. The subset of effects of an action a that assign a positive value to a fluent is called *positive effects* and denoted by $\text{eff}^+(a) \in \text{eff}(a)$ while $\text{eff}^-(a) \in \text{eff}(a)$ denotes the *negative effects*.

A *planning problem* is a tuple $P = \langle F, A, I, G \rangle$, where I is the initial state and $G \in \mathcal{L}(F)$ is the set of goal conditions over the state variables. A *plan* π is an action sequence $\pi = \langle a_1, \dots, a_n \rangle$, with $|\pi| = n$ denoting its *plan length*. The execution of π in I induces a *trajectory* $\langle s_0, a_1, s_1, \dots, a_n, s_n \rangle$ such that $s_0 = I$ and, for each $1 \leq i \leq n$, it holds $\rho(s_{i-1}, a_i)$ and $s_i = \theta(s_{i-1}, a_i)$. A plan π solves P iff the induced trajectory reaches a final state s_n such that $G \subseteq s_n$.

The baseline learning approach our proposal draws upon uses *actions with conditional effects* [Aineto et al., 2018]. The conditional effects of an action a_c is composed of two sets of literals: $C \in \mathcal{L}(F)$, the *condition*, and $E \in \mathcal{L}(F)$, the *effect*. The *triggered effects* resulting from the action application (conditional effects whose conditions hold in s) is defined as $\text{eff}_c(s, a) = \bigcup_{C \supseteq E \in \text{cond}(a_c), C \subseteq s} E$.

2.1 Learning action models as planning

The approach for learning STRIPS action models presented in [Aineto et al., 2018], which we will use as our baseline learning system (hereafter BLS, for short), is a compilation scheme that transforms the problem of learning the preconditions and effects of action models into a planning task P' . A STRIPS *action model* ξ is defined as $\xi = \langle \text{name}(\xi), \text{pars}(\xi), \text{pre}(\xi), \text{add}(\xi), \text{del}(\xi) \rangle$, where $\text{name}(\xi)$ and parameters, $\text{pars}(\xi)$, define the header of ξ ; and $\text{pre}(\xi)$, $\text{del}(\xi)$ and $\text{add}(\xi)$ are sets of fluents that represent the *preconditions*, *negative effects* and *positive effects*, respectively, of the actions induced from the action model ξ .

The BLS receives as input an empty domain model, which only contains the headers of the action models,

and a set of observations of plan executions, and creates a propositional encoding of the planning task P' . Let Ψ be the set of *predicates*² that shape the variables F . The set of propositions of P' that can appear in $\text{pre}(\xi)$, $\text{del}(\xi)$ and $\text{add}(\xi)$ of a given ξ , denoted as $\mathcal{I}_{\xi, \Psi}$, are FOL interpretations of Ψ over the parameters $\text{pars}(\xi)$. For instance, in a four-operator *blocksworld* [Slaney and Thiébaux, 2001], the $\mathcal{I}_{\xi, \Psi}$ set contains five elements for the `pickup(v_1)` model, $\mathcal{I}_{\text{pickup}, \Psi} = \{\text{handempty}, \text{holding}(v_1), \text{clear}(v_1), \text{ontable}(v_1), \text{on}(v_1, v_1)\}$ and eleven elements for the model of `stack(v_1, v_2)`, $\mathcal{I}_{\text{stack}, \Psi} = \{\text{handempty}, \text{holding}(v_1), \text{holding}(v_2), \text{clear}(v_1), \text{clear}(v_2), \text{ontable}(v_1), \text{ontable}(v_2), \text{on}(v_1, v_1), \text{on}(v_1, v_2), \text{on}(v_2, v_1), \text{on}(v_2, v_2)\}$. Hence, solving P' consists in determining which elements of $\mathcal{I}_{\xi, \Psi}$ will shape the preconditions, positive and negative effects of the action model ξ .

The decision as to whether or not an element of $\mathcal{I}_{\xi, \Psi}$ will be part of $\text{pre}(\xi)$, $\text{del}(\xi)$ or $\text{add}(\xi)$ is given by the plan that solves P' . Specifically, two different sets of actions are included in the definition of P' : *insert actions*, which insert preconditions and effects on an action model; and *apply actions*, which validate the application of the learned action models in the input observations. Roughly speaking, in the *blocksworld* domain, the insert actions of a plan that solves P' will look like `(insert_pre_stack_holding_v1), (insert_eff_stack_clear_v1), (insert_eff_stack_clear_v2)`, where the second action denotes a positive effect and the third one a negative effect both to be inserted in the model of `stack`; and the second set of actions of the plan that solves P' will be like `(apply_unstack_blockB_blockA), (apply_putdown_blockB)` and `(validate_1), (validate_2)`, where the last two actions denote the points at which the states generated through the `apply` actions must be validated with the observations of plan executions.

In a nutshell, the output of the BLS compilation is a plan that completes the empty input domain model by specifying the preconditions and effects of each action model such that the validation of the completed model over the input observations is successful.

3 One-shot learning task

The *one-shot* learning task to learn action models from *domain-specific knowledge* is defined as a tuple $\Lambda = \langle \mathcal{M}, \mathcal{O}, \Phi \rangle$, where:

- \mathcal{M} is the *initial empty model* that contains only the header of each action model to be learned.
- \mathcal{O} is a single learning example or plan observation; i.e. a sequence of (partially) observable states representing the evidence of the execution of an observed agent.
- Φ is a set of logic formulae that define *domain-specific knowledge*.

²The initial state of an observation is a full assignment of values to fluents, $|s_0| = |F|$, and so the predicates Ψ are extractable from the observed state s_0 .

Combination	Meaning
$\neg pre_e \wedge \neg eff_e$	e belongs neither to the preconditions nor effects of ξ ($e \notin pre(\xi) \wedge e \notin add(\xi) \wedge e \notin del(\xi)$)
$pre_e \wedge \neg eff_e$	e is only a precondition of ξ ($e \in pre(\xi) \wedge e \notin add(\xi) \wedge e \notin del(\xi)$)
$\neg pre_e \wedge eff_e$	e is a positive effect of ξ ($e \notin pre(\xi) \wedge e \in add(\xi) \wedge e \notin del(\xi)$)
$pre_e \wedge eff_e$	e is a negative effect of ξ ($e \in pre(\xi) \wedge e \notin add(\xi) \wedge e \in del(\xi)$)

Figure 1: Combinations of the fluent propositional encoding and their meaning

A *solution* to a learning task $\Lambda = \langle \mathcal{M}, \mathcal{O}, \Phi \rangle$ is a model \mathcal{M}' s.t. there exists a plan computable with \mathcal{M}' that is consistent with the headers of \mathcal{M} , the observed states of \mathcal{O} and the given domain knowledge in Φ .

3.1 The space of STRIPS action models

We analyze here the search space of a learning task $\Lambda = \langle \mathcal{M}, \mathcal{O}, \Phi \rangle$; i.e., the space of STRIPS action models. In principle, for a given action model ξ , any element of $\mathcal{I}_{\xi, \Psi}$ can potentially appear in $pre(\xi)$, $del(\xi)$ and $add(\xi)$. In practice, the actual space of possible STRIPS schemata is bounded by:

1. **Syntactic constraints.** The solution \mathcal{M}' must be consistent with the STRIPS constraints: $del(\xi) \subseteq pre(\xi)$, $del(\xi) \cap add(\xi) = \emptyset$ and $pre(\xi) \cap add(\xi) = \emptyset$. *Typing constraints* would also be a type of syntactic constraint [McDermott *et al.*, 1998].
2. **Observation constraints.** The solution \mathcal{M}' must be consistent with these *semantic constraints* derived from the learning samples \mathcal{O} , which in our case is a single plan observation. Specifically, the states induced by the plan computable with \mathcal{M}' must comprise the observed states of the sample.

Considering only the syntactic constraints, the size of the space of possible STRIPS models is given by $2^{2 \times |\mathcal{I}_{\Psi, \xi}|}$ because one element in $\mathcal{I}_{\xi, \Psi}$ can appear both in the preconditions and effects of ξ . In this work, the belonging of an $e \in \mathcal{I}_{\Psi, \xi}$ to the preconditions, positive effects or negative effects of ξ is handled with a refined propositional encoding that uses fluents of two types, pre_e and eff_e , instead of the three fluents used in the BLS. The four possible combinations of these two fluents are summarized in Figure 1. This compact encoding allows for a more effective exploitation of the syntactic constraints, and also yields the solution space of Λ to be the same as its search space.

Additionally, observation samples further constrains the space of possible action models.

3.2 The sampling space

The single plan observation of \mathcal{O} is defined as $\mathcal{O} = \langle s_0^o, s_1^o, \dots, s_m^o \rangle$, a sequence of possibly *partially observed states* except for the initial state s_0^o which is a *fully observable* state. As commented before, the predicates Ψ and the objects that shape the fluents F are then deducible from s_0^o . A partially observed state s_i^o , $1 \leq i \leq m$, is one in which

$|s_i^o| < |F|$; i.e., a state in which at least a fluent of F was not observed. Intermediate states can be *missing*, meaning that they are unobservable, so transiting between two consecutive observed states in \mathcal{O} may require the execution of more than one action ($\theta(s_i^o, \langle a_1, \dots, a_k \rangle) = s_{i+1}^o$ (with $k \geq 1$ is unknown but finite). The minimal expression of a learning sample must comprise at least two state observations, a full initial state s_0^o and a partially observed final state s_m^o so $m \geq 1$.

Figure 2 shows a learning example that contains an initial state of the blocksworld where the robot hand is empty and three blocks (namely `blockA`, `blockB` and `blockC`) are on top of the table and clear. The observation represents a partially observable final state in which `blockA` is on top of `blockB` and `blockB` on top of `blockC`.

```
(:predicates (on ?x ?y) (ontable ?x)
  (clear ?x) (handempty)
  (holding ?x))

(:objects blockA blockB blockC)

(:init (ontable blockA) (clear blockA)
  (ontable blockB) (clear blockB)
  (ontable blockC) (clear blockC)
  (handempty))

(:observation (on blockA blockB) (on blockB blockC))
```

Figure 2: Example of a two-state observationn for the learning STRIPS action models.

3.3 The domain-specific knowledge

Our approach is to introduce *domain-specific knowledge* in the form of *state constraints* to further restrict the space of the action models. Back to the *blocksworld* domain, one can argue that `on` (v_1, v_1) and `on` (v_2, v_2) will not appear in the $pre(\xi)$, $del(\xi)$ and $add(\xi)$ of any action model ξ because, in this specific domain, a block cannot be on top of itself. The notion of state constraint is very general and has been used in different areas of AI and for different purposes. In planning, state constraints are compact and abstract representations that relate the values of variables in each state traversed by a plan, and allow to specify the set of states where a given action is applicable, the set of states where a given *axiom* or *derived predicate* holds or the set of states that are considered goal states [Haslum *et al.*, 2018].

State invariants is a useful type of state constraints for computing more compact state representations of a given planning problem [Helmert, 2009] and for making *satisfiability planning* or *backward search* more efficient [Rintanen, 2014; Alcázar and Torralba, 2015]. Given a planning problem $P = \langle F, A, I, G \rangle$, a state invariant is a formula ϕ that holds in I , $I \models \phi$, and in every state s built out of F that is reachable by applying actions of A in I .

A *mutex* (mutually exclusive) is a state invariant that takes the form of a binary clause and indicates a pair of different properties that cannot be simultaneously true [Kautz and Selman, 1999]. For instance in a three-block *blocksworld*, $\neg on(block_A, block_B) \vee \neg on(block_A, block_C)$ is a *mutex* because `blockA` can only be on top of a single block.

Recently, some works point at extracting *lifted* invariants, also called *schematic* invariants [Rintanen, 2017], that hold

for any possible state and any possible set of objects. Invariant templates obtained by inspecting the lifted representation of the domain have also been exploited for deriving *lifted mutex* [Bernardini *et al.*, 2018]. In this work we exploit domain-specific knowledge that is given as *schematic mutex*. We pay special attention to *schematic mutex* because they identify mutually exclusive properties of a given type of objects [Fox and Long, 1998] and because they enable (1) an effective completion of a partially observed state and (2) an effective pruning of inconsistent STRIPS action models.

We define a schematic mutex as a $\langle p, q \rangle$ pair where both $p, q \in \mathcal{I}_{\xi, \Psi}$ are predicates that shape the preconditions or effects of a given action scheme ξ and they satisfy the formulae $\neg p \vee \neg q$, considering that their corresponding variables are universally quantified. For instance, *holding*(v_1) and *clear*(v_1) from the *blocksworld* are *schematic mutex* while *clear*(v_1) and *ontable*(v_1) are not because $\forall v_1, \neg \text{clear}(v_1) \vee \neg \text{ontable}(v_1)$ does not hold for every possible state. Figure 3 shows four clauses that define schematic mutexes for the *blocksworld* domain.

$$\begin{aligned} &\forall x_1, x_2 \neg \text{ontable}(x_1) \vee \neg \text{on}(x_1, x_2). \\ &\forall x_1, x_2 \neg \text{clear}(x_1) \vee \neg \text{on}(x_2, x_1). \\ &\forall x_1, x_2, x_3 \neg \text{on}(x_1, x_2) \vee \neg \text{on}(x_1, x_3) \text{ such that } x_2 \neq x_3. \\ &\forall x_1, x_2, x_3 \neg \text{on}(x_2, x_1) \vee \neg \text{on}(x_3, x_1) \text{ such that } x_2 \neq x_3. \end{aligned}$$

Figure 3: *Schematic mutexes* for the *blocksworld* domain.

4 Action model learning from *schematic mutexes*

In this section, we show how to exploit *schematic mutexes* for solving the learning task $\Lambda = \langle \mathcal{M}, \mathcal{O}, \Phi \rangle$.

4.1 Completing partially observed states with *schematic mutexes*

The addition of new literals to complete the partial states $\langle s_1^o \dots, s_m^o \rangle$ of an observation \mathcal{O} using a set of schematic mutexes Φ is done in a pre-processing stage.

Let Ω be the set of objects that appear in F as the values of the arguments of the predicates Ψ , and $\Phi = \langle p, q \rangle$ a schematic mutex. There exist many possible instantiations of Φ of the type $\langle p(\omega), q(\omega') \rangle$ with objects of Ω , where $\omega \subseteq \Omega^{args(p)}$ and $\omega' \subseteq \Omega^{args(q)}$. Let us now assume that the instantiation $p(\omega) \in s_j^o$, ($1 \leq j \leq m$), being s_j^o a partially observed state of \mathcal{O} . Then, two situations may occur: (a) $q(\omega') \in s_j^o$, in which case the expression $\neg p(\omega) \vee \neg q(\omega')$ holds in s_j^o ; or (b) $q(\omega') \notin s_j^o$, in which case the literal $\neg q(\omega')$ has not been observed in s_j^o and so we can safely complete the state with $\neg q(\omega')$ (the same applies inversely, when $q(\omega') \in s_j^o$ but $\neg p(\omega) \notin s_j^o$). In other words, if we find that one component of a schematic mutex is positively observed in a state and the other component is not observable in such state, we can complete the state with the missing negative literal. For instance, if the literal *holding*(*blockA*) is observed in a particular state and Φ contains the schematic

ID	Action	New conditional effect
1	(insert.pre) $_{\xi, p}$	$\{pre_ \xi_ q\} \triangleright \{invalid\}$
2	(insert.eff) $_{\xi, p}$	$\{pre_ \xi_ q \wedge eff_ \xi_ q \wedge pre_ \xi_ p\} \triangleright \{invalid\}$
3	(insert.eff) $_{\xi, p}$	$\{\neg pre_ \xi_ q \wedge eff_ \xi_ q \wedge \neg pre_ \xi_ p\} \triangleright \{invalid\}$
4	(apply) $_{\xi, \omega}$	$\{\neg pre_ \xi_ p \wedge eff_ \xi_ p \wedge q(\omega) \wedge \neg pre_ \xi_ q\} \triangleright \{invalid\}$
5	(apply) $_{\xi, \omega}$	$\{\neg pre_ \xi_ p \wedge eff_ \xi_ p \wedge q(\omega) \wedge \neg eff_ \xi_ q\} \triangleright \{invalid\}$

Figure 4: Summary of the new conditional effects added to the classical planning compilation for the learning of STRIPS action models.

mutex $\neg \text{holding}(v_1) \vee \neg \text{clear}(v_1)$, we extend the state observation with literal $\neg \text{clear}(\text{blockA})$ (despite this particular literal being initially unknown).

4.2 Pruning inconsistent action models with *schematic mutexes*

Our approach to learning action models consistent with the schematic mutexes in Φ is to ensure that newly generated states induced by the learned actions do not introduce any inconsistency. This is implemented by adding new conditional effects to the insert and apply actions of the BLS compilation. Figure 4 summarizes the new conditional effects added to the compilation and next, we describe them in detail:

- 1-3 For every schematic mutex $\langle p, q \rangle$, where both p and q belong to $\in \mathcal{I}_{\xi, \Psi}$, one conditional effect is added to the (insert.pre) $_{\xi, p}$ actions to prevent the insertion of two preconditions that are schematic mutex. Likewise, two conditional effects are added to the (insert.eff) $_{\xi, p}$ actions, one to prevent the insertion of two positive effects that are schematic mutex and another one to prevent two mutex negative effects.
- 4-5 For every schematic mutex $\langle p, q \rangle$, where both p and q belong to $\in \mathcal{I}_{\xi, \Psi}$, two conditional effects are added to the (apply) $_{\xi, \omega}$ actions to prevent positive effects that are inconsistent with an input observation (in (apply) $_{\xi, \omega}$ actions the variables in $vars(\xi)$ are bounded to the objects in ω that appear in the same position).

In theory, conditional effects of the type 4-5 are sufficient to guarantee that all the states traversed by a plan produced by the compilation are *consistent* with the input set of schematic mutexes Φ (obviously provided that the input initial state s_0^o is a valid state). In practice we include also conditional effects of the type 1-3 because they prune *invalid* action models at an earlier stage of the planning process (these effects extend the insert actions that always appear first in the solution plans).

4.3 Compilation properties

Lemma 1. Soundness. *Any classical plan π that solves P' (planning task that results from the compilation) produces a model \mathcal{M}' that solves the $\Lambda = \langle \mathcal{M}, \mathcal{O}, \Phi \rangle$ learning task.*

Proof. According to the P' compilation, once a given precondition or effect is inserted into the domain model \mathcal{M} it cannot be undone. In addition, once an action model is applied it cannot be modified. In the compiled planning task P' , only (apply) $_{\xi, \omega}$ actions can update the value of the state fluents F . This means that a state consistent

with an observation s_n^o can only be achieved executing an applicable sequence of $(\text{apply})_{\xi,\omega}$ actions that, starting in the corresponding initial state s_0^o , validates that every generated intermediate state s_i , s.t. $0 \leq i \leq n$, is consistent with the input state observations and *state-invariants*. This is exactly the definition of the solution condition for model \mathcal{M}' to solve the $\Lambda = \langle \mathcal{M}, \mathcal{O}, \Phi \rangle$ learning task. \square

Lemma 2. Completeness. *Any model \mathcal{M}' that solves the $\Lambda = \langle \mathcal{M}, \mathcal{O}, \Phi \rangle$ learning task can be computed with a classical plan π that solves P' .*

Proof. By definition $\mathcal{I}_{\xi,\Psi}$ fully captures the set of elements that can appear in an action model ξ using predicates Ψ . In addition the P' compilation does not discard any model \mathcal{M}' definable within $\mathcal{I}_{\xi,\Psi}$ that satisfies the mutexes in Φ . This means that, for every model \mathcal{M}' that solves the $\Lambda = \langle \mathcal{M}, \mathcal{O}, \Phi \rangle$, we can build a plan π that solves P' by selecting the appropriate $(\text{insert_pre})_{\xi,e}$ and $(\text{insert_eff})_{\xi,e}$ actions for programming the precondition and effects of the corresponding action models in \mathcal{M}' and then, selecting the corresponding $(\text{apply})_{\xi,\omega}$ actions that transform the initial state observation s_0^o into the final state observation s_m^o . \square

The size of P' depends on the arity of the predicates in Ψ , that shape variables F , and the number of parameters of the action models, $|\text{pars}(\xi)|$. The larger these arities, the larger $|\mathcal{I}_{\xi,\Psi}|$. The size of $\mathcal{I}_{\xi,\Psi}$ is the most dominant factor of the compilation because it defines the $\text{pre-}\xi\text{-e/eff-}\xi\text{-e}$ fluents, the corresponding set of `insert` actions, and the number of conditional effects in the $(\text{apply})_{\xi,\omega}$ actions. Note that *typing* can be used straightforward to constrain the FOL interpretations of Ψ over the parameters $\text{pars}(\xi)$, which will significantly reduce $|\mathcal{I}_{\xi,\Psi}|$ and hence the size of P' output by the compilation.

Classical planners tend to prefer shorter solution plans, so our compilation may introduce a bias to $\Lambda = \langle \mathcal{M}, \mathcal{O}, \Phi \rangle$ learning tasks preferring solutions that are referred to action models with a shorter number of preconditions/effects. In more detail, all $\{\text{pre-}\xi\text{-e, eff-}\xi\text{-e}\}_{\forall e \in \mathcal{I}_{\xi,\Psi}}$ fluents are false at the initial state of our P' compilation so classical planners tend to solve P' with plans that require a smaller number of `insert` actions.

This bias can be eliminated defining a cost function for the actions in P' (e.g. `insert` actions have *zero cost* while $(\text{apply})_{\xi,\omega}$ actions have a *positive constant cost*). In practice we use a different approach to disregard the cost of `insert` actions since classical planners are not proficient at optimizing plan cost with zero-cost actions. Instead, our approach is to use a SAT-based planner [Rintanen, 2014] that can apply all actions for inserting preconditions in a single planning step (these actions do not interact). Further, the actions for inserting action effects are also applied in another single planning step. The plan horizon for programming any action model is then always bounded to 2. The SAT-based planning approach is also convenient for its ability to deal with planning problems populated with dead-ends and because symmetries in the insertion of preconditions/effects into an action model do not affect the planning performance.

5 Evaluation

This section evaluates the performance of our approach for learning STRIPS action models with different amounts of available input knowledge.

Reproducibility

The domains used in the evaluation are IPC domains that satisfy the STRIPS requirement [Fox and Long, 2003], taken from the PLANNING.DOMAINS repository [Muise, 2016]. For each domain we generated 10 trajectories of length 10 via random walks to be used as training examples through all the experiments. We also introduce a new parameter, the *degree of observability* σ , which indicates the probability of observing a literal in an intermediate state. This parameter is used to build training examples with varying degrees of incompleteness from the generated trajectories. All experiments are run on an Intel Core i5 3.10 GHz x 4 with 16 GB of RAM.

For the sake of reproducibility, the compilation source code, evaluation scripts, used benchmarks and input *state-invariants* are fully available at the repository <https://github.com/anonsub/>.

Metrics

The learned models are evaluated using the *precision* and *recall* metrics for action models proposed in [Aineto et al., 2018], which compare the learned models against the reference model.

Precision measures the correctness of the learned models. Formally, $\text{Precision} = \frac{tp}{tp+fp}$, where tp is the number of true positives (predicates that appear in both the learned and reference action models) and fp is the number of false positives (predicates that appear in the learned action model but not in the reference model). Recall, on the other hand, measures the completeness of the model and is formally defined as $\text{Recall} = \frac{tp}{tp+fn}$ where fn is the number of false negatives (predicates that should appear in the learned action model but are missing).

5.1 Observability versus Knowledge

In our first experiment we seek to answer the question of whether domain knowledge can substitute observations. To this end, we evaluate the following 4 settings:

- **Neither observability nor knowledge:** This is the baseline setting where the input sample is reduced to the minimum and only the initial and final states are known ($\sigma = 0$).
- **Only knowledge:** Here we add domain knowledge encoded as schematic mutexes to the baseline scenario.
- **Only observability:** In this one, we use a more complete observation where part of the intermediate states is known ($\sigma = 0.2$).
- **Both observability and knowledge:** In the last setting we use we use both more complete input examples and schematic mutexes.

Table 1 compiles the average precision (P) and recall (R) for each domain in the different settings tested. The table also reports the number of schematic mutexes ($|\Phi|$) used for

	Φ	$\sigma = 0$		$\sigma = 0$ with Φ		$\sigma = 0.2$		$\sigma = 0.2$ with Φ	
		P	R	P	R	P	R	P	R
blocks	9	0.52	0.38	0.53	0.21	0.66	0.56	0.77	0.68
driverlog	8	0.49	0.33	0.33	0.31	0.54	0.38	0.70	0.53
ferry	2	0.50	0.40	0.57	0.41	0.59	0.64	0.59	0.70
floor-tile	7	0.30	0.40	0.58	0.46	0.68	0.46	0.75	0.48
grid	3	0.47	0.40	0.47	0.37	0.43	0.34	0.43	0.32
gripper	5	0.77	0.56	0.77	0.54	0.85	0.74	0.96	0.83
hanoi	3	0.84	0.76	0.76	0.75	0.96	0.75	0.97	0.79
n-puzzle	3	0.94	0.86	0.93	0.86	0.99	0.87	1.00	0.87
parking	8	0.48	0.37	0.60	0.40	0.58	0.45	0.67	0.49
transport	4	0.45	0.45	0.53	0.46	0.99	0.51	0.94	0.79
zeno-travel	4	0.72	0.39	0.75	0.40	0.80	0.42	0.93	0.55
		0.59	0.48	0.62	0.47	0.73	0.56	0.79	0.64

Table 1: Observability versus knowledge

each domain. Comparing the settings with only knowledge and only observability (second and third columns), it is clear that having more complete training examples is preferable. In fact, the improvement of using domain knowledge is marginal with respect to the baseline case. This is not the case for the last setting, where the use of domain knowledge shows a significant improvement in the quality of the learned models when compared to the setting with only observability.

5.2 Using knowledge to counter incompleteness

In the previous experiment we have established that domain knowledge is no substitute to observations, and that, given a minimum of observability, domain knowledge is able to enrich both the observations and learning process thus procuring better learned models. In this next experiment we are going to measure the improvement provided by domain knowledge at increasing degrees of observability.

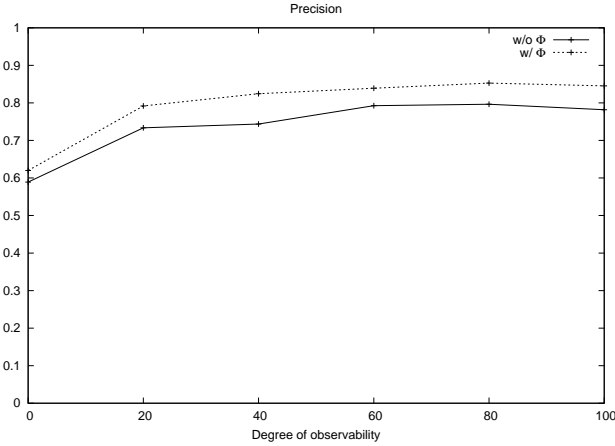


Figure 5: Comparison of the precision of the learned models for increasing degrees of observability.

Figures 5 and 6 compare the precision and recall of the learned models in the settings with and without domain knowledge. The values plotted in these figures are averages across all the domains seen in the previous experiment. The results show that using domain knowledge significantly improves the learned models no matter how complete the training examples are. Another interesting aspect is that domain knowledge is able to enrich observations in the range of 30% observability to the level of fully observable trajectories,

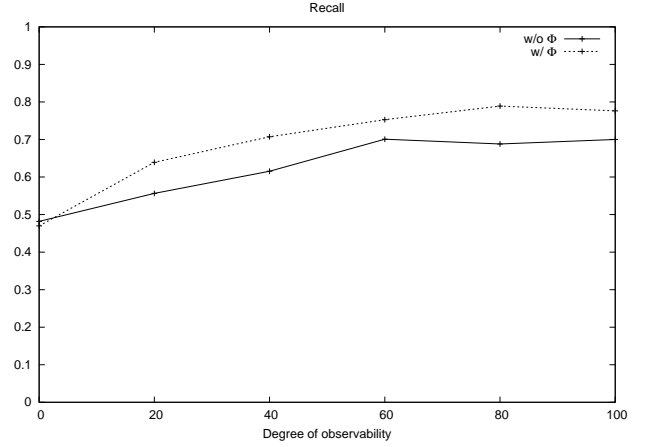


Figure 6: Comparison of the recall of the learned models for increasing degrees of observability.

which means that domain knowledge can make up for a lack of completeness in the training examples.

6 Conclusions

Our learning setting is related to the classical planning formulation where no action model is given [Stern and Juba, 2017]. This planning setting can be seen as a scenario when the action model is *learned* from a single example that contains only two state observations: the initial state and the goals.

In some contexts it is however reasonable to assume that the action model is not learned from scratch, e.g. because some parts of the action model are known [Zhuo *et al.*, 2013; Sreedharan *et al.*, 2018; Pereira and Meneguzzi, 2018]. Our compilation is also flexible to this particular learning scenario. The known preconditions and effects are encoded setting the corresponding fluents $\{pre_e_x, eff_e_x\}_{e \in \mathcal{I}_\xi, \Psi}$ to true in the initial state. Further, the corresponding insert actions, $insertPre_{p,\xi}$ and $insertEff_{p,\xi}$, become unnecessary and are removed from A_Λ , making the classical planning task P_Λ easier to be solved. For example, suppose that the preconditions of the *blocksworld* action schema *stack* are known, then the initial state is extended with literals, $(pre_holding_v1_stack)$ and $(pre_clear_v2_stack)$ and the associated actions $insertPre_{holding_v1_stack}$ and $insertPre_{clear_v2_stack}$ can be safely removed from the A_Λ action set without altering the *soundness* and *completeness* of the P_Λ compilation.

References

- [Aineto *et al.*, 2018] Diego Aineto, Sergio Jiménez, and Eva Onaindia. Learning STRIPS action models with classical planning. In *International Conference on Automated Planning and Scheduling, (ICAPS-18)*, pages 399–407, 2018.
- [Alcázar and Torralba, 2015] Vidal Alcázar and Alvaro Torralba. A reminder about the importance of computing and exploiting invariants in planning. In *ICAPS*, pages 2–6. AAAI Press, 2015.

- [Amir and Chang, 2008] Eyal Amir and Allen Chang. Learning partially observable deterministic action models. *Journal of Artificial Intelligence Research*, 33:349–402, 2008.
- [Bernardini *et al.*, 2018] Sara Bernardini, Fabio Fagnani, and David E. Smith. Extracting mutual exclusion invariants from lifted temporal planning domains. *Artificial Intelligence*, 258:1–65, 2018.
- [Fox and Long, 1998] Maria Fox and Derek Long. The automatic inference of state invariants in TIM. *Journal of Artificial Intelligence Research*, 9:367–421, 1998.
- [Fox and Long, 2003] Maria Fox and Derek Long. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research*, 20:61–124, 2003.
- [Haslum *et al.*, 2018] Patrik Haslum, Franc Ivankovic, Miquel Ramírez, Dan Gordon, Sylvie Thiébaux, Vikas Shivashankar, and Dana S. Nau. Extending classical planning with state constraints: Heuristics and search for optimal planning. *Journal of Artificial Intelligence Research*, 62:373–431, 2018.
- [Helmert, 2009] Malte Helmert. Concise finite-domain representations for pddl planning tasks. *Artificial Intelligence*, 173(5-6):503–535, 2009.
- [Kautz and Selman, 1999] Henry Kautz and Bart Selman. Unifying SAT-based and graph-based planning. In *IJCAI*, volume 99, pages 318–325, 1999.
- [Kucera and Barták, 2018] Jirí Kucera and Roman Barták. LOUGA: learning planning operators using genetic algorithms. In *Pacific Rim Knowledge Acquisition Workshop, PKAW-18*, pages 124–138, 2018.
- [Lotinac and Jonsson, 2016] Damir Lotinac and Anders Jonsson. Constructing hierarchical task models using invariance analysis. In *ECAI*, pages 1274–1282, 2016.
- [McDermott *et al.*, 1998] Drew McDermott, Malik Ghallab, Adele Howe, Craig Knoblock, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins. PDDL – The Planning Domain Definition Language, 1998.
- [Mourão *et al.*, 2012] Kira Mourão, Luke S. Zettlemoyer, Ronald P. A. Petrick, and Mark Steedman. Learning STRIPS operators from noisy and incomplete observations. In *Conference on Uncertainty in Artificial Intelligence, UAI-12*, pages 614–623, 2012.
- [Muggleton and De Raedt, 1994] Stephen Muggleton and Luc De Raedt. Inductive logic programming: Theory and methods. *The Journal of Logic Programming*, 19:629–679, 1994.
- [Muise, 2016] Christian Muise. Planning domains. *ICAPS system demonstration*, 2016.
- [Pereira and Meneguzzi, 2018] Ramon Fraga Pereira and Felipe Meneguzzi. Heuristic approaches for goal recognition in incomplete domain models. *arXiv preprint arXiv:1804.05917*, 2018.
- [Rintanen, 2014] Jussi Rintanen. Madagascar: Scalable planning with SAT. In *International Planning Competition, (IPC-2014)*, 2014.
- [Rintanen, 2017] Jussi Rintanen. Schematic invariants by reduction to ground invariants. In *National Conference on Artificial Intelligence, AAAI-17*, pages 3644–3650, 2017.
- [Slaney and Thiébaux, 2001] John Slaney and Sylvie Thiébaux. Blocks world revisited. *Artificial Intelligence*, 125(1-2):119–153, 2001.
- [Sohrabi *et al.*, 2016] Shirin Sohrabi, Anton V. Riabov, and Octavian Udrea. Plan recognition as planning revisited. In *International Joint Conference on Artificial Intelligence, (IJCAI-16)*, pages 3258–3264, 2016.
- [Sreedharan *et al.*, 2018] Sarath Sreedharan, Tathagata Chakraborti, and Subbarao Kambhampati. Handling model uncertainty and multiplicity in explanations via model reconciliation. In *International Conference on Automated Planning and Scheduling, (ICAPS-18)*, pages 518–526, 2018.
- [Stern and Juba, 2017] Roni Stern and Brendan Juba. Efficient, safe, and probably approximately complete learning of action models. In *International Joint Conference on Artificial Intelligence, (IJCAI-17)*, pages 4405–4411, 2017.
- [Yang *et al.*, 2007] Qiang Yang, Kangheng Wu, and Yunfei Jiang. Learning action models from plan examples using weighted MAX-SAT. *Artificial Intelligence*, 171(2-3):107–143, 2007.
- [Zhuo and Kambhampati, 2013] Hankz Hankui Zhuo and Subbarao Kambhampati. Action-model acquisition from noisy plan traces. In *International Joint Conference on Artificial Intelligence, IJCAI-13*, pages 2444–2450, 2013.
- [Zhuo *et al.*, 2013] Hankz Hankui Zhuo, Tuan Anh Nguyen, and Subbarao Kambhampati. Refining incomplete planning domain models through plan traces. In *International Joint Conference on Artificial Intelligence, IJCAI-13*, pages 2451–2458, 2013.
- [Zhuo, 2015] Hankz Hankui Zhuo. Crowdsourced action-model acquisition for planning. In *National Conference on Artificial Intelligence, AAAI-15*, pages 3439–3446, 2015.