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

The *plan recognition as planning* approach assumes that *observers* must have both correct and complete knowledge of the action model of the observed agents. This paper shows that this assumption can be relaxed formulating a novel setup for classical planning where action models are unknown but the state variables, action parameters and a single plan observation 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 for 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; Ramırez and Geffner 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 *counter-planning* (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 when applying *plan recognition as planning* on real-world problems. We formulate a novel set up for classical planning where no action model is given (instead, only the state variables, a single plan observation 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* that input the *goal recognition* task, and the *plan recognition as planning* approach for *goal recognition*.

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*). 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 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 a state s. Then, $\rho(s, a)$ holds iff $\mathsf{pre}(a) \subseteq s$, i.e. the action preconditions hold in the given state. 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 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 $\mathsf{eff}^+(a) \in \mathsf{eff}(a)$ while $\mathsf{eff}^-(a) \in \mathsf{eff}(a)$ denotes the *negative effects*.

Planning actions can also define *conditional effects*. Formally 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 $\mathsf{eff}_c(s, a) = \bigcup_{C \rhd E \in \mathsf{cond}(a_c), C \subseteq s} E$.

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 observations of τ as combinations of observable actions and observable fluents of the states of τ. We refer to the set of all possible combinations of observable elements of τ as $Obs(\tau)$.

We define *an observation* $\mathcal{O} \in Obs(\tau)$ as a sequence of possibly *partially observable 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 observable. A *partially observable state* is one in which $|s_i^o| < |F|$, $1 \leq i \leq m \leq n$; i.e., a state in which at least a fluent of F is not observable. It may be also the case that $|s_i^o| = 0$ when an intermediate state is fully unobservable.

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, as explained above) 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.

This observation model can distinguish between *observable state variables*, whose value may be read from sensors, and *hidden* (or *latent*) *state variables*, that cannot be observed. Given a subset of fluents $\Gamma \subseteq F$ we say that $\mathcal{O}$ is a Γ-observation of the execution of π on P iff for every observed state s_i^o, $1 \leq i \leq m$, s_i^o only contains fluents in Γ.

Goal recognition with classical planning

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

Following 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 (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 the particular goal $G \in G[\cdot]$ through a plan $\pi^{\top}$ that is *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).

Goal recognition as planning with unknown domain models

We define the task of *goal recognition with unknown domain models* as a $\langle P, \mathcal{O} \rangle$ 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 ρ and/or θ of a are undefined).
- $\mathcal{O}$ is the observation of the execution of an unknown plan π 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

To build an estimate of the $P(\mathcal{O}|G)$ likelyhood our mechanism matches the *plan recognition as planning* approach (Ramírez 2012) 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* $P^{\bot}_G$, the classical planning problem that constrains $P = \langle F, A, s^o_0, G\rangle$ to achieve $G \in G[\cdot]$ through a plan $\pi^{\bot}$ *inconsistent* with $\mathcal{O}(\tau)$ (where A is the set of actions extracted in step 2.).
4. *Solve* $P^{\bot}_G$ with a classical planner and extract $cost(\pi_{\bot})$ as the length of the found solution plan.
5. Compute the $\Delta(\pi^{\top}, \pi^{\bot})$ 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).

Planning with unknown domain models

This section formulates *planning with unknown domain models*, a novel 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) and can be seen as an extreme learning task where action models must be *learned* from a single example that contain only two state observations: the initial state and the goals.

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:

- For every $1 \leq i \leq n$ then $\rho(s_{i-1}, a_i) = \texttt{True}$.
- For every $1 \leq i \leq n$ then $s_i = \theta(s_{i-1}, a_i)$.
- The reached final state s_n meets the goals $G \subseteq s_n$.

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 STRIPS action

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.

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 1: Combinations of the propositional encoding and their meaning

Let Ψ be the set of *predicates* that shape the fluents F (the initial state of an observation is a full assignment of values to fluents, $|s^o_0| = |F|$, and so the predicates Ψ are extractable from the observed state s^o_0). 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`)`}. Hence, solving a $\Lambda = \langle \mathcal{M}, \mathcal{O}\rangle$ learning task is determining which elements of $\mathcal{I}_{\xi,\Psi}$ will shape the preconditions, positive and negative effects of the corresponding action model.

A valid actoin 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 1.

To illustrate better this encoding, Figure 2 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 ($p \in \mathcal{I}_{\Psi,stack}$).

```
(: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 2: 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* (Aineto, Jiménez, and Onaindia 2018), 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 ξ following the syntactic constraints of STRIPS .
     - Actions to insert an $e \in \mathcal{I}_{\Psi,\xi}$ *precondition* into ξ. The precondition is only inserted when neither pre_e_ξ nor eff_e_ξ exist in ξ.

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

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

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

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

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

The intuition of the compilation is that the dynamics of the actions for *applying* an action model ξ 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 3: Plan computed when solving the classical planning problem output by our compilation corresponding to a classical planning with unknown domain models.

Figure 3 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 post-

fix `(putdown blockB)`, `(unstack blockB blockC)`, `(putdown blockA)`, `(unstack blockA blockB)`.

Compilation properties

Lemma 1. *Soundness. Any classical plan π' 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 π' to solve P' is a solution plan to $P = \langle F, A[\cdot], I, G\rangle$. □

Lemma 2. *Completeness. Any plan π 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 ξ using predicates Ψ. Furthermore, the compilation does not discard any possible action model definable within $\mathcal{I}_{\Psi,\xi}$. This means that for every plan π that solves $P = \langle F, A[\cdot], I, G\rangle$, we can build a plan π' 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* Ψ 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.

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 (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 3, 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 3).

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.

Constraining the action model space

In some contexts it is however reasonable to assume that some portions of the action model are known (Zhuo, Nguyen, and Kambhampati 2013; Sreedharan, Chakraborti, and Kambhampati 2018; Pereira and Meneguzzi 2018). 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.

Domain-specific knowledge is also helpfull to constrain further the space of possible action 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 ξ 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) (Fox and Long 1998).

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

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.