

# Model Recognition as Planning

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@nimb.edu.au

## Abstract

Given a partially observed plan execution, and a set of possible planning models (models that share the same state variables but that update these variables according to different action models); *model recognition* is the task of identifying which model in the set produced/explains the given observation. The paper formalizes the *model recognition* task and introduces a novel method to estimate the probability of a STRIPS model to produce an observation of a plan execution. This method, that we called *model recognition as planning*, is built on top of off-the-shelf classical planning algorithms and is robust to missing intermediate states and actions in the observation of the plan execution. The effectiveness of *model recognition as planning* is shown in a set of STRIPS models encoding different kinds of *automata*. We show that *model recognition as planning* succeeds to identify the executed automata despite the actual applied transitions, plus some state variables e.g. the internal automata state, are unobserved.

## Introduction

*Plan recognition* is the task of predicting the future actions of an agent provided observations of its current behavior (Carberry 2001). *Goal recognition* is a closely related task that aims identifying the goals of the observed agent. Goal recognition is considered *automated planning* in reverse; while automated planning compute sequences of actions that accounts for a given goals, goal recognition compute goals that account for an observed sequence of actions (Geffner and Bonet 2013).

Diverse approaches has been proposed for plan/goal recognition such as *rule-based systems*, *parsing*, *graph-covering*, *Bayesian nets*, etc (Sukthankar et al. 2014). *Plan recognition as planning* is the model-based approach for plan/goal recognition that leverages the action model of the observed agent to compute its most likely goal (Ramírez 2012; Ramírez and Geffner 2009).

This paper formalizes the *model recognition* task where the object to recognize is not a goal but the *planning model* that shapes the behavior of the observed agent. Given a partially observed plan execution, and a set of possible planning models (models that share the same state variables but that update these variables with different action models);

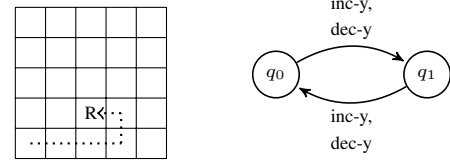


Figure 1: (Left) Robot navigating a  $5 \times 5$  grid. (Right) Automata to control that the robot only increments its x-coordinate when  $q_0$  holds (actions *inc-y* and *dec-y* update the robot y-coordinate and switch the automata state).

*model recognition* is the task of identifying the model in the set with the highest probability of producing/explaining the given observation.

To better illustrate *model recognition*, imagine a robot in a  $N \times N$  grid whose navigation is determined by the STRIPS model of Figure 2. According to this model, the robot can increment its *x-coordinate* when  $q_0$  holds (i.e. at *even rows* if  $q_0$  holds initially) and decrement it when  $q_1$  holds (at *odd rows* if  $q_0$  holds initially). Apart from the model of Figure 2, different models can be defined within the same state variables (e.g. altering the way  $q_0$  and  $q_1$  are required and updated) and these models can shape different kinds of robot navigation. Given a partially observed plan execution (like the one illustrated at Figure 1 where state variables  $q_0$  and  $q_1$  are unobserved) *model recognition* aims to identify the action model that produced/explains that observation.

*Model recognition* is of interest because once the planning model is recognized, then the model-based machinery for automated planning becomes applicable (Ghallab, Nau, and Traverso 2004). In addition, it enables identifying different kinds of automata by observing their execution. It is well-known that diverse automata representations, like *finite state controllers*, *Büchi automata*, *push-down automata*, *GOLOG programs* or *reactive policies*, can be encoded as classical planning models (Baier, Fritz, and McIlraith 2007; Bonet, Palacios, and Geffner 2010; Patrizi et al. 2011; Ivankovic and Haslum 2015; Segovia-Aguas, Jiménez, and Jonsson 2017).

The paper also introduces *model recognition as planning*; a novel method to estimate the probability of a given STRIPS model to produce an observed plan execution. Our method

```

(:action inc-x
:parameters (?v1 ?v2)
:precondition (and (xcoord ?v1) (next ?v1 ?v2) (q0))
:effect (and (not (xcoord ?v1)) (xcoord ?v2)))

(:action dec-x
:parameters (?v1 ?v2)
:precondition (and (ycoord ?v1) (next ?v2 ?v1) (q1))
:effect (and (not (xcoord ?v1)) (xcoord ?v2)))

(:action inc-y-even
:parameters (?v1 ?v2)
:precondition (and (ycoord ?v1) (next ?v1 ?v2) (q0))
:effect (and (not (ycoord ?v1)) (ycoord ?v2)
(not (q0)) (q1)))

(:action inc-y-odd
:parameters (?v1 ?v2)
:precondition (and (ycoord ?v1) (next ?v1 ?v2) (q1))
:effect (and (not (ycoord ?v1)) (ycoord ?v2)
(not (q1)) (q0)))

(:action dec-y-even
:parameters (?v1 ?v2)
:precondition (and (ycoord ?v1) (next ?v2 ?v1) (q0))
:effect (and (not (ycoord ?v1)) (ycoord ?v2)
(not (q0)) (q1)))

(:action dec-y-odd
:parameters (?v1 ?v2)
:precondition (and (ycoord ?v1) (next ?v2 ?v1) (q1))
:effect (and (not (ycoord ?v1)) (ycoord ?v2)
(not (q1)) (q0)))

```

Figure 2: Example of a STRIPS action model (given in PDDL) for robot navigation in a  $N \times N$  grid.

is built on top of off-the-shelf classical planning algorithms and is robust to missing intermediate states and actions in the observed plan execution. We evaluate the effectiveness of *model recognition as planning* with sets of STRIPS models that represent different *automata*. All the *automata* in a set are defined within the same state variables but with different *transition functions*. We show that *model recognition as planning* identifies the executed *automata* despite the actual applied transitions, plus some state variables (e.g. the internal automata state), are unobserved.

## Background

This section formalizes the models for *classical planning* and for the *observation* of the execution of a classical plan.

### Classical planning

We use  $F$  to denote the set of *fluents* (propositional variables) 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 assign conflicting values to any fluent). 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 and we explicitly include negative literals  $\neg f$  in states; i.e.  $|s| = |F|$ , so the size of the state space is  $2^{|F|}$ . Like in PDDL (Fox and Long 2003), we assume that fluents  $F$  are instantiated from a set of *predicates*  $\Psi$ . Each predicate  $p \in \Psi$  has an argument list of arity  $ar(p)$ . Given a set of *objects*  $\Omega$ , the set of fluents  $F$  is induced by assigning objects in  $\Omega$  to the

arguments of predicates in  $\Psi$ ; i.e.  $F = \{p(\omega) : p \in \Psi, \omega \in \Omega^{ar(p)}\}$  such that  $\Omega^k$  is the  $k$ -th Cartesian power of  $\Omega$ .

A *classical planning frame* is a  $\langle F, A \rangle$  pair, where  $F$  is a set of *fluents* and  $A$  is a set of *actions*. The semantics of actions  $a \in A$  is specified with two functions:  $\rho(s, a)$  that denotes whether action  $a$  is *applicable* in a state  $s$  and  $\theta(s, a)$  that denotes the *successor state* that results of applying action  $a$  in a state  $s$ . In this work we compactly represent the  $\rho$  and  $\theta$  functions of an action  $a \in A$  with the STRIPS model:

- $\text{pre}(a) \in \mathcal{L}(F)$ , the *preconditions* of  $a$ , is the set of literals that must hold for the action  $a \in A$  to be applicable.
- $\text{eff}^+(a) \in \mathcal{L}(F)$ , the *positive effects* of  $a$ , is the set of literals that are true after the application of action  $a \in A$ .
- $\text{eff}^-(a) \in \mathcal{L}(F)$ , the *negative effects* of  $a$ , is the set of literals that are false after the application of the action.

We say that an action  $a \in A$  is *applicable* in a state  $s$  iff  $\text{pre}(a) \subseteq s$ . The result of applying  $a$  in  $s$  is the *successor state* denoted by  $\theta(s, a) = \{s \setminus \text{eff}^-(a)\} \cup \text{eff}^+(a)$ .

A *classical planning problem* is a tuple  $P = \langle F, A, I, G \rangle$ , where  $I$  is an initial state and  $G \in \mathcal{L}(F)$  is a goal condition. A *plan*  $\pi$  is an action sequence  $\pi = \langle a_1, \dots, a_n \rangle$  and  $|\pi| = n$  denotes its *plan length*. The execution of a plan  $\pi$  on a classical planning problem  $P$  induces a *trajectory*  $\tau(\pi, P) = \langle a_1, s_1, \dots, a_n, s_n \rangle$  such that  $s_0 = I$  and, for each  $1 \leq i \leq n$ ,  $a_i$  is applicable in  $s_{i-1}$  and generates the successor state  $s_i = \theta(s_{i-1}, a_i)$ . A plan  $\pi$  *solves*  $P$  iff  $G \subseteq s_n$ , i.e., if the goal condition is satisfied at the last state reached after following the execution of plan  $\pi$  on problem  $P$ . A solution plan for  $P$  is *optimal* iff it has minimum length.

### Conditional effects

*Conditional effects* allow classical planning actions to have different semantics according to the value of the current state. This feature is useful for compactly defining our method for *model recognition as planning*.

An action  $a \in A$  with conditional effects is defined as a set of *preconditions*  $\text{pre}(a) \in \mathcal{L}(F)$  and a set of *conditional effects*  $\text{cond}(a)$ . Each conditional effect  $C \triangleright E \in \text{cond}(a)$  is composed of two sets of literals  $C \in \mathcal{L}(F)$ , the *condition*, and  $E \in \mathcal{L}(F)$ , the *effect*.

An action  $a \in A$  is *applicable* in a state  $s$  iff  $\text{pre}(a) \subseteq s$ , and the *triggered effects* resulting from the action application are the effects whose conditions hold in  $s$ :

$$\text{triggered}(s, a) = \bigcup_{C \triangleright E \in \text{cond}(a), C \subseteq s} E,$$

The result of applying action  $a$  in state  $s$  is the *successor state*  $\theta(s, a) = \{s \setminus \text{eff}_c^-(s, a)\} \cup \text{eff}_c^+(s, a)$  where  $\text{eff}_c^-(s, a) \subseteq \text{triggered}(s, a)$  and  $\text{eff}_c^+(s, a) \subseteq \text{triggered}(s, a)$  are, respectively, the triggered *negative* and *positive* effects.

### The observation model

Given a classical planning problem  $P = \langle F, A, I, G \rangle$ , a solution plan  $\pi$  for  $P$ , and a trajectory  $\tau(\pi, P)$  generated by the execution of  $\pi$  on  $P$ ; the observation of the execution of

$\pi$  on  $P$  is  $\mathcal{O}(\tau) = \langle a_1^o, s_1^o, \dots, a_l^o, s_m^o \rangle$ , an interleaved combination of  $1 \leq l \leq |\pi|$  observed actions and  $1 \leq m \leq |\pi|$  observed states:

- **Observed actions** are *consistent* with  $\pi$ , which means that  $\langle a_1^o, \dots, a_l^o \rangle$  is a sub-sequence of  $\pi$ .
- **Observed states** are a sequence of *partial states*  $\langle s_1^o, \dots, s_m^o \rangle$  that is *consistent* with the sequence of states  $\langle s_0, s_1, \dots, s_n \rangle$  traversed by the trajectory  $\tau(\pi, P)$ . The initial state  $I$  is fully observed while  $\langle s_1^o, \dots, s_m^o \rangle$  may be partial states, i.e. the value of certain fluents in the intermediate states may not be observed ( $|s_i^o| \leq |F|$  for  $1 \leq i \leq m$ ). Even more, certain intermediate states may be fully unobserved ( $0 \leq |s_i^o|$  for  $1 \leq i < m$ ).

We are assuming that a bijective monotone mapping exists between plans and observations (Ramírez and Geffner 2009). Therefore transiting between two consecutive observed states in  $\mathcal{O}(\tau)$  may require the execution of more than a single action ( $\theta(s_i^o, \langle a_1, \dots, a_k \rangle) = s_{i+1}^o$ , where  $k \geq 1$  is unknown but finite). In other words, having  $\mathcal{O}(\tau)$  does not imply knowing the actual length of  $\pi$ .

Figure 1 illustrates the *partial observation* of a five-state trajectory  $\{ \langle (\text{xcoord } 2) (\text{ycoord } 1) \rangle, \langle (\text{xcoord } 3) (\text{ycoord } 1) \rangle, \langle (\text{xcoord } 4) (\text{ycoord } 1) \rangle, \langle (\text{xcoord } 4) (\text{ycoord } 2) \rangle, \langle (\text{xcoord } 3) (\text{ycoord } 2) \rangle \}$ . In more detail, this observation only contains fluents of the kind  $(\text{xcoord } ?v)$  and  $(\text{ycoord } ?v)$ . The value of the remaining fluents, namely  $(\text{next } ?v1 \text{ } ?v2)$ ,  $(q0)$  and  $(q1)$ , cannot be observed.

**Definition 1 ( $\Phi$ -observation).** Given a subset of fluents  $\Phi \subseteq F$  we say that  $\mathcal{O}(\tau)$  is a  $\Phi$ -observation of the execution of  $\pi$  on  $P$  iff, for every  $1 \leq i \leq m$ , each observed state  $s_i^o$  only contains fluents in  $\Phi$ .

## Model Recognition

The *model recognition* task is a tuple  $\langle P, M, \mathcal{O} \rangle$  where:

- $P = \langle F, A[\cdot], I, G \rangle$  is a *classical planning problem* where  $A[\cdot]$  is a set of actions s.t. the semantics of actions  $a \in A[\cdot]$  is unknown (i.e. for each  $a \in A[\cdot]$  its corresponding  $\rho$  and/or  $\theta$  functions are undefined).
- $M = \{\mathcal{M}_1, \dots, \mathcal{M}_k\}$  is a *set of  $k$  different models* for the actions in  $A[\cdot]$  s.t., each model in  $\mathcal{M} \in M$ , defines different semantics for the actions in  $A[\cdot]$  (i.e. different function pairs  $\langle \rho, \theta \rangle$ ) but using the same set of state variables  $F$ .
- $\mathcal{O}(\tau)$  is an *observation* of the execution of an unknown solution plan  $\pi$  for the planning problem  $P$ .

*Model recognition* can be understood as a *classification task* where each class is represented with a different planning model  $\mathcal{M} \in M$  and the observed plan execution  $\mathcal{O}(\tau)$ , is the single example to classify. The planning model associated to each class acts as the corresponding *class prototype* and it summarizes any observation of a plan execution that could be synthesized with that model (i.e. the set of all the examples that belong to that class). To assign a model  $\mathcal{M} \in M$  to a given observation  $\mathcal{O}(\tau)$ , we follow the *naive Bayes classifier*. The *solution* to the *model recognition* task



Figure 3: *Bayesian network* abstracting that model  $\mathcal{M}$  can be transformed into a model  $\mathcal{M}'$  that produces a trajectory  $\tau(\pi, P)$  that (1), reaches the goals in  $P$  and (2), is consistent with  $\mathcal{O}(\tau)$ .

is then the subset of models in  $M$  that maximizes this expression.

$$\operatorname{argmax}_{\mathcal{M} \in M} P(\mathcal{O}|\mathcal{M})P(\mathcal{M}). \quad (1)$$

## Formulating the $P(\mathcal{O}|\mathcal{M})$ likelihood

The  $P(\mathcal{M})$  probability expresses whether one model is known to be *a priori* more likely than the others. If this probability is not given as input it is reasonable to assume that, *a priori*, all models are equiprobable. The challenge in our formulation for *model recognition* is the definition of the  $P(\mathcal{O}|\mathcal{M})$  *likelihood* that expresses the probability of observing  $\mathcal{O}(\tau)$  when  $\mathcal{M}$  is the planning model.

Our approach to formulate the  $P(\mathcal{O}|\mathcal{M})$  *likelihood* is to assess the cost of *transforming*  $\mathcal{M}$  into a model  $\mathcal{M}'$  that produces a trajectory  $\tau(\pi, P)$  such that: (1),  $\tau(\pi, P)$  reaches the goals in  $P$  and (2),  $\tau(\pi, P)$  is consistent with observation  $\mathcal{O}(\tau)$ . Figure 3 shows the *Bayesian network* that abstracts this procedure. Regarding this network we have the following formulation of the  $P(\mathcal{O}|\mathcal{M})$  *likelihood*:

$$P(\mathcal{O}|\mathcal{M}) = \sum_{\mathcal{M}'} \sum_{\tau} P(\mathcal{M}'|\mathcal{M})P(\tau|\mathcal{M}')P(\mathcal{O}|\tau), \quad (2)$$

where  $\tau$  ranges over all the trajectories consistent with  $\mathcal{O}(\tau)$  that can be synthesized with a model  $\mathcal{M}'$  and  $\mathcal{M}'$  ranges over all the models that can be generated *transforming*  $\mathcal{M}$ .

The exact computation of  $P(\mathcal{O}|\mathcal{M})$  with equation (2) is intractable. For most planning problems the set of trajectories consistent with an arbitrary observation can easily be huge, infinite in the case of planning problems without dead-ends (Lesh and Etzioni 1995). Even worse, the number of models  $\mathcal{M}'$  that can be generated *transforming* a given classical planning model  $\mathcal{M}$  explodes combinatorially with the number of state variables. Instead, our approach is to formulate an effectively estimate of  $P(\mathcal{O}|\mathcal{M})$  using an *edit distance* referred to STRIPS planning models. *Edit distances* are similarity metrics, traditionally computed over *strings* or *graphs*, and that has been proved successful for *pattern recognition* (Masek and Paterson 1980; Bunke 1997). In this work, we assess the cost of *transforming* a classical planning model  $\mathcal{M}$  into a model  $\mathcal{M}'$  computing *edit distances* between STRIPS planning models.

## Recognition of STRIPS models

Here we study the particular instantiation of *model recognition* where the semantics of models  $\mathcal{M} \in M$  (i.e. the corresponding  $\rho$  and  $\theta$  functions) is specified with STRIPS action schemata. We start formalizing STRIPS schemata and define the space of possible STRIPS schemata for a given set of state variables. Eventually, we introduce an *edit distance* for STRIPS schemata to estimate the  $P(\mathcal{O}|\mathcal{M})$  likelihoods for classical planning models.

## Well-defined STRIPS action schemata

STRIPS action schemata provide a compact representation for specifying classical planning models. Figure 2 shows six STRIPS action schemata that shape a particular kind of robot navigation in  $N \times N$  grids (no matter the grid size).

A STRIPS action schemata  $\xi$  is defined by four lists: A list of parameters  $pars(\xi)$ , and three list 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 schemata.

**Definition 2** (Comparable STRIPS action schemata). *Two STRIPS schemata  $\xi$  and  $\xi'$  are comparable iff  $pars(\xi) = pars(\xi')$ , both share the same list of parameters.*

For instance, we claim that the six action schemata of Figure 2 are *comparable* while, for example, the `stack(?v1, ?v2)` and `pickup(?v1)` schemata from a four operator *blocksworld* (Slaney and Thiébaux 2001) are not. Last but not least, we say that two STRIPS models  $\mathcal{M}$  and  $\mathcal{M}'$  are *comparable* iff there exists a bijective function  $\mathcal{M} \mapsto \mathcal{M}'$  that maps every action schemata  $\xi \in \mathcal{M}$  to a comparable schemata  $\xi' \in \mathcal{M}'$  and vice versa.

Given the set of predicates  $\Psi$ , that shape the propositional state variables, and a list of parameters  $pars(\xi)$ . The set of elements that can appear in the  $pre(\xi)$ ,  $del(\xi)$  and  $add(\xi)$  of the STRIPS action schemata  $\xi$  is given by FOL interpretations of  $\Psi$  over the parameters  $pars(\xi)$ . We denote this set of FOL interpretations as  $\mathcal{I}_{\Psi, \xi}$ . For any of the six action schemata defined in Figure 2 the  $\mathcal{I}_{\Psi, \xi}$  set contains the same ten elements,  $\mathcal{I}_{\Psi, \xi} = \{xcoord(v_1), xcoord(v_2), ycoord(v_1), ycoord(v_2), q0(), q1(), next(v_1, v_1), next(v_1, v_2), next(v_2, v_1), next(v_2, v_2)\}$ .

Despite any element from  $\mathcal{I}_{\Psi, \xi}$  can *a priori* appear in the  $pre(\xi)$ ,  $del(\xi)$  and  $add(\xi)$  of schemata  $\xi$ , the space of possible STRIPS schemata is constrained by a set  $\mathcal{C}$  that includes:

- *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  $2^{2 \times |\mathcal{I}_{\Psi, \xi}|}$ . For every action schemata in the navigation model of Figure 2 then  $2^{2 \times 10} = 1,048,576$ .
- *Domain-specific constraints.* One can introduce domain-specific knowledge to constrain further the space of possible schemata. For instance in a *robot navigation* model like the one in Figure 2, `q0()` and `q1()` are exclusive so they cannot hold at the same time in a  $pre(\xi)/del(\xi)/add(\xi)$  list. Further, `next(v1, v1)` and `next(v2, v2)` will not appear at any of these lists because the `next` predicate codes the *successor* function for *natural numbers*. These domain-specific constraints reduce further the size of the space of possible action schemata to  $2^{2 \times 7} = 16,384$  (for every schemata in the navigation model of Figure 2).

**Definition 3** (Well-defined STRIPS action schemata). *Given a set of predicates  $\Psi$ , a list of action parameters  $pars(\xi)$ , and set of FOL constraints  $\mathcal{C}$  then  $\xi$  is a **well-defined STRIPS***

*action schemata* iff its three lists  $pre(\xi) \subseteq \mathcal{I}_{\Psi, \xi}$ ,  $del(\xi) \subseteq \mathcal{I}_{\Psi, \xi}$  and  $add(\xi) \subseteq \mathcal{I}_{\Psi, \xi}$  only contain elements in  $\mathcal{I}_{\Psi, \xi}$  and they satisfy all the constraints in  $\mathcal{C}$ .

We say a classical planning action model is *well-defined* if all its STRIPS action schemata are *well-defined*.

## Edit distances for STRIPS action models

First, we define the two edit operations on a schemata  $\xi \in \mathcal{M}$  that belongs to a STRIPS action model  $\mathcal{M} \in \mathcal{M}$ :

- *Deletion.* An element is removed from a  $pre(\xi)/del(\xi)/add(\xi)$  list of operator schemata  $\xi \in \mathcal{M}$  such that the resulting schemata is a *well-defined* STRIPS action schemata.
- *Insertion.* An element in  $\mathcal{I}_{\Psi, \xi}$  is added to a  $pre(\xi)/del(\xi)/add(\xi)$  list of operator schemata  $\xi \in \mathcal{M}$  s.t. the resulting schemata is *well-defined*.

We can now formalize an *edit distance* that quantifies how similar two given STRIPS action models are. The distance is symmetric and meets the *metric axioms* provided that the two edit operations, *deletion* and *insertion*, have the same positive cost.

**Definition 4** (Edit distance). *Let  $\mathcal{M}$  and  $\mathcal{M}'$  be two comparable and well-defined STRIPS action models defined within the same set of predicates  $\Psi$ . The **edit distance**  $\delta(\mathcal{M}, \mathcal{M}')$  is the minimum number of edit operations that is required to transform  $\mathcal{M}$  into  $\mathcal{M}'$ .*

Since  $\mathcal{I}_{\Psi, \xi}$  is a bounded set, the maximum number of edits that can be introduced to an action model is bounded as well. The **maximum edit distance** of a STRIPS model  $\mathcal{M}$  built within predicates  $\Psi$  is  $\delta(\mathcal{M}, *) = \sum_{\xi \in \mathcal{M}} 3 \times |\mathcal{I}_{\Psi, \xi}|$  (note that, if we consider the syntactic STRIPS constraints,  $\delta(\mathcal{M}, *) = \sum_{\xi \in \mathcal{M}} 2 \times |\mathcal{I}_{\Psi, \xi}|$ ).

An observation  $\mathcal{O}(\tau)$  of the execution of a plan generated with an action model  $\mathcal{M}$ , reflects *semantic knowledge* that constrains further the space of possible schemata  $\xi \in \mathcal{M}$ . We talk then about a third kind of constraints, that we call *Observation constraints*, and that can be added to the  $\mathcal{C}$  set. In addition, *observation constraints* allow us to define an edit distance to elicit the  $P(\mathcal{O}|\mathcal{M})$  likelihoods. It can be argued that the shorter this distance the better the given model explains the given observation.

**Definition 5** (Observation edit distance). *Given a classical planning problem  $P$ , an observation  $\mathcal{O}(\tau)$  of the execution of a plan for solving that problem and a STRIPS action model  $\mathcal{M}$  (all defined within the same predicates  $\Psi$ ). The **observation edit distance**,  $\delta(\mathcal{M}, \mathcal{O})$ , is the minimal edit distance from  $\mathcal{M}$  to any comparable and well-defined model  $\mathcal{M}'$  s.t.  $\mathcal{M}'$  produces a trajectory  $\tau(\pi, P)$  that reaches the goals in  $P$  and is consistent with  $\mathcal{O}(\tau)$ ;*

$$\delta(\mathcal{M}, \mathcal{O}) = \min_{\forall \mathcal{M}' \rightarrow \mathcal{O}} \delta(\mathcal{M}, \mathcal{M}')$$

The *observation edit distance* could also be defined assessing the edition required by the observed plan execution to fit the given model. This implies defining *edit operations* that modify the observation  $\mathcal{O}(\tau)$  instead of modifying the model  $\mathcal{M}$  (Yang, Wu, and Jiang 2007; Sohrabi, Riabov,

and Udrea 2016). Our definition of the *observation edit distance* is more practical since normally,  $\mathcal{I}_{\Psi, \xi}$  is much smaller than  $F$  (the number of variables in the schemata should be smaller than the number of objects in a planning problem). What is more, our definition of the *observation edit distance* does not accumulate mismatches between the observation and the model that are due to the same source (the same model flaw).

**Definition 6 (Closest consistent models).** Given a model  $\mathcal{M}$ , the set  $M^*$  of the **closest consistent models** is the set of models  $\mathcal{M}'$  that: (1) can produce a trajectory  $\tau(\pi, P)$  that reaches the goals in  $P$  and is consistent with  $\mathcal{O}(\tau)$  and (2), their edit distance to  $\mathcal{M}$  is minimal;

$$\arg \min_{\forall \mathcal{M}' \rightarrow \mathcal{O}} \delta(\mathcal{M}, \mathcal{M}')$$

### Approximating the $P(\mathcal{O}|\mathcal{M})$ likelihood

Now we are ready to formulate an informative estimate of the  $P(\mathcal{O}|\mathcal{M})$  likelihood for the particular case where models  $\mathcal{M}$  are specified with STRIPS schemata.

**Full observability of the executed plan.** The *full observability of the executed plan* is a too strong assumption for *model recognition* but it allows us to understand how to build a reasonable estimate of  $P(\mathcal{O}|\mathcal{M})$  for the general case.

Under the assumption of *full observability*, there is only a single possible trajectory  $\tau^*(\pi, P)$  consistent with the input observation so  $P(\mathcal{O}|\tau^*) = 1$ . Further, there is also a single model that can exactly produce that trajectory (otherwise models are identical, at least, in the actions relevant to the observed trajectory so they can be considered the same model).

Provided that there is a single possible trajectory and a single possible model consistent with the input observation, i.e.  $M^* = \{\mathcal{M}^*\}$  then, the probabilities of expression (2) are not added up and expression (1) simplifies to:

$$\arg \max_{\mathcal{M} \in M} P(\mathcal{M}^*|\mathcal{M})P(\mathcal{M}). \quad (3)$$

Note that the term  $P(\tau|\mathcal{M}^*)$  is taken out of the maximization because it is independent of the input model  $\mathcal{M} \in M$ .

**Partial observability of the executed plan.** In a similar way, an approximation of the  $P(\mathcal{O}|\mathcal{M})$  likelihood can be built for the general case, where the executed plan is partially observed. We add the following two assumptions to deal with this general case:

1. *The principle of rationality:* The expectation that intentional agents will tend to choose actions that achieve their goals most efficiently, given their knowledge of the actual world state (Dennett 1983).
2. *The best plans for  $P$  are unique or have different cost.*

Similar assumptions were taken in previous work for *goal recognition* (Ramírez 2012). They allow us to assume that the sum in equation (2) is dominated by its largest term, so other terms in the sum are not added up. The largest term corresponds here to the shortest trajectory  $\tau^*$  consistent with the observation (which now is assumed to be unique). Note

that the more complete is the observation of the plan execution the more accurate is our estimate because it becomes more likely to assume that there is only one trajectory consistent with the input observation.

Again, we have that there is a single model  $\mathcal{M}^*$  that can produce a given trajectory  $\tau^*$ . With this said, both  $P(\mathcal{O}|\tau^*)$  and  $P(\tau^*|\mathcal{M}^*)$  are independent of  $\mathcal{M} \in M$  so equation (1) simplifies once again to equation (3) under the previous two assumptions.

**The  $P(\mathcal{M}^*|\mathcal{M})$  probability distribution.**  $P(\mathcal{M}'|\mathcal{M})$  indicates the probability of transforming a classical planning model  $\mathcal{M}$  into a model  $\mathcal{M}'$  by exclusively using the two *edit operations* previously defined, *deletion* and *insertion*.

We know from *pattern recognition* (Devroye, Györfi, and Lugosi 2013) that, if string symbols are uniformly random and independent, the distance of a given string to a fixed string follows a *Binomial distribution*. Moreover, the probability that a particular string will be within a distance  $D$  is the Cumulative Distribution Function (CDF) for the *Binomial distribution* (Wilcoxon 1981):

$$CDF(D) = \sum_{d=0}^D \binom{N}{d} p^d (1-p)^{N-d} \quad (4)$$

where  $N$  is the length of the string, and  $p < 0.5$  since we consider that the cost of applying an *edit operation* is higher than do not applying it.

With this regard, and considering that STRIPS models  $\mathcal{M} \in M$  can be encoded with a propositional representation of fixed length  $N$ , we formulate the  $P(\mathcal{M}'|\mathcal{M})$  probability distribution mapping the distance  $\delta(\mathcal{M}, \mathcal{M}')$  according to equation:

$$P(\mathcal{M}'|\mathcal{M}) = p^d (1-p)^{N-d} \quad (5)$$

where  $d = \delta(\mathcal{M}, \mathcal{M}')$ . Likewise we can formulate  $P(\mathcal{M}^*|\mathcal{M})$  with this same equation (5) but instead, the parameter  $d$  is given by the *observation distance*,  $d = \delta(\mathcal{M}, \mathcal{O})$ .

In other words, we are modeling the edition of a STRIPS planning model as a *Bernoulli process* in which there is a sequence of  $N$  independent events representing  $N$  binary decisions (the  $N$  possible applications of the edition operations) such that for every of these events  $P(X = \top) = p$  and  $P(X = \perp) = 1 - p$ .

### Model Recognition as classical planning

This section shows that, for STRIPS planning models,  $\delta(\mathcal{M}, \mathcal{O})$  can be computed (and hence an approximation  $P(\mathcal{O}|\mathcal{M})$ ) with a classical planning compilation. The compilation is inspired by the classical planning compilation for the learning of STRIPS planning models (Aineto, Jiménez, and Onaindia 2018). The intuition behind our compilation is that a solution to the resulting classical planning task is a sequence of actions that:

1. **Edits the action model  $\mathcal{M}$  to build  $\mathcal{M}'$ .** A solution plan starts with a *prefix* that modifies the preconditions and effects of the action schemata in  $\mathcal{M}$  using the two *edit operations* defined above, *deletion* and *insertion*.

```

00 : (insert_pre_xcoord.v1.inc-x)    07 : (validate-0)
01 : (insert_pre_next.v1.v2.inc-x)  08 : (editable.inc-x 1 2)
02 : (insert_pre_q0.inc-x)          09 : (editable.inc-x 2 3)
03 : (delete_del_xcoord.v2.inc-x)    10 : (editable.inc-x 3 4)
04 : (delete_add_xcoord.v1.inc-x)    11 : (editable.inc-y-even 1 2)
05 : (insert_del_xcoord.v1.inc-x)    12 : (editable.dec-x 4 3)
06 : (insert_add_xcoord.v2.inc-x)    13 : (validate-1)

```

Figure 4: Plan for editing (steps [0-6]) and validating (steps [7-13]) the model of Figure2 when schemata `inc-x` has no *preconditions* and positive/negative *effects* are swapped wrt Figure2.

2. **Validates the edited model  $\mathcal{M}'$ .** The solution plan continues with a postfix that induces a solution plan  $\pi$  for the original classical planning problem  $P$  and validates that  $\mathcal{O}(\tau)$  is an observation of the execution of  $\pi$  on  $P$ .

The value of the *observation distance*  $\delta(\mathcal{M}, \mathcal{O})$  is given by the number of *edit operations* (insertions and deletions) required by  $\mathcal{M}$  to be validated in the input observation.

Figure 4 shows the plan with a prefix (steps [0,6]) for editing the planning model of Figure2 when its schemata `inc-x` is defined without preconditions and its positive/negative effects are all swapped wrt Figure 2. The postfix of the plan (steps [7,13]) validates the edited action model at the observation of the plan execution illustrated at Figure 1. As explained our interest is not the edited model but the number of *edit operations* introduced to validate the given model in the input observation,  $\delta(\mathcal{M}, \mathcal{O}) = 7$  in the example of Figure 4.

### A propositional encoding for STRIPS action schemata

Given a STRIPS action schemata  $\xi$ , a propositional encoding for the *preconditions*, *negative* and *positive* effects of that schemata can be represented with fluents of the kind  $\{pre\_e\_ \xi, del\_e\_ \xi, add\_e\_ \xi\}_{\forall e \in \mathcal{I}_{\Psi, \xi}}$  such that  $e \in \mathcal{I}_{\Psi, \xi}$  is a single element from the set of FOL interpretations of predicates  $\Psi$  over the corresponding parameters  $pars(\xi)$ . We call these fluents the set of *editable fluents*. Figure 5 shows the propositional encoding for the six action schemata defined in Figure 2 using the *editable fluents*.

We use *conditional effects* to compactly define *editable actions*. Actions whose semantics is given by the value of the *editable fluents*  $\{pre\_e\_ \xi, del\_e\_ \xi, add\_e\_ \xi\}_{\forall e \in \mathcal{I}_{\Psi, \xi}}$  at the current state. Given an operator schemata  $\xi \in \mathcal{M}$  its *editable* version is formalized as:

$$\begin{aligned}
pre(editable_\xi) &= \{pre\_e\_ \xi \implies e\}_{\forall e \in \mathcal{I}_{\Psi, \xi}} \\
cond(editable_\xi) &= \{del\_e\_ \xi \triangleright \neg e\}_{\forall e \in \mathcal{I}_{\Psi, \xi}}, \\
&\quad \{add\_e\_ \xi\} \triangleright \{e\}_{\forall e \in \mathcal{I}_{\Psi, \xi}}.
\end{aligned}$$

Figure 6 shows the PDDL encoding of the *editable* `inc-x(?v1, ?v2)` schemata for robot navigation in a  $N \times N$  grid. Note that this editable schemata, when the fluents of Figure 5 hold, behaves exactly as defined in Figure 2.

### The compilation formalization

Conditional effects allow us to compactly define our compilation for computing  $\delta(\mathcal{M}, \mathcal{O})$  and hence, estimate the  $P(\mathcal{O}|\mathcal{M})$  likelihood. Given a STRIPS model  $\mathcal{M} \in \mathcal{M}$  and

```

;;; Propositional encoding for inc-x(?v1 ?v2)
(pre_xcoord_v1_inc-x) (pre_next_v1_v2_inc-x)
(pre_q0_inc-x)
(del_xcoord_v1_inc-x) (add_xcoord_v2_inc-x)

;;; Propositional encoding for dec-x(?v1 ?v2)
(pre_xcoord_v1_dec-x) (pre_next_v2_v1_dec-x)
(pre_q1_dec-x)
(del_xcoord_v1_dec-x) (add_xcoord_v2_dec-x)

;;; Propositional encoding for inc-y-even(?v1 ?v2)
(pre_ycoord_v1_inc-y-even) (pre_next_v1_v2_inc-y-even)
(pre_q0_inc-y-even)
(del_ycoord_v1_inc-y-even) (del_q0_inc-y-even)
(add_ycoord_v2_inc-y-even) (add_q1_inc-y-even)

;;; Propositional encoding for inc-y-odd(?v1 ?v2)
(pre_ycoord_v1_inc-y-odd) (pre_next_v1_v2_inc-y-odd)
(pre_q0_inc-y-odd)
(del_ycoord_v1_inc-y-odd) (del_q1_inc-y-odd)
(add_ycoord_v2_inc-y-odd) (add_q0_inc-y-odd)

;;; Propositional encoding for dec-y-even(v1 ?v2)
(pre_ycoord_v1_dec-y-even) (pre_next_v2_v1_dec-y-even)
(pre_q1_dec-y-even)
(del_ycoord_v1_dec-y-even) (del_q0_dec-y-even)
(add_ycoord_v2_dec-y-even) (add_q1_dec-y-even)

;;; Propositional encoding for dec-y-odd(?v1 ?v2)
(pre_ycoord_v1_dec-y-odd) (pre_next_v2_v1_dec-y-odd)
(pre_q1_dec-y-odd)
(del_ycoord_v1_dec-y-odd) (del_q1_dec-y-odd)
(add_ycoord_v2_dec-y-odd) (add_q0_dec-y-odd)

```

Figure 5: Propositional encoding for the six schemata from Figure 2.

the observation  $\mathcal{O}(\tau)$  of the execution of a plan for solving  $P = \langle F, A, I, G \rangle$ , our compilation outputs a classical planning task with conditional effects  $P' = \langle F', A', I', G' \rangle$ :

- $F'$  extends the original fluents  $F$  with:
  - The set of *editable fluents*  $\{pre\_e\_ \xi, del\_e\_ \xi, add\_e\_ \xi\}_{\forall e \in \mathcal{I}_{\Psi, \xi}}$  to model the possible STRIPS schemata.
  - The fluents to code the *observation constraints*:  $F_\pi = \{plan(a_i, i)\}_{1 \leq i \leq l}$  to code the  $i^{th}$  action in  $\mathcal{O}(\tau)$ . The static facts  $next_{i, i+1}$  and the fluents  $at_i$ ,  $1 \leq i < l$ , are also added to iterate through the  $l$  actions in  $\mathcal{O}(\tau)$ . The fluents  $\{test_j\}_{1 \leq j \leq m}$ , indicating the state observation  $s_j \in \mathcal{O}(\tau)$  where the action model is validated.
  - The fluents  $mode_{edit}$  and  $mode_{val}$  to indicate whether the operator schemata are edited or validated.
- $I'$  extends the original initial state  $I$  with the fluent  $mode_{edit}$  set to true as well as the fluents  $F_\pi$  plus fluents  $at_1$  and  $\{next_{i, i+1}\}$ ,  $1 \leq i < l$ , for tracking the plan step where the action model is validated. Our compilation assumes that initially  $\mathcal{M}'$  is defined as  $\mathcal{M}$ . Therefore the *editable fluents*  $\{pre\_e\_ \xi, del\_e\_ \xi, add\_e\_ \xi\}_{\forall e \in \mathcal{I}_{\Psi, \xi}}$  hold as given by  $\mathcal{M}$ .
- $G' = G \cup \{at_n, test_m\}$ .
- $A'$  comprises three kinds of actions with conditional effects:
  1. The *editable* version of the original actions given by  $\mathcal{M}$ . This actions have now an extra precondition because they can only be applied in the *validation* mode (i.e. when  $mode_{val}$  holds). When the observation  $\mathcal{O}(\tau)$  includes observed actions, they also include the extra conditional effects

```

(:action editable_inc-x
:parameters (?v1 ?v2)
:precondition
  (and (or (not (pre_xcoord_v1_inc-x)) (xcoord ?v1))
        (or (not (pre_xcoord_v2_inc-x)) (xcoord ?v2))
        (or (not (pre_ycoord_v1_inc-x)) (xcoord ?v1))
        (or (not (pre_ycoord_v2_inc-x)) (xcoord ?v2))
        (or (not (pre_q0_inc-x)) (q0))
        (or (not (pre_q1_inc-x)) (q1))
        (or (not (pre_next_v1_v1_inc-x)) (next ?v1 ?v1))
        (or (not (pre_next_v1_v2_inc-x)) (next ?v1 ?v2))
        (or (not (pre_next_v2_v1_inc-x)) (next ?v2 ?v1))
        (or (not (pre_next_v2_v2_inc-x)) (next ?v2 ?v2)))
:effect (and
  (when (del_xcoord_v1_inc-x) (not (xcoord ?v1)))
  (when (del_xcoord_v2_inc-x) (not (xcoord ?v2)))
  (when (del_ycoord_v1_inc-x) (not (xcoord ?v1)))
  (when (del_ycoord_v2_inc-x) (not (xcoord ?v2)))
  (when (del_q0_inc-x) (not (q0)))
  (when (del_q1_inc-x) (not (q1)))
  (when (del_next_v1_v1_inc-x) (not (next ?v1 ?v1)))
  (when (del_next_v1_v2_inc-x) (not (next ?v1 ?v2)))
  (when (del_next_v2_v1_inc-x) (not (next ?v2 ?v1)))
  (when (del_next_v2_v2_inc-x) (not (next ?v2 ?v2)))

  (when (add_xcoord_v1_inc-x) (xcoord ?v1))
  (when (add_xcoord_v2_inc-x) (xcoord ?v2))
  (when (add_ycoord_v1_inc-x) (xcoord ?v1))
  (when (add_ycoord_v2_inc-x) (xcoord ?v2))
  (when (add_q0_inc-x) (q0))
  (when (add_q1_inc-x) (q1))
  (when (add_next_v1_v1_inc-x) (next ?v1 ?v1))
  (when (add_next_v1_v2_inc-x) (next ?v1 ?v2))
  (when (add_next_v2_v1_inc-x) (next ?v2 ?v1))
  (when (add_next_v2_v2_inc-x) (next ?v2 ?v2)))
)

```

Figure 6: Editable version of the `inc-x(?v1, ?v2)` schemata for robot navigation in a  $N \times N$  grid.

$\{at_i, plan(a_i, i)\} \triangleright \{\neg at_i, at_{i+1}\}_{\forall i \in [1, l]}$  to validate that actions are applied, exclusively, in the same order as they appear in  $\mathcal{O}(\tau)$ .

2. Actions for *editing* operator schemata  $\xi \in \mathcal{M}$ . This includes the actions for *inserting* a new *precondition* into an action schemata  $\xi \in \mathcal{M}$  and for inserting a new *negative* or *positive* effect into the action schemata  $\xi \in \mathcal{M}$

$$\begin{aligned}
\text{pre}(\text{insertPre}_{e,\xi}) &= \{\neg \text{pre}_{e,\xi}, \neg \text{del}_{e,\xi}, \\
&\quad \neg \text{add}_{e,\xi}, \text{mode}_{edit}\}, \\
\text{cond}(\text{insertPre}_{e,\xi}) &= \{\emptyset\} \triangleright \{\text{pre}_{e,\xi}\}.
\end{aligned}$$

$$\begin{aligned}
\text{pre}(\text{insertEff}_{e,\xi}) &= \{\neg \text{del}_{e,\xi}, \neg \text{add}_{e,\xi}, \text{mode}_{edit}\}, \\
\text{cond}(\text{insertEff}_{e,\xi}) &= \{\text{pre}_{e,\xi}\} \triangleright \{\text{del}_{e,\xi}\}, \\
&\quad \{\neg \text{pre}_{e,\xi}\} \triangleright \{\text{add}_{e,\xi}\}.
\end{aligned}$$

Besides these actions,  $A'$  also contains the actions for *deleting* a *precondition* and a *negative/positive effect*.

3. Actions for *validating* the edited models at the  $s_j$  observed states,  $0 \leq j < m$ .

$$\begin{aligned}
\text{pre}(\text{validate}_j) &= s_j \cup \{\text{test}_{j-1}\}, \\
\text{cond}(\text{validate}_j) &= \{\emptyset\} \triangleright \{\neg \text{test}_{j-1}, \text{test}_j, \\
&\quad \{\text{mode}_{edit}\} \triangleright \{\neg \text{mode}_{edit}, \text{mode}_{val}\}.
\end{aligned}$$

## Evaluation

To evaluate the empirical performance of *model recognition as planning* we collect a set  $M$  of possible STRIPS models,

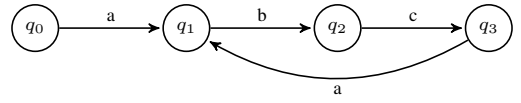


Figure 7: Four-symbol four-state *regular automata* for recognizing the  $\{(abc)^n : n \geq 1\}$  language ( $q_3$  is the acceptor state).

that share the same state variables but define different action models. Then, we randomly choose one of these models  $\mathcal{M} \in M$  and use it to produce an observation  $\mathcal{O}(\tau)$  of a plan execution. Finally, we follow our *model recognition as planning* method to identify the model  $\mathcal{M} \in M$  that produced  $\mathcal{O}(\tau)$ . The experiment is repeated for models of different kind and different observability of the given plan execution.

**Reproducibility.** MADAGASCAR is the classical planner we used to solve the instances that result from our compilations for its ability to deal with dead-ends (Rintanen 2014). Due to its SAT-based nature, MADAGASCAR can apply the actions for editing preconditions in a single planning step (in parallel) because there is no interaction between them. Actions for editing effects can also be applied in a single planning step, thus significantly reducing the planning horizon.

The compilation source code, evaluation scripts and benchmarks (including the used training and test sets) are fully available at this anonymous repository so any experimental data reported in the paper can be reproduced.

**Recognition of Regular Automata.** In this experiment the models in  $M$  represent different *regular automata*. Figure 7 illustrate a four-symbol four-state *regular automata* for recognizing the  $\{(abc)^n : n \geq 1\}$  language. The *input alphabet* is  $\Sigma = \{a, b, c, \square\}$ , and the machine states are  $Q = \{q_0, q_1, q_2, q_3\}$  (where  $q_3$  is the only acceptor state).

We generated a  $M$  set of different STRIPS models that encode different *regular automata*. Each  $\mathcal{M} \in M$  encodes a different *regular automata*. Each automata transition is encoded with a planning action. For instance, the execution of the *regular automata* defined in Figure 7, with the sequence of input symbols  $abcabc$ , produces the following six-action plan  $(a, q_0 \rightarrow q_1), (b, q_1 \rightarrow q_2), (c, q_2 \rightarrow q_3), (a, q_3 \rightarrow q_1), (b, q_1 \rightarrow q_2), (c, q_2 \rightarrow q_3)$ .

Here we assume that the actual applied transitions are unknown as well as the internal machine state. Assuming that the actual applied transitions is unknown means that the observation  $\mathcal{O}(\tau)$  of the execution of a regular automata contains no actions, it is simply a sequence of states  $\mathcal{O}(\tau) = \langle s_1, \dots, s_m \rangle$ . Assuming that the internal machine state is unknown means that  $\mathcal{O}(\tau)$  is a  $\Phi$ -observation and that the  $\Phi$  subset does not contain  $(q)$  fluents, with  $q \in Q$  and  $q \neq q_0$ .

**Recognition of navigation models.** In this experiment the given models in  $M$  represent different navigation models that are computed as the cross product of a regular automata with a four-operator STRIPS model for navigating a  $N \times N$  grid.

In this case the regular automata constrain the applications of the navigation actions producing different naviga-

tion policies e.g. like the one in Figure 2. Given an observation of a plan execution, like the one illustrated at Figure 1, here the task is to identify which navigation model produced that observation, despite the the applied actions are unobserved. In addition, for each observed state, only the value of fluents encoding the x and y coordinates of the agent are known while the value of the regular automata conditioning the navigation policy is unknown.

## Results

## Conclusions

This paper formalized the *model recognition* task and proposed, *model recognition as planning*, a method built on top of off-the-shelf classical planning algorithms to estimate the probability of a STRIPS model to produce a partial observation of a plan execution. The paper shows the effectiveness of *model recognition as planning* in a set of STRIPS models encoding different kinds of *automata*. *Model recognition as planning* succeeds to identify the executed automata despite the internal machine state or actual applied transitions, are unobserved.

Previous work on the learning of STRIPS action models also defined semantic error metrics to quantify the errors of a model with respect to the observation of a plan execution (Yang, Wu, and Jiang 2007). Our approach for quantifying this error is based on the definition of a edit distance for the model which allow us to not accumulate the repetition of errors coming from the the same model flaw. A related approach is recently followed for *model reconciliation* (Chakraborti et al. 2017) where model edition is used to conform the PDDL models of two different agents. Also related to this paper is the work on *Goal Recognition Design* but in that work the action model is given in advance and fixed (Keren, Gal, and Karpas 2014).

## References

- Aineto, D.; Jiménez, S.; and Onaindia, E. 2018. Learning strips action models with classical planning.
- Baier, J. A.; Fritz, C.; and McIlraith, S. A. 2007. Exploiting procedural domain control knowledge in state-of-the-art planners. In *ICAPS*, 26–33.
- Bonet, B.; Palacios, H.; and Geffner, H. 2010. Automatic derivation of finite-state machines for behavior control. In *AAAI Conference on Artificial Intelligence*.
- Bunke, H. 1997. On a relation between graph edit distance and maximum common subgraph. *Pattern Recognition Letters* 18(8):689–694.
- Carberry, S. 2001. Techniques for plan recognition. *User Modeling and User-Adapted Interaction* 11(1-2):31–48.
- Chakraborti, T.; Sreedharan, S.; Zhang, Y.; and Kambhampati, S. 2017. Plan explanations as model reconciliation: Moving beyond explanation as soliloquy. In *IJCAI*.
- Dennett, D. C. 1983. Intentional systems in cognitive ethology: The panglossian paradigm defended. *Behavioral and Brain Sciences* 6(3):343–355.
- Devroye, L.; Györfi, L.; and Lugosi, G. 2013. *A probabilistic theory of pattern recognition*, volume 31. Springer Science & Business Media.
- Fox, M., and Long, D. 2003. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research* 20:61–124.
- Geffner, H., and Bonet, B. 2013. A concise introduction to models and methods for automated planning.
- Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated Planning: theory and practice*. Elsevier.
- Ivankovic, F., and Haslum, P. 2015. Optimal planning with axioms. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*.
- Keren, S.; Gal, A.; and Karpas, E. 2014. Goal recognition design. In *ICAPS*.
- Lesh, N., and Etzioni, O. 1995. A sound and fast goal recognizer. In *IJCAI*, volume 95, 1704–1710.
- Masek, W. J., and Paterson, M. S. 1980. A faster algorithm computing string edit distances. *J. Comput. Syst. Sci.* 20(1):18–31.
- Patrizi, F.; Lipovetzky, N.; De Giacomo, G.; and Geffner, H. 2011. Computing infinite plans for ltl goals using a classical planner. In *IJCAI*, 2003–2008.
- Ramírez, M., and Geffner, H. 2009. Plan recognition as planning. In *International Joint conference on Artificial Intelligence*, 1778–1783.
- Ramírez, M. 2012. *Plan recognition as planning*. Ph.D. Dissertation, Universitat Pompeu Fabra.
- Rintanen, J. 2014. Madagascar: Scalable planning with sat. *Proceedings of the 8th International Planning Competition (IPC-2014)*.
- Segovia-Aguas, J.; Jiménez, S.; and Jonsson, A. 2017. Generating context-free grammars using classical planning. In *International Joint Conference on Artificial Intelligence, ICAPS-17*.
- Slaney, J., and Thiébaux, S. 2001. Blocks world revisited. *Artificial Intelligence* 125(1-2):119–153.
- Sohrabi, S.; Riabov, A. V.; and Udrea, O. 2016. Plan recognition as planning revisited. In *IJCAI*, 3258–3264.
- Sukthankar, G.; Geib, C.; Bui, H. H.; Pynadath, D.; and Goldman, R. P. 2014. *Plan, activity, and intent recognition: Theory and practice*. Newnes.
- Wilcox, R. R. 1981. A review of the beta-binomial model and its extensions. *Journal of Educational Statistics* 6(1):3–32.
- Yang, Q.; Wu, K.; and Jiang, Y. 2007. Learning action models from plan examples using weighted max-sat. *Artificial Intelligence* 171(2-3):107–143.