Online Action Recognition

Alejandro Suárez-Hernández[1], **Javier Segovia-Aguas**[1,2], **Carme Torras**[1], **Guillem Alenyà**[1]
[1]IRI - Institut de Robòtica i Informàtica Industrial, CSIC-UPC
[2]Universitat Pompeu Fabra
asuarez@iri.upc.edu, javier.segovia@upf.edu, torras@iri.upc.edu, galenya@iri.upc.edu

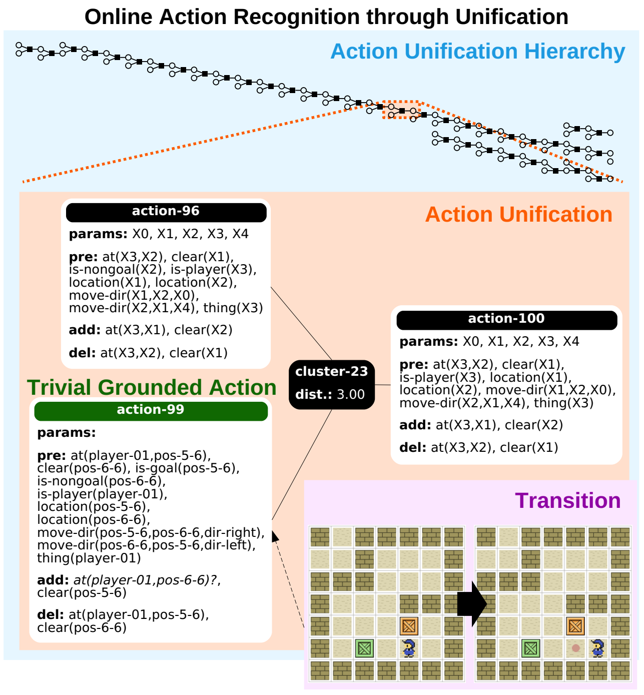
Abstract

Recognition in planning seeks to find agent intentions, goals or activities given a set of observations and a knowledge library (e.g. goal states, plans or domain theories). In this work we introduce the problem of *Online Action Recognition*. It consists in recognizing, in an *open world*, the planning action that *best* explains a partially observable state transition from a knowledge library of first-order STRIPS actions, which is initially empty. We frame this as an optimization problem, and propose two algorithms to address it: *Action Unification* (AU) and *Online Action Recognition through Unification* (OARU). The former builds on *logic unification* and generalizes two input actions using *weighted partial MaxSAT*. The latter looks for an action within the library that explains an observed transition. If there is such action, it generalizes it making use of AU, building in this way an AU hierarchy. Otherwise, OARU inserts a *Trivial Grounded Action* (TGA) in the library that explains just that transition. We report results on benchmarks from the International Planning Competition and PDDLGym, where OARU recognizes actions accurately with respect to expert knowledge, and shows real-time performance.

Introduction

The prediction of the most likely actions, plans or goals of an agent, has been a topic of interest in the planning community since the work by Ramírez and Geffner (2009), which posed the recognition task via planning given a domain theory and a set of observations. Other work adopted this convention of recognition as planning for related tasks, i.e. goal recognition and environment design (Keren, Gal, and Karpas 2014), production and recognition of context-free grammars (Segovia-Aguas, Jiménez, and Jonsson 2017a), classification of planning instances in generalized plans (Segovia-Aguas, Jiménez, and Jonsson 2017b), counter-planning (Pozanco et al. 2018), model recognition (Aineto et al. 2019) and recognition with noisy observations (Sohrabi, Riabov, and Udrea 2016; Aineto, Jimenez, and Onaindia 2020).

In this paper, we address the problem of *Online Action Recognition* (OAR), which consists in recognizing the actions performed by an agent in an *open-world*, given a *state transition* and a *knowledge library* of first-order STRIPS actions (also known as domain theory). We restrict OAR to


Online Action Recognition through Unification
Action Unification Hierarchy
Action Unification
Trivial Grounded Action
cluster-23
dist.: 3.00
Transition

action-96
params: X0, X1, X2, X3, X4
pre: at(X3,X2), clear(X1), is-nongoal(X2), is-player(X3), location(X1), location(X2), move-dir(X1,X2,X0), move-dir(X2,X1,X4), thing(X3)
add: at(X3,X1), clear(X2)
del: at(X3,X2), clear(X1)

action-100
params: X0, X1, X2, X3, X4
pre: at(X3,X2), clear(X1), is-player(X3), location(X1), location(X2), move-dir(X1,X2,X0), move-dir(X2,X1,X4), thing(X3)
add: at(X3,X1), clear(X2)
del: at(X3,X2), clear(X1)

action-99
params:
pre: at(player-01,pos-5-6), clear(pos-6-6), is-goal(pos-5-6), is-nongoal(pos-6-6), is-player(player-01), location(pos-5-6), location(pos-6-6), move-dir(pos-5-6,pos-6-6,dir-right), move-dir(pos-6-6,pos-5-6,dir-left), thing(player-01)
add: *at(player-01,pos-6-6)?*, clear(pos-5-6)
del: at(player-01,pos-5-6), clear(pos-6-6)

Figure 1: Illustration of AU and OARU in *sokoban*.

symbolic inputs and deterministic action effects. However, we set an initially empty knowledge library that must be inductively filled from partially observable transitions. We describe an algorithm called *Action Unification* (AU), related to *syntactic logic unification* (Snyder 2001). AU generalizes two actions using an encoding to *Weighted Partial MaxSAT* (WPMS). We also propose *Online Action Recognition through Unification* (OARU), an algorithm that makes repeated use of AU to merge ad-hoc explanations of the transitions into its action library. In the following, we use *action* as a generic category; *schema* or *model* for parameterized actions; *grounded action* for parameter-less actions; and *Trivial Grounded Action* (TGA) for ad-hoc actions derived directly from observations.

Example 1. *Figure 1 uses sokoban (Junghanns and Schaeffer 1997) to illustrate AU and OARU. The whole AU hierarchy, which summarizes the history of merged actions, is* *in the top. We zoom in one part, showing: (1)* `action-96`*, a schema already present in OARU's library; (2) a TGA* `action-99`*, constructed from the transition depicted graphically at the bottom; and (3) a 5-ary schema* `action-100` *produced by AU, which generalizes the previous two actions. The transition corresponds to the character moving to the right. AU has determined that the* distance *between* `action-96` *and* `action-99` *is 3. Later, we will see that this distance is calculated in terms of relaxed precondition predicates. Some predicates have a question mark indicating that they are uncertain due to the open world setting. We will also see how this uncertainty can be dispelled through AU. Overall, the example shows how OARU surmised from the past that the player could not move from a goal cell, and corrects this assumption when shown otherwise.*

The contributions of this work are: (1) a formal method for generalizing two planning actions with the AU algorithm, which deals with an *NP-Hard* problem; (2) a scalable, accurate and suitable for real-time usage algorithm to recognize and acquire general schemata from partially observable state transitions, named OARU; and (3) an evaluation of acquired knowledge libraries with expert handcrafted benchmarks from the International Planning Competition (IPC) (Muise 2016) and PDDLGym (Silver and Chitnis 2020).

Related Work

The problem of OAR relates to plan, activity and intent recognition (PAIR) (Sukthankar et al. 2014) and learning action models (Arora et al. 2018). The recognition in PAIR is defined as a prediction task of the most plausible future, i.e. the most probable plan or goal an agent will pursue (Ramírez and Geffner 2009; Ramírez and Geffner 2010), while our recognition task serves as an explanation of the past (Chakraborti et al. 2017; Aineto et al. 2019; Aineto, Jimenez, and Onaindia 2020). Also, model-based approaches for recognition assume a knowledge library with goals, plans or domain theories is known beforehand. We release the problem from this assumption, where the knowledge is acquired in an incremental online fashion.

We refer to learning the representation of the world dynamics as the problem of learning action models, which has been a topic of interest for long time (Gil 1994; Benson 1995; Wang 1995). STRIPS-like actions have been learned with algorithms such as ARMS (Yang, Wu, and Jiang 2007) with a weighted MaxSAT; SLAF (Amir and Chang 2008) with an online SAT solver that computes the CNF formula compatible with partial observations; LOCM (Cresswell, McCluskey, and West 2009; Cresswell and Gregory 2011; Gregory and Cresswell 2015) computing finite state machines of object sorts; FAMA (Aineto, Jiménez, and Onaindia 2018; Aineto, Celorrio, and Onaindia 2019) learning from minimal observability with an off-the-shelf classical planner; and Representation Discovery (Bonet and Geffner 2020) that learns with a SAT solver from plain graphs. In the latter, only action labels are known, while in the rest of approaches, the name and parameters of each declarative action are known, which strictly constraints the learning problem. In our case, no prior labels and parameters bound the problem, like in Suárez-Hernández et al. (2020), but inductively learned with every new partially observable state transition.

The work by Amado et al. (2018) proposes the LatPlan algorithm for goal recognition. It first learns grounded action models in the latent space from images, such as AMA2 (Asai and Fukunaga 2018). However, grounded actions are generated without soundness guarantees, may not generalize to other instances, and the learning method requires observing all possible non-symbolic transitions, while OARU inputs are symbolic, computes actions which do not require to observe all state transitions and generalize to other instances.

Preliminaries

Let us first introduce preliminary notation on first-order logic and unification, and weighted partial MaxSAT.

First-Order Logic

The formal language to describe relations between constant symbols is known as *first-order logic* or *predicate logic*. *Propositional logic* is subsumed by *first-order logic* in that propositions are specific interpretations of relations over constant symbols. The **syntax** of a first-order language consists of a (possibly) infinite set of *logical terms* (mathematical objects) and *well-formed formulae* (mathematical facts) denoted as T and $\mathcal{F}$, respectively.

Definition 1 (First-Order Logical Term). *A* first-order logical term *$t \in T$ is recursively defined as $t = c|v|f_n(t_1, \dots, t_n)$ where t is either a constant symbol $c \in \mathcal{C}$, a variable symbol $v \in \mathcal{V}$, or a functional symbol f_n with n arguments s.t. $t_i \in T$ for all $1 \leq i \leq n$.*

Definition 2 (Well-Formed Formula). *A* well-formed formula *(wff) $\varphi \in \mathcal{F}$ is inductively defined as a* predicate *(or atomic formula) $\varphi = p_n(t_1, \dots, t_n)$; a* negation *$\neg\varphi$; a* logical connective *conjunction $\varphi \wedge \psi$, disjunction $\varphi \vee \psi$, implication $\varphi \rightarrow \psi$, or biconditional $\varphi \leftrightarrow \psi$ s.t. ψ is also a wff; or a* quantifier *$\forall_v \varphi$ or $\exists_v \varphi$, for universal and existential quantification of a wff φ and variable $v \in \mathcal{V}$, respectively.*

The first-order **semantics** is a structure which consists of an interpretation function $\mathcal{I}$ and a *universe* $\mathcal{D}$ with the nonempty set of all objects. The set of objects is used to ascribe meaning to terms and formulae with an interpretation function but, while terms are interpreted into objects, predicates and other formulae have boolean interpretations. The *interpretation over a logical term* $\mathcal{I}(t)$ is an assigned function to term t which maps a tuple of already interpreted arguments of n objects in $\mathcal{D}^n$ to a single object in $\mathcal{D}$. Hence, the interpretation of constant terms is $\mathcal{I}(c) : \mathcal{D}^0 \rightarrow \mathcal{D}$, from variables is $\mathcal{I}(v) : \mathcal{D}^1 \rightarrow \mathcal{D}$, and functional symbols is $\mathcal{I}(f) : \mathcal{D}^n \rightarrow \mathcal{D}$, which maps 0, 1 and n objects into one, respectively. The interpretation of an n-ary predicate is $\mathcal{I}(p) : \mathcal{D}^n \rightarrow \{\text{true}, \text{false}\}$. The *interpretation over a well-formed formula* $\mathcal{I}(\varphi)$ consists of: (1) the interpretation of variables and functionals given a set of objects $\mathcal{D}$; and (2) the boolean evaluation of predicate interpretations, and logical connectives. In this paper we only focus on constant, variables and first-order predicate symbols such as follows:

Example 2. *In Figure 1, there are 7 predicate symbols* $\{\texttt{at}_2$, $\texttt{clear}_1$, $\texttt{is-player}_1$, $\texttt{location}_1$, $\texttt{move-dir}_3$, $\texttt{thing}_1\}$, *5 variables* $\{X0, X1, X2, X3, X4\}$ *and 5 constants* {`player-01`, `pos-5-6`, `pos-6-6`, `dir-left`, `dir-right`}. *In the first state shown, the player is in position* `pos-5-6`, *so* $\mathcal{I}(\texttt{at}_2(\textit{player-01}, \textit{pos-5-6})) = \text{true}$. *In the second one, it is no longer there, so the same interpretation is false. Predicates with variable arguments such as* $\texttt{at}(X1, X0)$ *require an interpretation of variables* $X1$ *and* $X0$ *before evaluation (known as grounding), e.g.* $\{\mathcal{I}(X0) = \textit{player-01}, \mathcal{I}(X1) = \textit{pos-5-6}\}$.

First-Order Unification

The problem of first-order unification (*unification* for short) consists in making both sides of a set of equations syntactically equal (Snyder 2001). Then, the unification problem is defined as a set of potential equations $\mathcal{U} = \{l_1 \doteq r_1, \ldots, l_n \doteq r_n\}$ where each left and right-hand sides are first-order logical terms $l_i, r_i \in T$.

A *substitution* $\sigma : \mathcal{V} \rightarrow T$ is a function that maps variables into terms, and its notation is $\{v_1 \rightarrow t_1, \ldots, v_n \rightarrow t_n\}$ where each $v_i \in \mathcal{V}$ and $t_i \in T$. An *instance* of a term t is $t\sigma$ and expanded as $t\{v_1 \rightarrow t_1, \ldots, v_n \rightarrow t_n\}$, where all variable substitutions are applied simultaneously. Two terms l and r of a potential equation $l \doteq r$ are syntactically equal if exists a substitution σ such that $l\sigma \equiv r\sigma$, where $\equiv$ stands for syntactically equivalent.

Example 3 (Substitution)**.** *Given a pair of constants* $player, loc \in \mathcal{C}$ *and variables* $v_1, v_2 \in \mathcal{V}$, *the potential equation* $\texttt{at}(v_1, \texttt{loc}) \doteq \texttt{at}(\texttt{player}, v_2)$ *is syntactically equivalent when the substitution is* $\sigma = \{v_1 \rightarrow \texttt{player}, v_2 \rightarrow \texttt{loc}\}$.

The substitution of terms is also named a *unifier*. A *solution* to $\mathcal{U}$ is a *unifier* σ if $l_i\sigma \equiv r_i\sigma$ for all $1 \leq i \leq n$. In the context of this paper, we are interested only in the unification of predicates without nested arguments (i.e. no functions).

Weighted Partial MaxSAT (WPMS)

A *literal* l is a boolean variable $l = x$ or its negation $l = \neg x$. A *clause* c is a disjunction of literals, and a *weighted clause* (c, w) extends the clause c with a natural number $w \in \mathbb{N}$ (or ∞) which represents the cost of not satisfying (c, w). In WPMS, a weighted clause is either *hard* if the weight is infinite, and *soft* otherwise.

In contrast to (partial Max) SAT where the input formula is described in conjunctive normal form (conjunctions of clauses), the input to a WPMS problem is a set of weighted clauses $\Phi = \{(c_1, w_1), \ldots, (c_n, w_n)\}$, which may be divided into $\Phi_s = \{(c_1, w_1), \ldots, (c_m, w_m)\}$ and $\Phi_h = \{(c_{m+1}, \infty), \ldots, (c_n, \infty)\}$, for soft and hard weighted clause sets, respectively. Then, $\Phi = \Phi_s \cup \Phi_h$.

Let $vars(\Phi)$ be the set of all boolean variables in Φ. A *truth assignment* is the interpretation of boolean variables into 0 or 1, i.e. $\mathcal{I} : vars(\Phi) \rightarrow \{0, 1\}$. Then, the truth assignment for a set of weighted clauses Φ is:

$$\mathcal{I}(\Phi) = \sum_{i=1}^{m} w_i(1 - \mathcal{I}(c_i)). \quad (1)$$

Therefore, WPMS is the problem of finding an *optimal assignment* $\mathcal{I}^*(\Phi)$ (Ansótegui and Gabas 2013), such that the hard clauses are satisfied and Equation 1 is minimized:

$$\mathcal{I}^*(\Phi) = min\{\mathcal{I}(\Phi) | \mathcal{I} : vars(\Phi) \rightarrow \{0, 1\}\}. \quad (2)$$

The optimal assignment of Equation 2 could be inferred from $\mathcal{I}^*(vars(\Phi))$. Also, note that Equation 1 only iterates over soft clauses, since falsifying a hard clause would result in infinite cost and the solution would be invalid.

STRIPS Action Model

This work aims at recognizing the action models $\mathcal{A}$ in a domain represented with the STRIPS fragment of the Planning Domain Definition Language (PDDL) (McDermott et al. 1998; Haslum et al. 2019) and negations. An *action schema* $\alpha \in \mathcal{A}$ is defined by a 3-tuple $\alpha = \langle head_\alpha, pre_\alpha, \mathit{eff}_\alpha \rangle$ where:

- $head_\alpha$ is the name of the action and a set of variables $\mathcal{V}' \subseteq \mathcal{V}$ referenced in pre_α and eff_α,
- pre_α is a *well-formed formula* with a conjunction of literals that represent the action preconditions,
- eff_α is the list of effects which consists of literals that are updated either to true (add/positive effect) or false (delete/negative effect). Often, eff_α is splitted into add_α and del_α lists.

Online Action Recognition

The problem of *Action Recognition* is a trivial task under certain assumptions such as fully observable transitions, finite set of relations and objects that are known to be true (*closed-world* assumption), and a complete knowledge library of STRIPS action models to represent the possible interactions between an agent and the world. From an observer perspective, the action applied by an agent would be easily identified with a naive linear algorithm that loops over the set of grounded actions. Nevertheless, the problem becomes more complex when the knowledge library only contains *action signatures* (set of action symbols with their arity) and transitions are partially observable. Then, the action models must be learned (Amir and Chang 2008; Mourao et al. 2012; Aineto, Jiménez, and Onaindia 2018). The problem is even harder when the knowledge library starts empty and the interpretation of the each relation between objects is unknown beforehand (*open-world* assumption) (Minker 1982). We focus on the online version of the latter paradigm.

Input: Set of Observations

Inputs consist of observations over a set of transitions. Implicitly, each transition is a (s, a, s') triplet, where that s is the previous state, a is the action applied by an agent, and s' is the successor state. However, in OAR, the action applied by the agent is never observed, so each transition observation is $o = (s, s')$. When the input consists of a set of observations $\mathcal{O}$, each observation $o \in \mathcal{O}$ is sequentially processed following the order in which they were observed.

We use a *factored state* representation where each state s is defined by a set of predicates $p_n(d_1, \ldots, d_n)$ (or their negation), such that p_n is a predicate symbol of arity n, and every $d_i \in \mathcal{D}$ for $1 \leq i \leq n$ is an object in the universe.

In an *open world*, a grounded predicate $p_n(d_1, \ldots, d_n)$ is known to be true if each $d_i \in \mathcal{C}$ and it is explicitly true in a given state s (resp. $\neg p_n(d_1, \ldots, d_n)$). Therefore, a state s is *fully observable* if for every predicate and constants, there is an interpretation $\mathcal{I}_s(p_n(d_1, \ldots, d_n))$ which maps the grounded predicate either to true or false. Thus, a state s is *partially observable* if there is no such interpretation $\mathcal{I}_s$ for all known predicates and constants.

Next, we introduce some definitions over $\varphi = p_n(d_1, \ldots, d_n)$ that will become handy:

- A function $\text{type}(\varphi) := p_n$.
- A function $\text{arg}_i(\varphi) := d_i$.
- A function $\iota_s(\varphi) := \text{true}$ iff $\mathcal{I}_s(\varphi)$ is defined, else false.

Output: First-Order STRIPS Action Model

The output for each observation $o \in \mathcal{O}$ is a grounded action a composed of the action model $\alpha \in \mathcal{A}$ and its substitution $\sigma \in \Sigma$. We denote as $\mathcal{D}_\alpha$ the set of objects (either constants or variables) referenced within α's effects and preconditions. An action a is said to be grounded or instantiated when the variables in $head_\alpha$ are substituted by a tuple of constants from $\mathcal{D}$ of equal size. We denote this by $\alpha\sigma$, where $\sigma \in \Sigma$ is an object substitution (and thus Σ is the set of all possible substitutions), as those introduced earlier. This nomenclature extends naturally to the substitution of arbitrary objects, not just variables. We adopt the convention that if σ does not define an explicit substitution for an object $d \in \mathcal{D}_\alpha$, then its application leaves all references to d unchanged.

A **solution** to the problem is then a *grounded action* a, such that $a = \alpha\sigma$ where $\alpha \in \mathcal{A}$ and $\sigma \in \Sigma$, which completes the observation (s, s') into (s, a, s') and the following conditions hold:

Condition 1 (Valid Preconditions)**.** *The interpretation evaluates the preconditions to true, i.e.* $\mathcal{I}_s(pre_a) = \top$*, in other words, the action preconditions hold in the previous state, i.e.* $pre_a \subseteq s$*.*

Condition 2 (Valid Transition)**.** *The successor state* s' *is a direct consequence from applying* a *in* s*, such that* $s' = (s \setminus del_a) \cup add_a$*.*

Problem Statement

The OAR problem is tabular in that background theories, such as schemata $\mathcal{A}$, are unknown. Indeed, the problem is twofold, where the action is recognized if exists in the knowledge library, otherwise learns a new action model (Arora et al. 2018) and unifies it with previous knowledge.

Definition 3 (Online Action Recognition)**.** *The OAR problem consists in finding a function* $\mathcal{P} : \mathcal{O} \rightarrow \mathcal{A} \times \Sigma$ *that sequentially maps each observation* $o \in \mathcal{O}$ *into an action* $\alpha \in \mathcal{A}$*, and a substitution* $\sigma \in \Sigma$ *s.t. Conditions 1 and 2 hold. If* $\alpha \notin \mathcal{A}$ *initially, a new* α *must be learned.*

Methodology

The main algorithm is OARU. It starts with the set of predicate symbols but neither universe nor action library knowledge are known (i.e. $\mathcal{D} = \emptyset$ and $\mathcal{A} = \emptyset$). Both are greedily learned with every new observation $o = (s, s')$ as follows:

1. For each pair of consecutive states s and s', it generates a Trivial Grounded Action (TGA) a that explains this transition, but does not generalize.
2. OARU merges a with the *closest* action in $\alpha \in \mathcal{A}$ via Action Unification (AU) using a weighted partial MaxSAT, which results in a new action α' and a grounding σ.
3. OARU recognizes grounded action $a' = \alpha'\sigma$ as the underlying action that produced observation o.

Construction of Trivial Grounded Actions

A TGA is a grounded STRIPS action a that explains just one transition from s to s'. Hence, s' can be reached if a is applied to s, but does not generalized to other states. This may be also denoted by $s \xrightarrow{a} s'$. To construct a in a full observability setting, we set the conjunction of the predicates in s as the precondition, i.e. $pre_a = s$. The effects are defined as an add list $add_a = (s' \setminus s)$, and a delete list $del_a = (s \setminus s')$. To extend this basic construction to support partial observability, we consider uncertain predicates in s and s' as potentially true and false, and mark their occurrences within pre_a, add_a and del_a and effect as uncertain[1].

The preconditions and effects of any action can be joined into a single set L_a of *labeled predicates*, each defined by a triplet $\chi = (\varphi, l, k)$, where:

- $\varphi = p_n(d_1, \ldots, d_n)$ is a predicate as defined earlier,
- $l \in \{\text{PRE}, \text{ADD}, \text{DEL}\}$ is the label to denote either the precondition, add or delete list,
- k is either true if the predicate is known to belong to l with total certainty, or false if it is unknown.

Notice that L_a is sufficient to represent a, since the parameters of a can be constructed as the union of all the variables appearing in L_a (i.e. all variables in $\mathcal{D}_a$).

The k flag of a labeled predicate is set depending on whether φ is known in s and s'.

Example 4. *Consider sokoban, and a predicate* $\varphi = \texttt{at}(\texttt{agent}, \texttt{pos-1-1})$*. Suppose* $\mathcal{I}_s(\varphi) = \text{true}$*, but* $\iota_{s'}(\varphi) = \text{false}$ *(i.e. the location of the agent is known to be (1,1) in s, but this is not known with certainty in* s'*). Then, the created TGA contains a labeled predicate* $\chi = (\varphi, \text{DEL}, \text{false})$ *because it is unknown whether* φ *is removed in the transition.*

Action Unification with Weighted Partial MaxSAT

Let us focus on Action Unification (AU), OARU's mechanism to merge a TGA into its action library.

AU's goal is to find an action α_u that generalizes α_1 and α_2, *if it exists*. We say that α_u generalizes α_1 and α_2 if there are two substitutions σ_1, σ_2 such that $\alpha'_i = \alpha_u\sigma_i$, $\mathit{eff}_{\alpha'_i} = \mathit{eff}_{\alpha_i}$ and $pre_{\alpha_i} \models pre_{\alpha'_i}$ for $i \in \{1, 2\}$. Intuitively,

[1]The full details are given in the extended version of this article in arXiv.org

α_u *preserves* the effects of α_1 and α_2, while its preconditions are relaxations of pre_{α_1} and pre_{α_2}, lifting some objects in the process. First, we seek to preserve as many predicates in the precondition as possible. Second, among all the α_u actions with maximal preconditions, we want one with the least number of new parameters. This relaxation/lifting mechanism makes α_u applicable in a wider range of situations. So far, we have described AU as a dual-objective optimization problem.

To perform AU given α_1 and α_2, we encode as a WPMS the problem of finding an injective partial function $\tau : \mathcal{D}_{\alpha_1} \to \mathcal{D}_{\alpha_2}$ such that $\tau(o_1) = o_2$ implies that o_1 and o_2 will map to the same object in α_u. If $o_1 = o_2 = o \in \mathcal{C}$, the reference to o is maintained within α_u as a constant. Otherwise, o_1 and o_2 will be lifted. We say that two labeled predicates $\chi_1 = (\varphi_1, l_1, k_1) \in \mathrm{L}_{\alpha_1}$ and $\chi_2 = (\varphi_2, l_2, k_2) \in \mathrm{L}_{\alpha_2}$ *match* iff $l_1 = l_2$, $\mathrm{type}(\varphi_1) = \mathrm{type}(\varphi_2) = p_n$, and $\tau(\mathrm{arg}_i(\varphi_1)) = \mathrm{arg}_i(\varphi_2)$ for $1 \leq i \leq n$. We denote as $\mathrm{M}_{\alpha_1,\alpha_2}$ the set of tuples (χ_1, χ_2) of potential matches. We say that a labeled predicate $\chi \in \mathrm{L}_{\alpha_i}$ is *preserved* if it has a match in the other action. A labeled predicate is preserved as certain if it is matched with a certain one. In order of priorities, our conditions for τ are: (1) all certain effect predicate are preserved; (2) as many precondition and uncertain predicates as possible are preserved; and (3) as fewer parameters as possible are introduced.

In the WPMS encoding, denoted as Φ_{α_1,α_2}, we use the following decision variables:

- x_{o_1,o_2} for $o_i \in \mathcal{D}_{\alpha_i}$, means that τ maps o_1 to o_2,
- y_{χ_1,χ_2} for $(\chi_1, \chi_2) \in \mathrm{M}_{\alpha_1,\alpha_2}$, means that χ_1 matches χ_2,
- z_{i,χ_i} s.t. $i \in \{1, 2\}$, $\chi_i \in \mathrm{L}_{\alpha_i}$, means that χ_i is preserved.

We define four constraints that must be satisfied (**H**ard), and two constraints to be optimized (**S**oft):

(**H1**) τ is an injective partial function, so $\forall o_1, o_2$:

$$\begin{aligned}&\text{At-Most-1}(\{x_{o_1,o'_2} \mid \forall o'_2\}),\\ &\text{At-Most-1}(\{x_{o'_1,o_2} \mid \forall o'_1\}).\end{aligned} \tag{3}$$

(**H2**) Two potential matches $(\varphi_1, \ldots)$ and $(\varphi_2, \ldots)$, from α_1 and α_2 respectively, match iff τ maps every ith argument of φ_1 to the corresponding ith argument of φ_2:

$$y_{\chi_1,\chi_2} \Leftrightarrow \bigwedge_i x_{\mathrm{arg}_i(\varphi_1),\mathrm{arg}_i(\varphi_2)}. \tag{4}$$

(**H3**) A labeled predicate $\chi_i \in \mathrm{L}_{\alpha_i}$ is preserved iff it has at least one match in the other action:

$$\begin{aligned}z_{1,\chi_1} &\Leftrightarrow \bigvee_{\chi'_2} y_{\chi_1,\chi'_2},\\ z_{2,\chi_2} &\Leftrightarrow \bigvee_{\chi'_1} y_{\chi'_1,\chi_2}.\end{aligned} \tag{5}$$

(**H4**) Preserve non uncertain effects (i.e. $\chi_i = (\varphi_i, l_i, \mathrm{true}) \in \mathrm{L}_{\alpha_i}$, with $l_i \neq \textsc{Pre}$):

$$z_{i,\chi_i}. \tag{6}$$

(S1) (**Weight**=1) Avoid lifting objects, i.e. $\forall o_1, o_2 \in \mathcal{C}$ s.t. $o_1 \neq o_2$:

$$\neg x_{o_1,o_2}. \tag{7}$$

(S2) (**Weight**=W_{big}) Preserve preconditions and uncertain effects (i.e. $\chi_i = (\varphi_i, l_i, k_i)$ with $l_i = \textsc{Pre} \vee \neg k_i$):

$$z_{i,\chi_i}. \tag{8}$$

For compactness, **(H2)** and **(H3)** have not been expressed in *Clausal Normal Form* (CNF), but transforming them to CNF is trivial through the Tseitin transformation (Tseitin 1983). In **(H1)**, At-Most-1 forbids more than one of the literals in the given set to become true. We use the quadratic encoding. Different weights are given to **(S1)** and **(S2)**. W_{big} must be large enough so that preserving predicates has priority over avoiding the introduction of parameters. $W_{big} \geq \min(|\mathcal{D}_{\alpha_1}|, |\mathcal{D}_{\alpha_2}|) + 1$ accomplishes this. Let us highlight that $\mathcal{I}^*(\Phi_{\alpha_1,\alpha_2})$ may be interpreted as a scaled distance between α_1 and α_2:

$$\begin{aligned}\mathcal{I}^*(\Phi_{\alpha_1,\alpha_2}) &= W_{big} \cdot N_{np} + N_{param},\\ dist_{\alpha_1,\alpha_2} &= \mathcal{I}^*(\Phi)/W_{big},\end{aligned} \tag{9}$$

where N_{np} is the number of non-preserved predicates, and N_{param} is the number of introduced parameters. $dist_{\alpha_1,\alpha_2}$ has an intuitive meaning: its integer part represents the number of eliminated predicates (N_{np}) while its fractional part is proportional to the number of introduced parameters (N_{param}/W_{big}). If α_1 and α_2 cannot be unified, we define $dist_{\alpha_1,\alpha_2} = \infty$.

Complexity of Action Unification. We have proposed an approach for AU that requires a reduction to a WPMS problem. However, WPMS is known to be *NP-Hard*, so it is reasonable to wonder if AU is in a more tractable complexity class. We claim that this is not the case.

Theorem 1. *Action Unification's problem is NP-Hard.*[2]

Despite the worst-case complexity of AU, we show that, in practice, real-time performance is possible.

Online Action Recognition Through Unification

OARU shows similarities to *Hierarchical Clustering* (HC). Actions act as data points, and AU computes distances and builds clusters. Unlike standard HC, we cluster only actions whose effects can be preserved.

OARU's recognition subroutine is outlined in Algorithm 1. This subroutine updates $|\mathcal{A}|$ on the basis of a new observation $o = (s, s')$, and explains $s \to s'$ with a grounded action. It starts building a TGA a from o (line 1). After initializing some bookkeeping variables (line 2), it obtains, if possible, the closest action to a from $|\mathcal{A}|$, and the result of AU (lines 3-8). If a could be unified to at least some $\alpha \in \mathcal{A}$, α is replaced by the unified action α', and a' (the return value) is set to the grounding of α' that fills the gap in o (lines 9-12). Otherwise, the TGA is assigned to the return value and added as is to $|\mathcal{A}|$ (lines 13-16).

In practice, we also have a top-level procedure that initializes $\mathcal{A}$ to an empty set and runs Algorithm 1 for each observation it encounters. OARU fills $\mathcal{A}$ progressively, and outputs the grounded action that explains each transition.

[2]Proof in extended article in arXiv.org

Algorithm 1 OARU

```
Input: Observation $o = (s, s')$, action library $\mathcal{A}$
Output: Grounded action $a'$ s.t. $s \xrightarrow{a'} s'$
 1: $a \leftarrow \mathrm{BuildTGA}(s, s')$
 2: $\alpha \leftarrow \emptyset$, $\alpha' \leftarrow \emptyset$, $d_{min} \leftarrow \infty$
 3: for all $\beta \in \mathcal{A}$ do
 4:     $(\alpha_u, dist_{\beta,a}) \leftarrow \mathrm{ActionUnification}(\beta, a)$
 5:     if $dist_{\beta,a} < d_{min}$ then
 6:         $\alpha \leftarrow \beta$, $\alpha' \leftarrow \alpha_u$, $d_{min} \leftarrow dist_{\beta,a}$
 7:     end if
 8: end for
 9: if $\alpha' \neq \emptyset$ then
10:     Remove $\alpha$ from $\mathcal{A}$
11:     $a' \leftarrow \alpha'\sigma$, with $\sigma$ s.t. $eff_{\alpha'\sigma} = eff_a$
12:     Add $\alpha'$ to $\mathcal{A}$
13: else
14:     $a' \leftarrow a$
15:     Add $a$ to $\mathcal{A}$
16: end if
17: return $a'$
```

Evaluation

We have implemented[3] and evaluated[4] OARU in a benchmark of 9 domains (Muise 2016; Silver and Chitnis 2020). We have conducted two sets of experiments: with full and with partial observability. We use goal-oriented traces, i.e. list of pairs of consecutive states where a goal condition holds in the last state, so that all relevant actions are produced.

For each domain, OARU observes the transitions that arise while solving 8 problems in succession. Each problem depicts a different number of objects and, thus, has a different impact in the number of variables and clauses that AU needs to encode[5]. For each domain, we report:

- $|\mathcal{A}|$: final size of OARU's library.
- $|\mathcal{O}|$: total number of transitions across the 8 problems.
- **T**: Average CPU time taken by Algorithm 1.
- **M**: Peak memory usage for solving AU.
- **Prec.**: Precision of a recognized action a_g, compared to the grounded action a_{ref} that was used to perform the transition. It is the ratio of correct labeled predicates in a_g:

$$Prec_{a_{ref}}(a_g) = 100\frac{|\mathrm{L}_{a_g} \cap \mathrm{L}_{a_{ref}}|}{|\mathrm{L}_{a_g}|}. \tag{10}$$

- **Rec.**: Average recall (%) of a_g compared to a_{ref}. It is the ratio of predicates in a_{ref} that have been captured by a_g:

$$Rec_{a_{ref}}(a_g) = 100\frac{|\mathrm{L}_{a_g} \cap \mathrm{L}_{a_{ref}}|}{|\mathrm{L}_{a_{ref}}|}. \tag{11}$$

[3]https://github.com/sprkrd/sat_strips_learn

[4]Machine specifications: AMD Ryzen 7 3700X @ 3.6GHz, 32GB of DDR4 RAM CL15 @ 3200MHz.

[5]See extended article for domain and problem characteristics

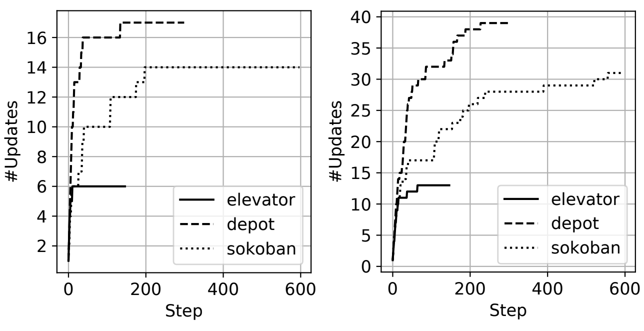
Domain	$\lvert\mathcal{A}\rvert$	$\lvert\mathcal{O}\rvert$	T	M	Prec.	Rec.
blocks	4	96	14 ± 5	1.0	100 ± 1	100 ± 0
depot	5	300	70 ± 57	1.7	92 ± 12	96 ± 5
elevator	3	147	31 ± 32	1.6	87 ± 11	73 ± 10
gripper	3	262	46 ± 30	1.7	100 ± 3	100 ± 0
minecraft	4	23	7 ± 4	1.5	97 ± 6	100 ± 0
onearmedgripper	3	284	38 ± 21	1.7	100 ± 3	100 ± 0
rearrangement	4	42	19 ± 11	1.7	93 ± 9	97 ± 5
sokoban	4	598	49 ± 24	1.7	90 ± 3	91 ± 2
travel	5	48	22 ± 17	1.9	84 ± 15	89 ± 11

(a)

Domain	$\lvert\mathcal{A}\rvert$	$\lvert\mathcal{O}\rvert$	T	M	Prec.	Rec.
blocks	4	96	16 ± 7	1.3	90 ± 15	99 ± 6
depot	5	300	120 ± 66	3.8	88 ± 15	95 ± 7
elevator	3	147	48 ± 26	5.3	83 ± 22	66 ± 19
gripper	3	262	45 ± 25	5.2	96 ± 10	100 ± 2
minecraft	4	23	30 ± 27	5.4	65 ± 23	99 ± 6
onearmedgripper	3	284	32 ± 14	5.4	95 ± 10	100 ± 3
rearrangement	4	42	28 ± 15	5.5	80 ± 20	96 ± 7
sokoban	4	598	158 ± 146	13	89 ± 12	86 ± 5
travel	5	48	91 ± 114	16	68 ± 27	85 ± 10

(b)

Table 1: Results with (a) full and (b) partial observability. T is in milliseconds, M in MB, and Prec. and Rec. in %.



(a) (b)

Figure 2: Accumulated updates for three domains in (a) full and (b) partial observability.

T and **M** are performance-related (quantitative evaluation). **Prec.** and **Rec.** measure the correctness and completeness of a_g (qualitative evaluation). We report both evaluations for full and partial observability in Table 1.

Let us focus first on the full-observability results (Table 1a). Times are in general well below 1 second. Therefore, our claim about the real-time potential of OARU holds, as long as the throughput of observations is reasonably limited (e.g. a few seconds between observations). Memory requirements are also within a reasonable bound, requiring in the order of a few MBs to solve AU. However, we have found that some domains are challenging to OARU when it has just started with an empty $\mathcal{A}$. For instance, OARU cannot cope with *gripper* in a timely fashion if it starts with a problem with 20 objects or more. This is understandable, because the size of the WPMS encoding grows quadratically with the matches of predicates and objects. We believe this is not a huge setback, since it is very natural to ramp up the number of objects progressively. Additional performance is achieved through some optimizations not described here, like a *broad phase* stage in Algorithm 1 to avoid a full call to AU when it is obvious that actions cannot be unified.

We see in general that the recall is very large. The reason lies in how OARU works. In deterministic settings, effects are almost always identified with high accuracy, while preconditions are relaxed as little as possible. In 5 cases the recall is lower than 100%. In 3 of those, moreover, $|\mathcal{A}|$ is not equal to the number of actions in the expert's domain or *Ground Truth Model* (GTM). In *elevator* (GTM with 4 actions), there are two very similar actions for going up and down that only differ in one predicate of the precondition. OARU recognizes them as a generic move up or down action and discards that predicate. On the other hand, *sokoban* (GTM with 3 actions) contains an action for moving stones to non-goal positions that deletes an *at-goal*$(\cdot)$ predicate which is not necessarily in the state. Thus, its deletion is not always observed, and OARU fills $|\mathcal{A}|$ with two different actions that address different contexts. *Travel* (GTM with 4 actions) presents a similar situation, but for adding an already present predicate. The 2 remaining cases (*rearrangement* and *depot*) have lower than 100% recalls for similar reasons, but their $|\mathcal{A}|$ is equal to their GTM's.

The precision is not as high as the recall because OARU holds onto precondition predicates that are not present in the GTM, but are confirmed by all the observed transitions. For instance, grid-based worlds with adjacency predicates, like *sokoban*, show symmetries. To move from `pos-5-6` to `pos-6-6`, a predicate such as `move-dir(pos-5-6, pos-6-6, dir-right)` must hold true. However, OARU observes that, whenever that happens, the predicate `move-dir(pos-6-6, pos-5-6, dir-left)` is also true. However, the GTM lists only one adjacency precondition. In most cases, OARU is not incorrect from a semantic standpoint, but the precision metric counts it as an error.

Table 1b shows the results for partial observability. These have been obtained making parts of the state ignored by OARU at random. Namely, the interpretation of from 0 to 5 predicates has been removed. Results are averaged over 5 executions of the 8 problems for all the domains. We see penalties in the performance. This is most evident in *sokoban*, because all problems present more than 50 objects. The introduction of uncertain predicates drives up the number of potential matches significantly. Other domains show smaller increases in recognition times and memory requirements. Precision and recall have suffered, but are not significantly worse than the full observability results. We have also experimented with greater proportions of unknown predicates (0 to 10). OARU starts requiring large computational times for AU in *elevator* and *sokoban* (in the order of several minutes). As we would expect, partial observability has a much larger computational burden. A way to address all these problems is to start with smaller problems and build $|\mathcal{A}|$ progressively.

Figure 2 shows how often $\mathcal{A}$ is updated (via action addition or replacement) in three domains. The X axis shows the number of steps taken, while the Y axis shows the accumulated number of updates. We consider that an update is made only when a TGA is not entirely subsumed by an action already present in $\mathcal{A}$ (i.e. either a parameter is added or a labeled predicate is removed). Notice that $\mathcal{A}$ stabilizes early on for *elevator*. For *sokoban*, it plateaus half-way towards the final number of steps, and *depot* at approximately two thirds. The drop in the rise rate of the curves shows that, early on, $\mathcal{A}$ is empty and is updated rather frequently with new observations. However, as $\mathcal{A}$ is filled, OARU sees more actions and comes up with good general schemata that generalize all the transitions that could happen. The shapes of the full and partial observability curves are very similar. The most notable differences are in the scale and that the partial observability curves do not entirely plateau, but still exhibit some small jumps towards the final steps. This is intuitive: partial observations cause an increased number of updates, and that $\mathcal{A}$ does not stabilize until later.

Discussion and Conclusions

In this paper, we have proposed OARU, an algorithm for recognizing STRIPS action models from partially observable state transitions. It uses AU to merge observations into its action library, constructing an action hierarchy and improving its action models through generalization. OARU shows a high computing performance and the recognized actions have strong similarities to expert handcrafted models. Thus, OARU ability to learn without action signatures makes it a promising contender to other model learning approaches.

As future work we consider to learn more expressive actions with generalized planning (Jiménez, Segovia-Aguas, and Jonsson 2019), and refine actions with negative examples (Aguas, Jiménez, and Jonsson 2020). Allow noisy observations extracted from sensor sampling (Yang 2009) which would make it suitable for more realistic applications on the wild. Also recognizing hidden variables and intermediate states in action sequences with Bayesian inference (Aineto, Jimenez, and Onaindia 2020). Finally, we think there is a great potential for an application-ready methodology adopting OARU's philosophy to learn from low-level data, i.e. robot motions (Konidaris, Kaelbling, and Lozano-Perez 2018), images (Asai and Fukunaga 2018) or graphs (Bonet and Geffner 2020).

Acknowledgments

The research leading to these results has received funding from the EU H2020 research and innovation programme under grant agreement no.731761, IMAGINE; the HuMoUR project TIN2017-90086-R (AEI/FEDER, UE); and AEI through the María de Maeztu Seal of Excellence to IRI (MDM-2016-0656). Javier Segovia-Aguas was also partially supported by TAILOR, a project funded by EU H2020 research and innovation programme no. 952215, an ERC Advanced Grant no. 885107, and grant TIN-2015-67959-P from MINECO, Spain.

References

Aguas, J. S.; Jiménez, S.; and Jonsson, A. 2020. Generalized Planning with Positive and Negative Examples. In *Pro-*

ceedings of the AAAI Conference on Artificial Intelligence, 9949–9956.

Aineto, D.; Celorrio, S. J.; and Onaindia, E. 2019. Learning action models with minimal observability. *AIJ* 275: 104–137.

Aineto, D.; Jiménez, S.; and Onaindia, E. 2018. Learning STRIPS action models with classical planning. In *ICAPS*.

Aineto, D.; Jimenez, S.; and Onaindia, E. 2020. Observation Decoding with Sensor Models: Recognition Tasks via Classical Planning. In *ICAPS*, volume 30, 11–19.

Aineto, D.; Jiménez, S.; Onaindia, E.; and Ramírez, M. 2019. Model Recognition as Planning. In *ICAPS*, 13–21.

Amado, L.; Pereira, R. F.; Aires, J.; Magnaguagno, M.; Granada, R.; and Meneguzzi, F. 2018. Goal recognition in latent space. In *IJCNN*, 1–8. IEEE.

Amir, E.; and Chang, A. 2008. Learning partially observable deterministic action models. *JAIR* 33: 349–402.

Ansótegui, C.; and Gabas, J. 2013. Solving (Weighted) Partial MaxSAT with ILP. In *CPAIOR*, volume 13, 403–409.

Arora, A.; Fiorino, H.; Pellier, D.; Métivier, M.; and Pesty, S. 2018. A review of learning planning action models. *The Knowledge Engineering Review* 33.

Asai, M.; and Fukunaga, A. 2018. Classical Planning in Deep Latent Space: Bridging the Subsymbolic-Symbolic Boundary. In McIlraith, S. A.; and Weinberger, K. Q., eds., *Proceedings of the AAAI Conference on Artificial Intelligence*, 6094–6101.

Benson, S. 1995. Inductive learning of reactive action models. In *Machine Learning Proceedings 1995*, 47–54. Elsevier.

Bonet, B.; and Geffner, H. 2020. Learning first-order symbolic representations for planning from the structure of the state space. In *ECAI*.

Chakraborti, T.; Sreedharan, S.; Zhang, Y.; and Kambhampati, S. 2017. Plan explanations as model reconciliation: Moving beyond explanation as soliloquy. In *IJCAI*, 156–163.

Cresswell, S.; and Gregory, P. 2011. Generalised Domain Model Acquisition from Action Traces. In *ICAPS*.

Cresswell, S.; McCluskey, T. L.; and West, M. M. 2009. Acquisition of Object-Centred Domain Models from Planning Examples. In *ICAPS*.

Gil, Y. 1994. Learning by experimentation: Incremental refinement of incomplete planning domains. In *Machine Learning Proceedings 1994*, 87–95. Elsevier.

Gregory, P.; and Cresswell, S. 2015. Domain Model Acquisition in the Presence of Static Relations in the LOP System. In *ICAPS*, 97–105.

Haslum, P.; Lipovetzky, N.; Magazzeni, D.; and Muise, C. 2019. An introduction to the planning domain definition language. *Synthesis Lectures on Artificial Intelligence and Machine Learning* 13(2): 1–187.

Jiménez, S.; Segovia-Aguas, J.; and Jonsson, A. 2019. A review of generalized planning. *The Knowledge Engineering Review* 34.

Junghanns, A.; and Schaeffer, J. 1997. Sokoban: A challenging single-agent search problem. In *In IJCAI Workshop on Using Games as an Experimental Testbed for AI Reasearch*. Citeseer.

Keren, S.; Gal, A.; and Karpas, E. 2014. Goal recognition design. In *ICAPS*.

Konidaris, G.; Kaelbling, L. P.; and Lozano-Perez, T. 2018. From skills to symbols: Learning symbolic representations for abstract high-level planning. *JAIR* 61: 215–289.

McDermott, D.; Ghallab, M.; Howe, A.; Knoblock, C.; Ram, A.; Veloso, M.; Weld, D.; and Wilkins, D. 1998. PDDL-the planning domain definition language. Technical Report CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control.

Minker, J. 1982. On indefinite databases and the closed world assumption. In *International Conference on Automated Deduction*, 292–308. Springer.

Mourao, K.; Zettlemoyer, L. S.; Petrick, R. P. A.; and Steedman, M. 2012. Learning STRIPS Operators from Noisy and Incomplete Observations. ArXiv preprint arXiv:1210.4889.

Muise, C. 2016. Planning.Domains. In *ICAPS - Demonstrations*. URL http://www.haz.ca/papers/planning-domains-icaps16.pdf.

Pozanco, A.; Yolanda, E.; Fernández, S.; and Borrajo, D. 2018. Counterplanning using Goal Recognition and Landmarks. In *IJCAI*, 4808–4814.

Ramírez, M.; and Geffner, H. 2009. Plan recognition as planning. In *IJCAI*, 1778–1783.

Ramírez, M.; and Geffner, H. 2010. Probabilistic Plan Recognition Using Off-the-Shelf Classical Planners. In Fox, M.; and Poole, D., eds., *Proceedings of the AAAI Conference on Artificial Intelligence*.

Segovia-Aguas, J.; Jiménez, S.; and Jonsson, A. 2017a. Generating context-free grammars using classical planning. In *IJCAI*, 4391–4397.

Segovia-Aguas, J.; Jiménez, S.; and Jonsson, A. 2017b. Unsupervised classification of planning instances. In *ICAPS*, 452–460.

Silver, T.; and Chitnis, R. 2020. PDDLGym: Gym Environments from PDDL Problems. arXiv preprint arXiv:2002.06432.

Snyder, F. B.-W. 2001. Unification theory. *Handbook of automated reasoning* 1: 447–533.

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.

Suárez-Hernández, A.; Segovia-Aguas, J.; Torras, C.; and Alenyà, G. 2020. STRIPS Action Discovery. arXiv preprint arXiv:2001.11457.

Tseitin, G. S. 1983. On the complexity of derivation in propositional calculus. In *Automation of reasoning*, 466–483. Springer.

Wang, X. 1995. Learning by observation and practice: An incremental approach for planning operator acquisition. In *Machine Learning Proceedings 1995*, 549–557. Elsevier.

Yang, Q. 2009. Activity recognition: linking low-level sensors to high-level intelligence. In *IJCAI*, volume 9, 20–25.

Yang, Q.; Wu, K.; and Jiang, Y. 2007. Learning action models from plan examples using weighted MAX-SAT. *AIJ* 171(2-3): 107–143.