Goal Recognition as Planning with Unknown Action Models

Diego Aineto and **Sergio Jiménez** and **Eva Onaindia**
Departamento de Sistemas Informáticos y Computación
Universitat Politècnica de València.
Camino de Vera s/n. 46022 Valencia, Spain
{dieaigar,serjice,onaindia}@dsic.upv.es

Miquel Ramírez
School of Computing and Information Systems
The University of Melbourne
Melbourne, Victoria. Australia
miquel.ramirez@nimelb.edu.au

Abstract

Plan recognition as planning assumes that *observers* must have correct and complete knowledge of the action model of the observed agents. We relax this assumption formulating a novel setup for classical planning where action models are unknown but the state variables and the action parameters are however known. The experimental results demonstrate that this novel classical planning setup allows us to solve standard goal recognition benchmarks, still using an off-the-shelf classical planner, but without knowing beforehand the precise action model of the observed agents.

Introduction

Goal recognition is a particular classification task in which each class represents a different goal of the observed agents and classification examples are observations of the agents pursuing one of those goals. While there exist diverse approaches for *goal recognition*, *plan recognition as planning* (Ramírez and Geffner 2009; 2010) is one of the most popular and it is currently at the core of various model-based activity recognition tasks such as, *goal recognition design* (Keren, Gal, and Karpas 2014), *deceptive planning* (Masters and Sardina 2017), *planning for transparency* (MacNally et al. 2018) or *counterplanning* (Pozanco et al. 2018).

Plan recognition as planning leverages the action model of the observed agents, a single plan observation, and an off-the-shelf classical planner to estimate the most likely goal of the observed agents (Ramírez 2012). In this paper we show how to relax the assumption of *knowing the action model of the observed agents*, which can become a too strong requirement. We formulate a novel set up for classical planning where no action model is given (instead, only the state variables and the action parameters are known beforehand) and we show that this formulation neatly fits into the *plan recognition as planning* approach. The experimental results demonstrate that we can solve standard goal recognition benchmarks, still using an off-the-shelf classical planner, but without requiring having at hand a model of the *preconditions* and *effects* of the actions of the observed agents.

Background

This section formalizes the *classical planning* model that we follow in this work, the kind of *observations* of plan executions that input the *goal recognition* task, and the *plan recognition as planning* approach for the *goal recognition* task.

Classical planning with conditional effects

We denote as F (*fluents*) the set of propositional state variables. A partial assignment of values to fluents is represented by L (*literals*). To implicitly represent the unobserved literals of a state we adopt the *open world assumption* (i.e. what is not known to be true in a state is *unknown*). 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 comprises a set of preconditions $\mathsf{pre}(a) \in \mathcal{L}(F)$ and a set of effects $\mathsf{eff}(a) \in \mathcal{L}(F)$. The semantics of an action a is specified with two functions: $\rho(s, a)$ denoting whether a is *applicable* in a state s and $\theta(s, a)$ denoting the *successor state* that results from applying a in s. Then $\rho(s, a)$ holds, i.e. the action preconditions hold in the given state iff $\mathsf{pre}(a) \subseteq s$. The result of executing an applicable action a in a state s is a new state $\theta(s, a) = \{s \setminus \neg\mathsf{eff}(a) \cup \mathsf{eff}(a)\}$, where $\neg\mathsf{eff}(a)$ is the complement of $\mathsf{eff}(a)$, which is subtracted from s 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 $\mathsf{eff}^+(a) \in \mathsf{eff}(a)$, while $\mathsf{eff}^-(a) \in \mathsf{eff}(a)$ denotes the *negative effects*.

A planning action a can also define a set of *conditional effects* $cond(a)$. Formally a conditional effect $C \rhd E \in cond(a)$ is composed of two sets of literals: the *condition*, $C \in \mathcal{L}(F)$, and the *effect*, $E \in \mathcal{L}(F)$. The *triggered effects* resulting from the action application (conditional effects whose conditions hold in s) is defined as $\mathsf{eff}_c(s, a) = \bigcup_{C \rhd E \in \mathsf{cond}(a), C \subseteq s} E$.

A *planning problem* is a tuple $P = \langle F, A, I, G\rangle$, where A is a set of actions, 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 $\pi = \langle a_1, \ldots, a_n\rangle$, with $|\pi| = n$ denoting its *plan length*. The execution of π in I induces a *trajectory* $\tau = \langle s_0, a_1, s_1, \ldots, a_n, s_n\rangle$ 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 observation model

Given a planning problem $P = \langle F, A, I, G \rangle$, a plan π that solves P, and the corresponding trajectory τ induced by the execution of π in I, $\tau = \langle s_0, a_1, s_1, \ldots, a_n, s_n \rangle$; there exist as many possible observations of τ as combinations of observable actions and observable fluents of the states in τ. We refer to the set of all possible combinations of observable elements of τ as $Obs(\tau)$.

In this work we define *an observation* $\mathcal{O} \in Obs(\tau)$ as a sequence of *partially observed states*, $\mathcal{O} = \langle s_0^o, s_1^o \ldots, s_m^o \rangle$, except for the initial state $s_0^o = I$ which is fully observed. A *partially observed state* is one in which $|s_i^o| < |F|$, $1 \leq i \leq m \leq n$; i.e., a state in which at least the value of a fluent in F is *unknown*. It may be also the case that $|s_i^o| = 0$ when an intermediate state is fully unknown. This observation model allow us to distinguish between *observable state variables*, whose value may be read from sensors, and *hidden* (or *latent*) *state variables*, that cannot be observed.

The observation model can also include *observed actions* as fluents indicating the applied action in a given state. This means that a sequence of observed actions $\langle a_1^o, \ldots, a_l^o \rangle$ is a sub-sequence of $\pi = \langle a_1, \ldots, a_n \rangle$ such that $a_i^o \in s_{i-1}^o$, $0 \leq i \leq l$. Consequently, the number of fluents that represent observed actions, l, can range from 0 (in a fully unobservable action sequence) to $|\pi| = n$ (in a fully observed action sequence).

Given $\mathcal{O} \in Obs(\tau)$, the number of observed states of $\mathcal{O} = \langle s_0^o, s_1^o \ldots, s_m^o \rangle$ ranges from 2 (at least the initial and final state) to $|\pi|+1$. The number of fluents of the full observable state s_0^o will be $|F|$, or $|F|+1$ in case the fluent of the applied action in s_0 is also observed. Every observable intermediate state will comprise a number of fluents between $[1, |F|+1]$, where a single fluent may represent a sensing fluent of the state or the observation of the applied action.

Goal recognition as planning

Goal recognition is a specific classification task in which each class represents a different possible goal $G \in G[\cdot]$ (where $G[\cdot]$ is the set of *recognizable* goals) and there is a single classification example $\mathcal{O}$ (that represents the observation of a plan execution where agents act to achieve a goal $G \in G[\cdot]$).

According to the *naive Bayes classifier*, the *solution* to the *goal recognition* task is the subset of goals in $G[\cdot]$ that maximizes this expression.

$$argmax_{G \in G[\cdot]} P(\mathcal{O}|G)P(G). \quad (1)$$

The *plan recognition as planning* approach for goal recognition shows that the $P(\mathcal{O}|G)$ likelihood can be estimated leveraging the action model of the observed agents and off-the-shelf classical planners (Ramírez 2012). Given $P = \langle F, A, I, G[\cdot] \rangle$ then $P(\mathcal{O}|G)$ is estimated computing, for each goal $G \in G[\cdot]$, the cost difference of the solution plans to two classical planning problems:

- $P_G^\top$, the classical planning problem built constraining $P = \langle F, A, I, G \rangle$ to achieve $G \in G[\cdot]$ through a plan $\pi^\top$ *consistent* with the input observation $\mathcal{O}$.
- $P_G^\perp$, the classical planning problem that constrains solutions of $P = \langle F, A, I, G \rangle$ to plans $\pi^\perp$, that achieve $G \in G[\cdot]$, but that are *inconsistent* with $\mathcal{O}$.

The higher the value of the $\Delta(\pi^\top, \pi^\perp)$ cost difference, the higher the probability of the observed agents to aim goal $G \in G[\cdot]$. *Plan recognition as planning* uses the *sigmoid function* to map the previous cost difference into a likelihood:

$$P(\mathcal{O}|G) = \frac{1}{1 + e^{-\beta \Delta(\pi^\top, \pi^\perp)}} \quad (2)$$

This expression is derived from the assumption that while the observed agents are not perfectly rational, they are more likely to follow cheaper plans, according to a *Logistic* distribution. The larger the value of β, the more rational the agents, and the less likely that they will follow suboptimal plans. Recent work on *goal recognition* exploit the structure of action *preconditions* and *effects* to compute fast estimates of the $P(\mathcal{O}|G)$ likelihood (Pereira, Oren, and Meneguzzi 2017).

To compute the target probability distribution $P(G|\mathcal{O})$ the $P(\mathcal{O}|G)$ likelihoods can be plugged into the *Bayes rule*. In this case the $P(\mathcal{O})$ probabilities are obtained by normalization (goal probabilities must add up to 1 when summed over all possible goals).

Goal recognition as planning with unknown action models

This section formalizes the goal recognition task addressed in this paper, defines a setup for classical planning with unknown action models, and shows how to leverage this setup to solve goal recognition tasks when the action model of the observed agents is unknown.

Goal recognition with unknown action models

We define the task of *goal recognition with unknown domain models* as a $\langle P, \mathcal{O} \rangle$ pair:

- $P = \langle F, A[\cdot], I, G[\cdot] \rangle$ is a classical planning problem where $G[\cdot]$ is the set of *recognizable* goals and $A[\cdot]$ is a set of actions s.t., for each $a \in A[\cdot]$, the semantics of a is unknown (i.e. the functions ρ and/or θ of a are undefined).
- $\mathcal{O}$ is the observation of the execution of an unknown plan π to reach goal $G \in G[\cdot]$ starting from the given initial state $I = s_0^o$.

The parameters of the actions in $A[\cdot]$ are known. This means that they are either given in P or deducible from $\mathcal{O}$.

The *solution* to the *goal recognition with unknown domain models* task is again the subset of goals in $G[\cdot]$ that maximizes expression (1). With this regard this new task is defined exactly as the original goal recognition task (Ramírez 2012) except that the *preconditions* and *effects* of the input set of *actions* are now unknown.

Planning with unknown domain models

Our approach to estimate the $P(\mathcal{O}|G)$ likelihoods is to compute solution plans to the classical planning problems $P_G^\top$ and $P_G^\perp$, as in the *plan recognition as planning* approach for goal recognition. The only difference of our approach is that we solve $P_G^\top$ while we build a reasonable model for the actions in $A[\cdot]$ leveraging the available input knowledge $\langle P, \mathcal{O}\rangle$. To do so, we formulate *planning with unknown domain models*, a setup for classical planning where no action model is given. The state variables, action parameters and a single plan observation are known though. This setup is closely related to the learning of planning action models (Stern and Juba 2017) since it can be understood as *one-shot learning* (i.e. learning from a single example).

A *classical planning with unknown domain models* is defined as a tuple $P = \langle F, A[\cdot], I, G\rangle$, where $A[\cdot]$ is a set of actions s.t., the semantics of each action $a \in A[\cdot]$ is unknown (i.e. the functions ρ and/or θ of a are undefined).

A *solution plan* is a sequence of actions $\pi = \langle a_1, \ldots, a_n\rangle$ whose execution on I induces a trajectory $\tau = \langle s_0, a_1, s_1, \ldots, a_n, s_n\rangle$ such that $s_0 = I$ and *there exists* at least one possible action model (e.g. one possible definition of the ρ and θ functions within the given state variables) satisfying the following constraints:

- For every $1 \leq i \leq n$ then $\rho(s_{i-1}, a_i) = \texttt{True}$.
- For every $1 \leq i \leq n$ then $s_i = \theta(s_{i-1}, a_i)$.
- Goals are met at the final state $G \subseteq s_n$.

Further constraints can be defined to build more accurate models for the $\rho(s, a)$ and $\theta(s, a)$ functions of the $A[\cdot]$ actions:

- *Observations*. The states trajectories induced by plans computable with the actions in $A[\cdot]$ must be *consistent* with the observed states in $\mathcal{O}$. Observations $\mathcal{O}$ act as ordered *landmarks* for the planning problem P (Hoffmann, Porteous, and Sebastia 2004) since all the fluents of the sets in $\mathcal{O}$ must be achieved by any plan that solves P and in the same order as defined in the observation $\mathcal{O}$.
- *Domain-specific knowledge*. *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). *State invariants* reduce the space of the possible preconditions and effects of the actions in $A[\cdot]$. For instance, in the *blocksworld* one can argue that `on(?x,?x)` will not appear in the preconditions/effects of an action because, in this specific domain, a block cannot be on top of itself.
- *Partially-specified action models*. In some contexts some portions of the preconditions and effects of the actions in $A[\cdot]$ can be known (Zhuo, Nguyen, and Kambhampati 2013; Sreedharan, Chakraborti, and Kambhampati 2018; Pereira and Meneguzzi 2018).

$\forall x_1, x_2\ \neg ontable(x_1) \vee \neg on(x_1, x_2)$.
$\forall x_1, x_2\ \neg clear(x_1) \vee \neg on(x_2, x_1)$.
$\forall x_1, x_2, x_3\ \neg on(x_1, x_2) \vee \neg on(x_1, x_3)\ such\ that\ x_2 \neq x_3$.
$\forall x_1, x_2, x_3\ \neg on(x_2, x_1) \vee \neg on(x_3, x_1)\ such\ that\ x_2 \neq x_3$.

Figure 1: *Schematic mutexes* for the *blocksworld* domain.

Estimating $P(\mathcal{O}|G)$ by planning with unknown action models

Our formalization of *classical planning with unknown domain models* can be used to estimate the $P(\mathcal{O}|G)$ likelihoods following the *plan recognition as planning* approach for goal recognition:

1. Compute an action model A that can solve $P_G^\top$, now the classical planning problem *with unknown action model* that constrains solutions of $\langle F, A[\cdot], I, G\rangle$ to plans $\pi^\top$ *consistent* with the input observation $\mathcal{O}$, and take $cost(\pi^\top)$ from that solution.
2. Solve $P_G^\perp$, the classical planning problem that constrains $\langle F, A, I, G\rangle$ to achieve $G \in G[\cdot]$ through a plan $\pi^\perp$ *inconsistent* with $\mathcal{O}$, and take $cost(\pi^\perp)$ from that solution.
3. Compute the $\Delta(\pi^\top, \pi^\perp)$ cost difference and plug it into equation (2) to get the $P(\mathcal{O}|G)$ likelihoods.

Completing observations with *schematic mutexes*

The estimate of the $P(\mathcal{O}|G)$ likelihoods can be optimizing by adding a pre-processing step:

0. Complete the input observation $\mathcal{O}$ using the domain-specific knowledge if available.

Next we provide the details os this pre-process. 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 $block_A$ 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, Fagnani, and Smith 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 of predicates in Ψ (the initial state is a full assignment of values to fluents so the predicates Ψ are extractable from I) that satisfy the formulae $\neg p \vee \neg q$, considering that their corresponding parameters are variables universally quantified. For instance, $holding(v_1)$ and $clear(v_1)$ from the *blocksworld*

Encoding	Meaning
$\neg pre_{p,\xi} \wedge \neg eff_{p,\xi}$	p belongs neither to the preconditions nor effects of ξ $(p \notin pre(\xi) \wedge p \notin add(\xi) \wedge p \notin del(\xi))$
$pre_{p,\xi} \wedge \neg eff_{p,\xi}$	p is only a precondition of ξ $(p \in pre(\xi) \wedge p \notin add(\xi) \wedge p \notin del(\xi))$
$\neg pre_{p,\xi} \wedge eff_{p,\xi}$	p is a positive effect of ξ $(p \notin pre(\xi) \wedge p \in add(\xi) \wedge p \notin del(\xi))$
$pre_{p,\xi} \wedge eff_{p,\xi}$	p is a negative effect of ξ $(p \in pre(\xi) \wedge p \notin add(\xi) \wedge p \in del(\xi))$

Figure 2: Combinations of the propositional encoding and their meaning

are *schematic mutex* while $clear(v_1)$ and $ontable(v_1)$ are not because $\forall v_1, \neg clear(v_1) \vee \neg ontable(v_1)$ does not hold for every possible state. Figure 1 shows an example of four clauses that define schematic mutexes for the *blocksworld* domain.

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) $\neg q(\omega') \in s_j^o$, in which case the expression $\neg p(\omega) \vee \neg q(\omega')$ holds in s_j^o; or (b) $\neg q(\omega') \notin s_j^o$, in which case the literal 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 mutex $\neg holding(v_1) \vee \neg clear(v_1)$, we extend the state observation with literal `¬clear(blockA)` (despite this particular literal being initially unknown).

Classical planning with unknown STRIPS models

This section shows that, when the unknown precondition and effects of the actions in $A[\cdot]$ follow the STRIPS model, we can solve the task of *planning with unknown domain models* (and hence estimate the $P(\mathcal{O}|G)$ likelyhood) with an off-the-shelf classical planner.

A propositional encoding for STRIPS actions models

A STRIPS *action model* is defined as $\xi = \langle name(\xi), pars(\xi), pre(\xi), add(\xi), del(\xi)\rangle$, where $name(\xi)$ and parameters, $pars(\xi)$, define the header of ξ; and $pre(\xi)$, $del(\xi)$ and $add(\xi)$) are sets of fluents that represent the *preconditions*, *negative effects* and *positive effects*, respectively, of the actions induced from the action model ξ.

```
(:action stack
   :parameters (?v1 ?v2)
   :precondition (and (holding ?v1) (clear ?v2))
    :effect (and (not (holding ?v1)) (not (clear ?v2))
                 (clear ?v1) (handempty) (on ?v1 ?v2)))


(pre_holding_v1_stack) (pre_clear_v2_stack)
(eff_holding_v1_stack) (eff_clear_v2_stack)
(eff_clear_v1_stack) (eff_handempty_stack) (eff_on_v1_v2_stack)
```

Figure 3: PDDL encoding of the `stack(?v1,?v2)` schema and our propositional representation for this same schema.

Let Ψ be the set of *predicates* that shape the fluents F. The set of propositions that can appear in $pre(\xi)$, $del(\xi)$ and $add(\xi)$ of a given ξ, denoted as $\mathcal{I}_{\xi,\Psi}$, are FOL interpretations of Ψ over the parameters $pars(\xi)$.

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, $\mathcal{I}_{pickup,\Psi}$={`handempty`, `holding`$(v_1)$`,clear`$(v_1)$`,ontable`$(v_1)$`, on`$(v_1,v_1)$} and eleven elements for the model of `stack`(v_1,v_2), $\mathcal{I}_{stack,\Psi}$={`handempty, holding`(v_1)`, holding`(v_2)`, clear`(v_1)`,clear`(v_2)`,ontable`(v_1)`,ontable`(v_2)`, on`(v_1,v_1)`,on`(v_1,v_2)`, on`(v_2,v_1)`, on`(v_2,v_2)}.

An action model ξ 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* are also a type of syntactic constraint that reduce the size of $\mathcal{I}_{\xi,\Psi}$ (McDermott et al. 1998). Considering only these 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. Given $p \in \mathcal{I}_{\Psi,\xi}$, the belonging of p to the preconditions, positive effects or negative effects of ξ is handled with a propositional encoding that uses fluents of two types, $pre_{p,\xi}$ and $eff_{p,\xi}$. The four possible combinations of these two fluents are summarized in Figure 2.

To illustrate better this encoding, Figure 3 shows the PDDL encoding of the `stack(?v1,?v2)` schema and our propositional representation for this same schema with $pre_{p,stack}$ and $eff_{p,stack}$ fluents ($p \in \mathcal{I}_{\Psi,stack}$).

A classical planning compilation for planning with unknown domain models

The approach for learning STRIPS action models presented in (Aineto, Jiménez, and Onaindia 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 $\xi = \langle name(\xi), pars(\xi), pre(\xi), add(\xi), del(\xi)\rangle$, where $name(\xi)$ and parameters, $pars(\xi)$, define the header of ξ; and $pre(\xi)$, $del(\xi)$ and $add(\xi)$) 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*[1] that shape the variables F. The set of propositions of P' that can appear in $pre(\xi)$, $del(\xi)$ and $add(\xi)$ of a given ξ, denoted as $\mathcal{I}_{\xi,\Psi}$, are FOL interpretations of Ψ over the parameters $pars(\xi)$. 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, $\mathcal{I}_{pickup,\Psi}$={`handempty`, `holding(`$v_1$`)`,`clear(`$v_1$`)`,`ontable(`$v_1$`)`, `on(`$v_1,v_1$`)`} and eleven elements for the model of `stack(`v_1,v_2`)`, $\mathcal{I}_{stack,\Psi}$={`handempty`, `holding(`v_1`)`, `holding(`v_2`)`, `clear(`v_1`)`,`clear(`v_2`)`,`ontable(`v_1`)`,`ontable(`v_2`)`, `on(`v_1,v_1`)`,`on(`v_1,v_2`)`, `on(`v_2,v_1`)`, `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 each action model ξ.

The decision as to whether or not an element of $\mathcal{I}_{\xi,\Psi}$ will be part of $pre(\xi)$, $del(\xi)$ or $add(\xi)$ 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)`,`(validate_1)`,`(apply_putdown blockB)`,`(validate_2)`, where the `validate` 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.

Observations

This require the compilation to include actions for *validating* partially observed states $s_j^o \in \mathcal{O}$. These actions are also part of the postfix of the solution plan π_Λ and they are aimed at checking that the observation $\mathcal{O}$ follows after the execution of the apply actions.

$$\begin{aligned}\mathsf{pre}(\mathsf{validate_j}) =& s_j^o \cup \{test_{j-1}\},\\ \mathsf{cond}(\mathsf{validate_j}) =& \{\emptyset\} \rhd \{\neg test_{j-1}, test_j\}.\end{aligned}$$

[1]The 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.

ID	Action	New conditional effect
1	$(\mathtt{insert\_pre})_{\xi,\mathrm{p}}$	$\{pre\_\xi\_q\} \rhd \{invalid\}$
2	$(\mathtt{insert\_eff})_{\xi,\mathrm{p}}$	$\{pre\_\xi\_q \wedge eff\_\xi\_q \wedge pre\_\xi\_p\} \rhd \{invalid\}$
3	$(\mathtt{insert\_eff})_{\xi,\mathrm{p}}$	$\{\neg pre\_\xi\_q \wedge eff\_\xi\_q \wedge \neg pre\_\xi\_p\} \rhd \{invalid\}$
4	$(\mathtt{apply})_{\xi,\omega}$	$\{\neg pre\_\xi\_p \wedge eff\_\xi\_p \wedge$ $q(\omega) \wedge \neg pre\_\xi\_q\} \rhd \{invalid\}$
5	$(\mathtt{apply})_{\xi,\omega}$	$\{\neg pre\_\xi\_p \wedge eff\_\xi\_p \wedge$ $q(\omega) \wedge \neg eff\_\xi\_q\} \rhd \{invalid\}$

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

There will be a validate action in π_Λ for every observed state in $\mathcal{O}$. The position of the validate actions in π_Λ will be determined by the planner by checking that the state resulting after the execution of an apply action comprises the observed state $s_j^o \in \mathcal{O}$.

Schematic mutexes

We could extend the classical planning compilation for the learning of STRIPS action models (Aineto, Jiménez, and Onaindia 2018) to check the consistency of the *state-constraints* in Φ at every state traversed by a solution to the compiled problem. Unfortunately, checking arbitrary ϕ formulae is too expensive for current classical planners.

Instead, our approach is to define a mechanism to check *state-constraints* in the form of *schematic mutex*. To implement this checking mechanism we add new conditional effects to the *insert* and *apply* actions of the classical planning compilation. Figure 4 summarizes the new conditional effects added to the compilation and next, we describe them in detail:

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 $\mathcal{I}_{\xi,\Psi}$, one conditional effect is added to the $(\mathtt{insert\_pre})_{\xi,\mathrm{p}}$ actions to prevent the insertion of two preconditions that are schematic mutex. Likewise, two conditional effects are added to the $(\mathtt{insert\_eff})_{\xi,\mathrm{p}}$ 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 $\mathcal{I}_{\xi,\Psi}$, two conditional effects are added to the $(\mathtt{apply})_{\xi,\omega}$ actions to prevent positive effects that are inconsistent with an input observation (in $(\mathtt{apply})_{\xi,\omega}$ actions the variables in $pars(\xi)$ 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).

The goals of the planning task P' generated by the original BLS compilation are extended with the $\neg invalid$ literal to validate that only states consistent with the state constraints defined in Φ are traversed by solution plans. Remarkably, the $\neg invalid$ literal allows us also to define $(\mathtt{apply})_{\xi,\omega}$ actions more compactly than in the original compilation. Disjunctions are no longer required to code the possible preconditions of an action schema since they can now be encoded with conditional effects of the type $\{pre\_\xi\_p \wedge \neg p(\omega)\} \rhd \{invalid\}$.

Partially specified action models

Our compilation approach is also flexible to this particular scenario. The known preconditions and effects are encoded setting the corresponding fluents $\{pre\_e\_\xi, eff\_e\_\xi\}_{\forall e \in \mathcal{I}_{\Psi,\xi}}$ to true in the initial state. Further, the corresponding insert actions, $\mathsf{insertPre}_{\mathsf{p},\xi}$ and $\mathsf{insertEff}_{\mathsf{p},\xi}$, 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 I is extended with literals, `(pre_holding_v1_stack)` and `(pre_clear_v2_stack)` and the associated actions $\mathsf{insertPre}_{\mathsf{holding_v1,stack}}$ and $\mathsf{insertPre}_{\mathsf{clear_v2,stack}}$ can be safely removed from the A_Λ action set without altering the *soundness* and *completeness* of the P_Λ compilation.

Evaluation

Related Work

The problem of *classical planning with unknown domain models* has been previously addressed (Stern and Juba 2017). In this work we evidence the relevance of this task for addressing *goal recognition* when the action model of the observed agent is not available.

The paper also showed that *goal recognition*, when the domain model is unknown, is closely related to the learning of planning action models. With this regard, the classical planning compilation for learning STRIPS action models (Aineto, Jiménez, and Onaindia 2018) is very appealing because it allows to produce a STRIPS action model from minimal input knowledge (a single initial state and goals pair), and to refine this model if more input knowledge is available (e.g. observation constraints). Most of the existing approaches for learning action models aim maximizing an statistical consistency of the learned model with respect to the input observations so require large amounts of input knowledge and do not produce action models that are guaranteed to be *logically consistent* with the given input knowledge.

Our approach for *planning with an unknown domain model* is related to *goal recognition design* (Keren, Gal, and Karpas 2014). The reason is that we are encoding the space of propositional schemes as state variables of the planning problem (the initial state encodes the *empty* action model with no preconditions and no effects) and provide actions to modify the value of this state variables as in *goal recognition design*. The aims of *goal recognition design* are however different. *Goal recognition design* applied to *goal recognition with unknown domain models* would compute the action model, in the space of possible models, that allows to reveal any of the possible goals as early as possible.

Conclusions

Classical planners tend to preffer shorter solution plans, so our compilation may introduce a bias to $P = \langle F, A[\cdot], I, G\rangle$ problems preferring solutions that are referred to action models with a shorter number of *preconditions/effects*. In more detail, all $\{pre\_e\_\xi, eff\_e\_\xi\}_{\forall e \in \mathcal{I}_{\Psi,\xi}}$ fluents are false at the initial state of our $P' = \langle F', A', I, G\rangle$ compilation so classical planners tend to solve P' with plans that require a shorter number of *insert* actions.

This bias could be eliminated defining a cost function for the actions in A' (e.g. *insert* actions has *zero cost* while $\mathsf{apply}_{\xi,\omega}$ actions has a *positive constant cost*). In practice we use a different approach to disregard the cost of *insert* actions because classical planners are not proficiency optimizing *plan cost* with zero-cost actions. Instead, our approach is to use a SAT-based planner (Rintanen 2014) because it 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 a single planning step so the plan horizon for programming any action model is always bound to 2, which significantly reduces the planning horizon.

Our compilation for *planning with unknown domain models* can then be understood as an extension of the SATPLAN approach for classical planning (Kautz, Selman, and others 1992) with two additional initial layers: a first layer for inserting the action preconditions and a second one for inserting the action effects. These two extra layers are followed by the typical N layers of the SATPLAN encoding (extended however to apply the action models that are determined by the previous two initial layers, the $\mathsf{apply}_{\xi,\omega}$ actions). Regarding again the example of Figure **??**, this means that steps [00-04] are applied in paralel in the first SATPLAN layer, steps [05-13] are applied in paralel in the second layer and each step [14-17] is applied sequentially and correponds to a differerent SATPLAN layer (so just six layers are necesary to compute the example plan of Figure **??**).

The SAT-based planning approach is also convenient for the task of *goal recognition as planning with unknown domain models* because its ability to deal with classical planning problems populated with dead-ends and because symmetries in the insertion of preconditions/effects into an action model do not affect to the planning performance.

Acknowledgments

This work is supported by the Spanish MINECO project TIN2017-88476-C2-1-R. D. Aineto is partially supported by the *FPU16/03184* and S. Jiménez by the *RYC15/18009*. M. Ramírez research is partially funded by DST Group Joint & Operations Analysis Division.

References

Aineto, D.; Jiménez, S.; and Onaindia, E. 2018. Learning STRIPS action models with classical planning. In *International Conference on Automated Planning and Scheduling, (ICAPS-18)*, 399–407.

Alcázar, V., and Torralba, A. 2015. A reminder about the importance of computing and exploiting invariants in planning. In *ICAPS*, 2–6. AAAI Press.

Bernardini, S.; Fagnani, F.; and Smith, D. E. 2018. Extracting mutual exclusion invariants from lifted temporal planning domains. *Artificial Intelligence* 258:1–65.

Fox, M., and Long, D. 1998. The automatic inference of state invariants in TIM. *Journal of Artificial Intelligence Research* 9:367–421.

Helmert, M. 2009. Concise finite-domain representations for pddl planning tasks. *Artificial Intelligence* 173(5-6):503–535.

Hoffmann, J.; Porteous, J.; and Sebastia, L. 2004. Ordered landmarks in planning. *Journal of Artificial Intelligence Research* 22:215–278.

Kautz, H., and Selman, B. 1999. Unifying SAT-based and graph-based planning. In *IJCAI*, volume 99, 318–325.

Kautz, H. A.; Selman, B.; et al. 1992. Planning as satisfiability. In *ECAI*, volume 92, 359–363. Citeseer.

Keren, S.; Gal, A.; and Karpas, E. 2014. Goal recognition design. In *International Conference on Automated Planning and Scheduling, (ICAPS-14)*, 154–162.

MacNally, A. M.; Lipovetzky, N.; Ramirez, M.; and Pearce, A. R. 2018. Action selection for transparent planning. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, 1327–1335. International Foundation for Autonomous Agents and Multiagent Systems.

Masters, P., and Sardina, S. 2017. Deceptive path-planning. In *IJCAI 2017*, 4368–4375. AAAI Press.

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

Pereira, R. F., and Meneguzzi, F. 2018. Heuristic approaches for goal recognition in incomplete domain models. *arXiv preprint arXiv:1804.05917*.

Pereira, R. F.; Oren, N.; and Meneguzzi, F. 2017. Landmark-based heuristics for goal recognition. In *Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17)*. AAAI Press.

Pozanco, A.; E.-Martín, Y.; Fernández, S.; and Borrajo, D. 2018. Counterplanning using goal recognition and landmarks. In *International Joint Conference on Artificial Intelligence, (IJCAI-18)*, 4808–4814.

Ramírez, M., and Geffner, H. 2009. Plan recognition as planning. In *International Joint conference on Artifical Intelligence, (IJCAI-09)*, 1778–1783. AAAI Press.

Ramírez, M., and Geffner, H. 2010. Probabilistic plan recognition using off-the-shelf classical planners. In *Twenty-Fourth AAAI Conference on Artificial Intelligence*.

Ramírez, M. 2012. *Plan recognition as planning*. Ph.D. Dissertation, Universitat Pompeu Fabra.

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

Rintanen, J. 2017. Schematic invariants by reduction to ground invariants. In *National Conference on Artificial Intelligence, AAAI-17*, 3644–3650.

Slaney, J., and Thiébaux, S. 2001. Blocks world revisited. *Artificial Intelligence* 125(1-2):119–153.

Sreedharan, S.; Chakraborti, T.; and Kambhampati, S. 2018. Handling model uncertainty and multiplicity in explanations via model reconciliation. In *International Conference on Automated Planning and Scheduling, (ICAPS-18)*, 518–526.

Stern, R., and Juba, B. 2017. Efficient, safe, and probably approximately complete learning of action models. In *International Joint Conference on Artificial Intelligence, (IJCAI-17)*, 4405–4411.

Zhuo, H. H.; Nguyen, T. A.; and Kambhampati, S. 2013. Refining incomplete planning domain models through plan traces. In *International Joint Conference on Artificial Intelligence, IJCAI-13*, 2451–2458.