# Goal Recognition as Planning with Unknown Domain 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

This paper shows how to relax a strong assumption of the *plan recognition as planning* approach that is *the observer knows the action model of the observed agents*. The paper introduces a novel formulation for classical planning in a setting where no action model is given (instead, only the state variables and the action parameters are known) and it shows that this formulation neatly fits with the *plan recognition as planning* approach for *goal recognition*. The experimental results demonstrate that this novel formulation for classical planning allow us to solve standard goal recognition benchmarks, still using an off-the-shelf classical planner, but without knowing beforehand the action model of the observed agents.

## Introduction

*Goal recognition* is a particular classification task in which each class represents a different goal and classification examples are observations of agents pursuing one of those goals. While there exist diverse approaches for *goal recognition*, *plan recognition as planning* (**?**; **?**) 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* (**?**), *deceptive planning* (**?**), *planning for transparency* (**?**) or *counter-planning* (**?**).

*Plan recognition as planning* leverages the action model of the observed agents and an off-the-shelf classical planner to estimate the most likely goal of the agents being observed (**?**). 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 when applying *plan recognition as planning* on real-world problems. We introduce a novel formulation for classical planning in a setting 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 this novel formulation allows to 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* that input the *goal recognition* task, and the *plan recognition as planning* approach for *goal recognition*.

### Classical planning with conditional effects

Let $F$ be the set of propositional state variables (*fluents*) describing a state. A *literal* $l$ is a valuation of a fluent $f \in F$; i.e. either $l = f$ or $l = \neg f$. A set of literals $L$ represents a partial assignment of values to fluents (without loss of generality, we will assume that $L$ does not contain conflicting values). Given $L$, let $\neg L = \{\neg l : l \in L\}$ be its complement. 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 *state* $s$ is a full assignment of values to fluents; $|s| = |F|$.

A *classical planning action* $a \in A$ has: a precondition $\mathsf{pre}(a) \in \mathcal{L}(F)$, a set of effects $\mathsf{eff}(a) \in \mathcal{L}(F)$, and a positive action cost $cost(a)$. The semantics of actions $a \in A$ is specified with two functions: $\rho(s, a)$ denotes whether action $a$ is *applicable* in a state $s$ and $\theta(s, a)$ denotes the *successor state* that results of applying action $a$ in a state $s$. Then, $\rho(s, a) = \texttt{True}$ whenever $\mathsf{pre}(a) \subseteq s$ (i.e. if its precondition holds in $s$) otherwise $\rho(s, a) = \texttt{False}$. The result of executing an applicable action $a \in A$ in a state $s$ is a new state $\theta(s, a) = (s \setminus \neg\mathsf{eff}(a)) \cup \mathsf{eff}(a)$. Subtracting the complement of $\mathsf{eff}(a)$ from $s$ ensures that $\theta(s, a)$ remains a well-defined state. The subset of action effects that assign a positive value to a state 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* of an action $a \in A$.

A *classical planning problem* is a tuple $P = \langle F, A, I, G\rangle$, where $I$ is the initial state and $G \subseteq \mathcal{L}(F)$ is the set of goal conditions over the state variables. A *plan* $\pi$ is an action sequence $\pi = \langle a_1, \ldots, a_n\rangle$, with $|\pi| = n$ being the *length* of $\pi$ and $cost(\pi) = \sum_{a\in\pi} cost(a)$ being its *cost*. The execution of $\pi$ on the initial state of $P$ induces a *trajectory* $\tau(\pi, P) = \langle s_0, a_1, s_1, \ldots, a_n, s_n\rangle$ such that $s_0 = I$ and, for each $1 \leq i \leq n$, then $\rho(s_{i-1}, a_i) = \texttt{True}$ and $s_i = \theta(s_{i-1}, a_i)$. A plan $\pi$ solves $P$ iff the induced *trajectory* $\tau(\pi, P)$ reaches a final state $G \subseteq s_n$, where all goal conditions are met. A solution plan is *optimal* iff its cost is minimal.

We also define *actions with conditional effects* because they are useful to compactly formulate our approach for *goal recognition with unknown domain models*. An action $a_c \in A$ with conditional effects is a set of preconditions $\mathsf{pre}(a_c) \in \mathcal{L}(F)$ and a set of *conditional effects* $\mathsf{cond}(a_c)$. Each conditional effect $C \rhd E \in \mathsf{cond}(a_c)$ is composed of two sets of literals: $C \in \mathcal{L}(F)$, the *condition*, and $E \in \mathcal{L}(F)$, the *effect*. An action $a_c$ is applicable in a state $s$ if $\rho(s, a_c)$ is true, and the result of applying action $a_c$ in state $s$ is $\theta(s, a_c) = \{s \setminus \neg\mathsf{eff}_c(s, a) \cup \mathsf{eff}_c(s, a)\}$ where $\mathsf{eff}_c(s, a)$ are the *triggered effects* resulting from the action application (conditional effects whose conditions hold in $s$):

$$\mathsf{eff}_c(s, a) = \bigcup_{C \rhd E \in \mathsf{cond}(a_c), C \subseteq s} E,$$

## The observation model

We assume a bijective monotone mapping between actions/states of trajectories and observations (**?**), thus also granting the inverse consistency relationship (the trajectory is a superset of the observation). This means that transiting between two consecutive observed states may require the execution of more than a single action. In other words, having an input observation does not imply knowing the actual length of the corresponding plan.

Given a planning problem $P = \langle F, A, I, G\rangle$, a plan $\pi$ and a trajectory $\tau(\pi, P)$, we define an interleaved combination of actions and states that represents the *observation from the execution of $\pi$ in $P$*. Formally, $\mathcal{O}(\tau) = \langle s^o_0, a^o_1, s^o_1 \ldots, a^o_l, s^o_m\rangle$, $s^o_0 = I$, and:

- The **observed actions** are consistent with $\pi$, which means that $\langle a^o_1, \ldots, a^o_l\rangle$ is a sub-sequence of $\pi$. The number of observed actions, $l$, ranges from 0 (fully unobserved action sequence) to $|\pi|$ (fully observed action sequence).
- The **observed states** $\langle s^o_0, s^o_1, \ldots, s^o_m\rangle$ is a sequence of possibly *partially observed states*, except for the initial state $s^o_0$, which is fully observed. A partially observable state $s^o_i$ is one in which $|s^o_i| < |F|$; i.e., a state in which at least a fluent of $F$ is not observable. Note that this definition also comprises the case $|s^o_i| = 0$, when the state is fully unobservable. Whatever the sequence of observed states of $\mathcal{O}(\tau)$ is, it must be consistent with the sequence of states of $\tau(\pi, P)$, meaning that $\forall i, s^o_i \subseteq s_i$. The number of observed states, $m$, range from 1 (the initial state, at least), to $|\pi| + 1$, and each *observed* states comprises $[1, |F|]$ fluents ($\mathcal{O}(\tau)$ can still miss intermediate states that are *unobserved*).

## Goal recognition with classical planning

*Goal recognition* is a specific classification task in which each class represents a different possible goal $G \in G[\cdot]$ and there is a single classification example, $\mathcal{O}(\tau)$, that represents an observation of agents acting to achieve a goal $G \in G[\cdot]$.

Following the paradigm of 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 shows that the $P(\mathcal{O}|G)$ likelihood can be estimated leveraging the action model of the observed agents and an off-the-shelf classical planner (**?**). Given a *classical planning problem* $P = \langle F, A, I, G[\cdot]\rangle$ (where $G[\cdot]$ represents the set of *recognizable* goals) then $P(\mathcal{O}|G)$ is estimated computing, for each goal $G \in G[\cdot]$, the cost difference of the solution plans to these two classical planning problems:

- $P^{\top}_G$, the classical planning problem built constraining $P = \langle F, A, I, G\rangle$ to achieve the particular goal $G \in G[\cdot]$ through a plan $\pi^{\top}$ that is *consistent* with the input observation $\mathcal{O}(\tau)$.
- $P^{\perp}_G$, 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 $\mathcal{O}(\tau)$.

The higher the value of the $\Delta(\pi^{\top}, \pi^{\perp})$ cost difference, the higher the probability of the observed agents to aim goal $G \in G[\cdot]$. With this regard, *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 $\beta$, the more rational the agents, and the less likely that they will follow suboptimal plans. Recent work exploits the structure of action *preconditions* and *effects* to compute fast estimates of the $P(\mathcal{O}|G)$ likelihood (**?**).

# Planning with unknown domain models

This section introduces a novel formulation for classical planning in a setting where no action model is given. This setting has already shown related to the learning of action models for planning (**?**) and it can be seen as an extreme scenario when the action model is *learned* from a single example that contains only two state observations: the initial state and the goals.

A *classical planning with unknown domain models* is 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 $\rho$ and/or $\theta$ of $a$ are undefined). A *solution plan* is a sequence of actions $\pi = \langle a_1, \ldots, a_n\rangle$ whose execution induces a trajectory $\tau(\pi, I) = \langle s_0, a_1, s_1, \ldots, a_n, s_n\rangle$ such that $s_0 = I$ and *there exists* at least one possible action model (e.g. one possible definition of the $\rho$ and $\theta$ functions within the given state variables) that satisfies $\rho(s_{i-1}, a_i) = \texttt{True}$ and $s_i = \theta(s_{i-1}, a_i)$, for every $1 \leq i \leq n$, and such that the reached final state $s_n$ meets the goal condition $G \subseteq s_n$.

Next we show that a propositional encoding for the space of STRIPS action models allows us to solve $P = \langle F, A, I, G[\cdot]\rangle$ problems with off-the-shelf classical planners.

## A propositional encoding for the space of STRIPS action models

*A* STRIPS *action schema* $\xi$ is defined by: A list of *parameters* $pars(\xi)$, and three sets of predicates (namely $pre(\xi)$, $del(\xi)$ and $add(\xi)$) that shape the kind of fluents that can appear in the *preconditions*, *negative effects* and *positive effects* of the actions induced from that schema. Let be $\Psi$ the set of *predicates* that shape the propositional state variables $F$, and a list of *parameters*, $pars(\xi)$. The set of elements that can appear in $pre(\xi)$, $del(\xi)$ and $add(\xi)$ of the STRIPS action schema $\xi$ is the set of FOL interpretations of $\Psi$ over the parameters $pars(\xi)$, and is denoted as $\mathcal{I}_{\Psi,\xi}$.

For instance in a four-operator *blocksworld* (**?**), the $\mathcal{I}_{\Psi,\xi}$ set contains only five elements for the case of the `pickup(`$v_1$`)` schemata, $\mathcal{I}_{\Psi,pickup}$={`handempty`, `holding(`$v_1$`)`, `clear(`$v_1$`)`, `ontable(`$v_1$`)`, `on(`$v_1, v_1$`)`} while it contains eleven elements for the `stack(`$v_1$`,`$v_2$`)` schemata, $\mathcal{I}_{\Psi,stack}$={`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(`$v_2, v_2$`)`}.

Despite any element of $\mathcal{I}_{\Psi,\xi}$ can *a priori* appear in the $pre(\xi)$, $del(\xi)$ and $add(\xi)$ of schema $\xi$, in practice the actual space of possible STRIPS schemata is bounded by constraints:

1. **Syntactic constraints**. STRIPS constraints require $del(\xi) \subseteq pre(\xi)$, $del(\xi) \cap add(\xi) = \emptyset$ and $pre(\xi) \cap add(\xi) = \emptyset$. Considering exclusively these syntactic constraints, the size of the space of possible STRIPS schemata is given by $2^{2\times|\mathcal{I}_{\Psi,\xi}|}$. *Typing constraints* are also of this kind (**?**).
2. **Observation constraints**. The observation of the actions and states resulting from the execution of a plan depicts *semantic knowledge* that constraints further the space of possible action schemata.

In this work we introduce a novel propositional encoding of the *preconditions*, *negative*, and *positive* effects of a STRIPS action schema $\xi$ that uses only fluents of two kinds `pre_e_`$\xi$ and `eff_e_`$\xi$ (where $e \in \mathcal{I}_{\Psi,\xi}$). This encoding exploits the syntactic constraints of STRIPS so it is more compact than previous encodings proposed by **? ?** for learning STRIPS actions with classical planning. In more detail, if `pre_e_`$\xi$ holds it means that $e \in \mathcal{I}_{\Psi,\xi}$ is a *precondition* in $\xi$. If `pre_e_`$\xi$ and `eff_e_`$\xi$ holds it means that $e \in \mathcal{I}_{\Psi,\xi}$ is a *negative effect* in $\xi$ while if $pre\_e\_\xi$ does not hold but `eff_e_`$\xi$ holds, it means that $e \in \mathcal{I}_{\Psi,\xi}$ is a *positive effect* in $\xi$. To illustrate this, Figure 1 shows the PDDL encoding of the *blocksworld* `stack(?v1,?v2)` schema and our propositional representation for this schema using only `pre_e_stack` and `eff_e_stack` fluents ($e \in \mathcal{I}_{\Psi,stack}$).

*Domain-specific knowledge* is also helpfull to constrain further the space of possible schemata. For instance, in the *blocksworld* 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 $add(\xi)$ lists of an action schema $\xi$ because, in this specific domain, a block cannot be on top of itself. With this regard, *state invariants* can be exploited either as *syntactic* constraints (to reduce the space of possible action models) but also as *semantic* constraints (to complete partial observations of the states traversed by a plan) (**?**).

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

## A classical planning compilation for planning with unknown domain models

Inspired by the *classical planning compilation for learning* STRIPS *action models* (**?**), we define here a compilation to address the task of *planning with unknown domain models* that uses our compact propositional encoding of STRIPS action models.

Given a classical planning problem with unknown domain models $P = \langle F, A[\cdot], I, G\rangle$ we create a classical planning problem $P' = \langle F', A', I, G\rangle$ such that:

- $F'$ extends $F$ with a fluent $mode_{insert}$, to indicate whether action models are being *programmed* or *applied*, and the $\{pre\_e\_\xi, eff\_e\_\xi\}_{\forall e\in\mathcal{I}_{\Psi,\xi}}$ fluents for the propositional encoding of the corresponding space of STRIPS action models.
- $A'$ replaces the actions in $A$ with two types of actions.
  1. Actions for *inserting* a *precondition*, *positive/negative* effect into the action model $\xi$ following the syntactic constraints of STRIPS .
     - Actions to insert an $e \in \mathcal{I}_{\Psi,\xi}$ *precondition* into $\xi$. The precondition is only inserted when neither $pre\_e\_\xi$ nor $eff\_e\_\xi$ exist in $\xi$.

$$\mathsf{pre}(\mathsf{insertPre}_{\mathsf{p},\xi}) = \{\neg pre\_e\_\xi, \neg eff\_e\_\xi, mode_{insert}\},$$
$$\mathsf{cond}(\mathsf{insertPre}_{\mathsf{p},\xi}) = \{\emptyset\} \rhd \{pre\_e\_\xi\}.$$

     - Actions to insert an $e \in \mathcal{I}_{\Psi,\xi}$ *effect* to the action model $\xi$.

$$\mathsf{pre}(\mathsf{insertEff}_{\mathsf{p},\xi}) = \{\neg eff\_e\_\xi, mode_{insert}\},$$
$$\mathsf{cond}(\mathsf{insertEff}_{\mathsf{p},\xi}) = \{\emptyset\} \rhd \{eff\_e\_\xi\}.$$

  2. Actions for *applying* an action model $\xi$ built by the *insert* actions and bounded to objects $\omega \subseteq \Omega^{|pars(\xi)|}$ (where $\Omega$ is the set of *objects* used to induce the fluents $F$ by assigning objects in $\Omega$ to the $\Psi$ predicates and $\Omega^k$ is the $k$-th Cartesian power of $\Omega$). Note that the action parameters, $pars(\xi)$, are bound to the objects in $\omega$ that appear in the same position.

$$\mathsf{pre}(\mathsf{apply}_{\xi,\omega}) = \{pre\_e\_\xi \implies p(\omega)\}_{\forall e \in \mathcal{I}_{\Psi,\xi}},$$
$$\mathsf{cond}(\mathsf{apply}_{\xi,\omega}) = \{pre\_e\_\xi \wedge eff\_e\_\xi\} \rhd \{\neg p(\omega)\}_{\forall e \in \mathcal{I}_{\Psi,\xi}},$$
$$\{\neg pre\_e\_\xi \wedge eff\_e\_\xi\} \rhd \{p(\omega)\}_{\forall e \in \mathcal{I}_{\Psi,\xi}}\},$$
$$\{\emptyset\} \rhd \{\neg mode_{insert}\}.$$

The intuition of the compilation is that the dynamics of the actions for *applying* an action model $\xi$ is determined by the values of the corresponding $\{pre\_e\_\xi, eff\_e\_\xi\}_{\forall e \in \mathcal{I}_{\Psi,\xi}}$ fluents in the current state. For instance when executing `(apply_stack blockB blockA)` in a state $s$, its preconditions and effects are activated according to the values of the corresponding $\{pre\_e\_stack, eff\_e\_stack\}_{\forall e \in \mathcal{I}_{\Psi,stack}}$ fluents in $s$. This means that if the current state $s$ holds $\{(\texttt{pre\_stack\_holding\_v1}), (\texttt{pre\_stack\_clear\_v2})\} \subset s$, then it must be checked that positive literals `(holding blockB)` and `(clear blockA)` hold in $s$. Otherwise, a different set of precondition literals will be checked for the stack action. The same applies to the effects of `(apply_stack blockB blockA)`. Executing `(apply_stack blockB blockA)`, will add the literals `(on blockB blockA)`,`(clear blockB)`,`(not(clear blockA))`,`(handempty)` and `(not(clear blockB))` to the successor state only if the $\{pre\_e\_stack, eff\_e\_stack\}_{\forall e \in \mathcal{I}_{\Psi,stack}}$ fluents has been correctly programmed by the corresponding *insert* actions.

**00**:(insert_pre_stack_holding_v1)
01:(insert_pre_stack_clear_v2)
**02**:(insert_pre_pickup_handempty)
03:(insert_pre_pickup_clear_v1)
04:(insert_pre_pickup_ontable_v1)
**05**:(insert_eff_stack_clear_v1)
06:(insert_eff_stack_clear_v2)
07:(insert_eff_stack_handempty)
08:(insert_eff_stack_holding_v1)
09:(insert_eff_stack_on_v1_v2)
**10**:(insert_eff_pickup_clear_v1)
11:(insert_eff_pickup_ontable_v1)
12:(insert_eff_pickup_handempty)
13:(insert_eff_pickup_holding_v1)
**14**:(apply_pickup blockB)
15:(apply_stack blockB blockC)
16:(apply_pickup blockA)
17:(apply_stack blockA blockB)

Figure 2: Plan computed when solving the classical planning problem output by our compilation corresponding to a classical planning with unknown domain models.

Figure 2 shows a solution plan computed when solving a $P' = \langle F', A', I, G\rangle$ classical planning problem output by our compilation. In the initial state of that problem three blocks (`blockA`, `blockB` and `blockC`) are clear and on top of the table, the robot hand is empty. The problem goal is having the three-block tower `blockA` on top of `blockB` and `blockB` on top of `blockC`. The plan shows the *insert* actions for the `stack` scheme (steps $00 - 01$ insert the preconditions, steps $05 - 10$ insert the effects), steps $02 - 04$ insert the preconditions of the `pickup` scheme (while steps $10 - 13$ insert the effects of this scheme). Finally, steps $14 - 17$ is the plan postfix that applies the programmed action model to achieve the goals $G$ starting from $I$. Note that another valid solution could be computed for instance, by inserting the same preconditions and effects but into the *unstack* and *putdown* actions and then applying instead the four step postfix `(putdown blockB)`, `(unstack blockB blockC)`, `(putdown blockA)`, `(unstack blockA blockB)`.

## The bias of the initially *empty* action model

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 (**?**) 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 (**?**) 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 2, 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 2).

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.

## Compilation properties

**Lemma 1.** *Soundness. Any classical plan $\pi'$ that solves $P'$ produces a solution to the classical planning problem with unknown domain models $P = \langle F, A[\cdot], I, G\rangle$.*

*Proof.* Once a given precondition (or effect) is inserted into an action model it can never be removed back and once an action model is applied (via an $\mathsf{apply}_{\xi,\omega}$ action) it cannot longer be *programed*. The set of goals $G$ can only be achieved executing an applicable sequence of $\mathsf{apply}_{\xi,\omega}$ actions that, starting in the corresponding initial state reach a state $G \subseteq s_n$ (the fluents of the original problem $F$ can exclusively be modified by the $\mathsf{apply}_{\xi,\omega}$ actions). This means

that the action model used by the $\mathsf{apply}_{\xi,\omega}$ actions has to be consistent with the traversed intermediate states. We know that this must be true by the definition of the $\mathsf{apply}_{\xi,\omega}$ so hence, the subsequence of $\mathsf{apply}_{\xi,\omega}$ appearing in $\pi'$ to solve $P'$ is a solution plan to $P = \langle F, A[\cdot], I, G\rangle$. □

**Lemma 2.** *Completeness. Any plan $\pi$ that solves $P = \langle F, A[\cdot], I, G\rangle$ is computable solving the corresponding classical planning task $P'$.*

*Proof.* By definition $\mathcal{I}_{\Psi,\xi}$ fully captures the set of elements that can appear in an action model $\xi$ using predicates $\Psi$. Furthermore, the compilation does not discard any possible action model definable within $\mathcal{I}_{\Psi,\xi}$. This means that for every plan $\pi$ that solves $P = \langle F, A[\cdot], I, G\rangle$, we can build a plan $\pi'$ by selecting the appropriate actions for inserting precondition and effects to the corresponding action model and then selecting the corresponding $\mathsf{apply}_{\xi,\omega}$ actions that transform the initial state $I$ into a state $G \subseteq s_n$. □

The size of the classical planning task $P'$ output by our compilation depends on the arity of the given *predicates* $\Psi$ and the number of parameters of the action models, $|pars(\xi)|$. The larger these arities, the larger $|\mathcal{I}_{\Psi,\xi}|$. This term dominates the compilation size because it defines the $\{pre\_e\_\xi, eff\_e\_\xi\}$ fluents, the corresponding set of *insert* actions, and the number of conditional effects of the $\mathsf{apply}_{\xi,\omega}$ actions.

## Goal recognition as planning with unknown domain models

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

- $P = \langle F, A[\cdot], I, G[\cdot]\rangle$ 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 $\rho$ and/or $\theta$ of $a$ are undefined).
- $\mathcal{O}(\tau)$ is an observation of a trajectory $\tau(\pi, P)$ produced by the execution of an unknown plan $\pi$ that reaches the goals $G \in G[\cdot]$ starting from the initial state $I$ in $P$.

The *solution* to the *goal recognition with unknown domain models* task is again the subset of goals in $G[\cdot]$ that maximizes expression (1).

### Estimating the $P(\mathcal{O}|G)$ likelyhood with unknown domain models

Now we are ready to build an estimate of the $P(\mathcal{O}|G)$ likelyhood. Our mechanism matches the *plan recognition as planning* approach (**?**) except that we compute $cost(\pi^\top)$ using our compilation for *classical planning with unknown domain models*.

In more detail, we build the estimate of the $P(\mathcal{O}|G)$ likelyhood following these four steps:

1. *Build* $P^\top_G$, the classical planning problem that constrains solutions of the problem $P = \langle F, A[\cdot], s^o_0, G\rangle$ to plans $\pi^\top$ *consistent* with the input observation $\mathcal{O}(\tau)$. Note that $s^o_0 \in \mathcal{O}(\tau)$ is the initial state in the given observation.
2. *Solve* $P^\top_G$, using the proposed compilation for *classical planning with unknown domain models*. Extract from this solution (1), $cost(\pi^\top)$ (by counting the number of $\mathsf{apply}_{\xi,\omega}$ actions in the solution) but also (2), the action model $A$ that is determined by the *insert* actions used in $\pi^\top$ to achieve the goals $G$.
3. *Build* $P^\perp_G$, the classical planning problem that constrains $P = \langle F, A, s^o_0, G\rangle$ to achieve $G \in G[\cdot]$ through a plan $\pi^\perp$ *inconsistent* with $\mathcal{O}(\tau)$ (where $A$ is the set of actions extracted in step 2.).
4. *Solve* $P^\perp_G$ with a classical planner and extract $cost(\pi_\perp)$ as the length of the found solution plan.
5. Compute the $\Delta(\pi^\top, \pi^\perp)$ cost difference and plug it into equation (2) to get the $P(\mathcal{O}|G)$ likelihoods.

To compute the target probability distribution $P(G|\mathcal{O})$ plug the $P(\mathcal{O}|G)$ likelihoods into the *Bayes rule* from which the goal posterior probabilities are obtained. 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).

### Extending the observation model of *plan recognition as planning*

The work on *plan recognition as planning* usually assumes an observation model that is referred only to logs of executed actions. However, the approach applies also to more expressive observation models that consider state observations as well, like the observation model defined above, with a simple two-fold extension:

- One fluent $\{validated_j\}_{0\leq j\leq m}$ to point at every $s^o_j \in \mathcal{O}(\tau)$ state observation and $validated_m$ is added to every possible goal $G \in G[\cdot]$ to constrain solution plans $\pi^\top$ to be consistent with all the state observations.
- One $\mathsf{validate_j}$ action to constraint $\pi^\top$ to be consistent with the $s^o_j \in \mathcal{O}(\tau)$ input state observation, $(1 \leq j \leq m)$.

$$\mathsf{pre(validate_j)} = s^o_j \cup \{validated_{j-1}\},$$
$$\mathsf{cond(validate_j)} = \{\emptyset\} \rhd \{\neg validated_{j-1}, validated_j\}.$$

## Evaluation

## Related Work

The problem of *classical planning with unknown domain models* has been previously addressed (**?**). 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 (which it is typically a too strong assumption at many real-world applications).

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 (**?**) 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* (**?**). 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

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 (**?**; **?**; **?**). Our compilation approach is also flexible to this particular learning 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 $A_\Lambda$, making the classical planning task $P_\Lambda$ 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 $A_\Lambda$ action set without altering the *soundness* and *completeness* of the $P_\Lambda$ compilation.

### 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. AAAI Press.

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

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.

Ramırez, M., and Geffner, H. 2010. Probabilistic plan recognition using off-the-shelf classical planners. In *Proceedings of the Conference of the Association for the Advancement of Artificial Intelligence (AAAI 2010)*, 1121–1126.

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)*.

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.