Journal of Artificial Intelligence Research 74 (2022) 1091-1123 Submitted 06/2021; published 07/2022

A Comprehensive Framework for Learning Declarative Action Models

Diego Aineto DIEAIGAR@VRAIN.UPV.ES
Sergio Jiménez SERJICE@VRAIN.UPV.ES
Eva Onaindia ONAINDIA@VRAIN.UPV.ES
Valencian Research Institute for Artificial Intelligence
Universitat Politècnica de València, Valencia, Spain

Abstract

A declarative action model is a compact representation of the state transitions of dynamic systems that generalizes over world objects. The specification of declarative action models is often a complex hand-crafted task. In this paper we formulate declarative action models via *state constraints*, and present the learning of such models as a *combinatorial search*. The comprehensive framework presented here allows us to connect the learning of declarative action models to well-known problem solving tasks. In addition, our framework allows us to characterize the existing work in the literature according to four dimensions: (1) the target action models, in terms of the state transitions they define; (2) the available learning examples; (3) the functions used to guide the learning process, and to evaluate the quality of the learned action models; (4) the learning algorithm. Last, the paper lists relevant successful applications of the learning of declarative actions models and discusses some open challenges with the aim of encouraging future research work.

1. Introduction

Learning declarative action models from the execution of actions is a well-studied problem with successful systems like ARMS (Yang, Wu, & Jiang, 2007), SLAF (Amir & Chang, 2008), LOCM (Cresswell, McCluskey, & West, 2013) or FAMA (Aineto, Jiménez, & Onaindia, 2019). The importance of declarative models has been highlighted by the broad variety of tasks that make use of them such as the synthesis of plans (Ghallab, Nau, & Traverso, 2004); the computation of future rewards (Diuk, Cohen, & Littman, 2008); the recognition of the intentions of observed agents (Ramírez, 2012); system diagnosis (Grastien, Haslum, & Thiébaux, 2012); decoding unobserved behaviour of an agent (Aineto, Jiménez, & Onaindia, 2020); the automatic reconciliation of different world models (Chakraborti, Sreedharan, Zhang, & Kambhampati, 2017) and many more.

Our primary motivation for conducting the research presented in this paper is to come up with a comprehensive framework for learning declarative action models that help researchers understand the principal design choices when building a learning system. In order to achieve this, we interpret the learning task using well-founded AI formalisms, such as *state constraints* and *combinatorial search*, and propose a framework built upon four pillars: the type of input data, the hypothesis space of the learning problem, the function to guide the search and the learning paradigm.

The proposed framework allows us to map approaches to learning declarative action models into well-studied AI problem-solving tasks (e.g. such as *logic satisfiability*, *constraint satisfaction*, *AI planning*, *function optimization*, or *belief tracking*). More importantly, it allows us to undertake a formal analysis of the advantages and limitations of these approaches, and to compare them beyond their empirical performance. In addition, by analysing the state-of-the-art on the basis of the four aforementioned pillars, the current issues and challenges that the field faces become apparent, opening the door to new lines of research.

1.1 Declarative Action Models

In this paper we use the term *declarative* as it is used in etymology; by learning *declarative action models* we refer to acquiring action knowledge that humans can speak about. We humans speak of the world in terms of objects, that have certain properties, and in terms of the relations that are established between such objects. Furthermore, we tend to cluster similar objects into abstract classes (types) and assume that different objects, that behave similarly, are actually instances of an abstract class. This assumption facilitates generalization and transfer learning to unseen environments (Ardón, Pairet, Lohan, Ramamoorthy, & Petrick, 2021).

We refer to a *declarative action model* as a compact and high-level representation of the transition model of a dynamic system. A declarative action model generalizes to a family of (possibly infinite) different tasks regardless of the number of objects of the task, the identity of such objects, their initial setup, or the objective of the task.

The input to the task of learning a declarative action model is a collection of observations of the execution of actions. Observations may come from different tasks and, therefore, comprise information about different objects as well as on different numbers of objects. A declarative action model learned from observation samples of state transitions of a collection of tasks is expected to apply to tasks not seen in the task collection. Approaches to learning declarative action models specifically aim for task-independent models as generalization is a desired property in all types of learned models.

We put the focus on declarative models represented with *action schemes*. Action schemes compactly specify a set of transitions (that share a common structure) in terms of parameterized *preconditions* and *effects*. A precondition is a Boolean function that captures whether an action is applicable in a given state. An effect is a successor function that specifies the next state that results after an action application. Action schemes generalize over any number or identity of world objects; they do not refer to specific objects but instead they leverage parameters to indirectly refer to the different world objects.

The output of a learning system is a set of action schemes and so the system needs to learn the preconditions and effects of each scheme. Learning preconditions can be regarded as a Boolean classification problem that splits the state space into positive states, in which actions are applicable, and negative states. Learning the effects of an action is concerned with computing the state constraints that determine the updated values of the state variables.

In the comprehensive framework presented in this paper, we formulate declarative action models via *state constraints* and the learning of declarative action models as a *combinatorial search*:

- Our formulation makes explicit the *alphabet* of the state constraints that appear in the preconditions/effects of an action. Specifying this alphabet enables to objectively characterize the expressive power of the different learning systems.

- The space of the combinatorial search is comprised of the set of declarative action models that can be built with the given *alphabet* of the state constraints. The larger the alphabet, the larger the search space and hence, the more computationally challenging the learning process. This paper considers that the search in the space of declarative action models is guided by a certain function that estimates, how likely it is that an action model actually generated the learning examples. Learning systems are also characterized by their ability to estimate the fitness of an action model to a given set of learning examples.

1.2 Related Topics

Here we review different AI tasks where models of state transitions, as well as the inductive learning of such models, play an important role.

The inductive learning of properties and relations that generalize over world objects is studied since the early days of AI, with seminal works on the classical *blocksworld* domain (Fikes, Hart, & Nilsson, 1972; Winston, 1970). This kind of machine learning is characterized by the use of first-order logic representations and it is studied under different names and approaches, including *inductive logic programming* (ILP) (Muggleton, 1992; Bergadano & Gunetti, 1996) and *statistical relational learning* (Getoor & Taskar, 2007; De Raedt, 2008; De Raedt & Kersting, 2017). Because of the first-order nature of these learning approaches, the developed algorithms usually apply straightforwardly to the learning of declarative action models. In fact the international ILP competition evaluated the participant systems on benchmarks for learning the possible transitions of an agent in different 2D environments[1].

Automated planning is the model-based approach for autonomous control (Ghallab et al., 2004; Geffner & Bonet, 2013). Classical planning is the simplest model in automated planning, and it is concerned with the selection of actions, in a given deterministic transition system, to achieve a given goal condition, starting from a given initial state. In classical planning, relational learning has been used to relieve the knowledge acquisition bottleneck and automatically specify a compact representation of the possible state transitions (Jiménez, De La Rosa, Fernández, Fernández, & Borrajo, 2012; Arora, Fiorino, Pellier, Etivier, & Pesty, 2018). In addition, relational learning has also shown useful for the learning of generalized policies that can solve a set of classical planning instances from a given domain (Khardon, 1999; Martin & Geffner, 2004; Yoon, Fern, & Givan, 2008; De la Rosa, Jiménez, Fuentetaja, & Borrajo, 2011; Hu & De Giacomo, 2011; Jiménez, Segovia-Aguas, & Jonsson, 2019). These two learning tasks are actually connected; a policy can be understood as the extension of an action model, with additional preconditions, to capture where actions must be applied to achieve a given objective (Francès, Bonet, & Geffner, 2021).

In a *Markov Decision Process* (MDP), a decision-making agent selects an action to apply in a fully observable state and the environment responds updating the state of the system with the corresponding transition (Kolobov, 2012). MDPs usually include also a reward function to model the immediate reward that is received by the decision-making agent after a state transition. The Partially Observable Markov Decision Process (POMDP) is a generalization of the MDP model which does not assume that states are fully observable (Shani, Pineau, & Kaplow, 2013). Decision-making problems are formalized over MDPs and POMDPs to compute state-action policies that reach some goal state, or that maximize the cumulative reward over the states reachable from the given initial state(s) of the MDP/POMDP.

As happens in classical planning, machine learning is also used to avoid the hand-crafted specification of MDP and POMDP models. However, specifying an MDP/POMDP model goes beyond determining the set of possible state transitions; MDPs specify also the *reward function* and POMDPs specify the *observation function*, which determines the probability of getting a partic-

1. http://ilp16.doc.ic.ac.uk/competition

ular observation from a given state. MDP/POMDP models are often represented with propositional formalisms, that explicitly enumerate the state space, or with flat *attribute-value* representations (Konidaris, Kaelbling, & Lozano-Pérez, 2018). The connection with the learning of declarative action models becomes evident when MDP/POMDPs leverage action schemes that generalize over the different number and identity of the world objects (Finney, Gardiol, Kaelbling, & Oates, 2002; Abbeel & Ng, 2005; Pasula, Zettlemoyer, & Kaelbling, 2007a). Relational learning is also applied in MDP/POMDP to induce compact first-order policies, and compact first-order representations of the value function, that improve the scalability of solvers (Sanner & Boutilier, 2009; Wang & Khardon, 2010).

The learning of MDP/POMDP models is usually studied in the context of *reinforcement learning*. Reinforcement Learning (RL) (Sutton & Barto, 2018) is a general paradigm for developing sequential decision-making agents with minimal supervision. An RL agent is situated in an environment, whose dynamics are initially unknown, and interacts with that environment taking actions and receiving the corresponding rewards (the positive or negative reinforcement). RL agents compute a policy that determines the actions to take balancing two competing motivations: (i), the *exploration* of the unknown environment and (ii), the *exploitation* of the collected knowledge. The RL problem is typically formulated in the MDP/POMDP framework, assuming that the transition and reward functions of the underlying MDP/POMDP are unknown. With this regard, we can find RL agents that work with atomic representations of the states (Kaelbling, Littman, & Moore, 1996; Shani, Brafman, & Shimony, 2005), factored state representations (Kearns & Koller, 1999; Sallans & Hinton, 2004; Strehl, Diuk, & Littman, 2007), as well as first-order state representations, that allow the RL agent to abstract from specific objects and goals (Džeroski, De Raedt, & Driessens, 2001; Tadepalli, Givan, & Driessens, 2004). Besides a policy, *model-based* RL agents learn also an approximation of the underlying MDP/POMDP model, which is updated with new observed transitions, and that is leveraged to guide exploration with the aim of achieving better rewards safer and faster (Pasula, Zettlemoyer, & Kaelbling, 2007b; Diuk et al., 2008; Walsh, Goschin, & Littman, 2010; Lang, Toussaint, & Kersting, 2012; Martínez, Alenyà, Ribeiro, Inoue, & Torras, 2017). RL agents cannot however tackle some of the expressive models that are actually handled in automated planning and they can struggle in domains where rewards are scarce, which are common in pure planning problems (Ahmed, Träuble, Goyal, Neitz, Wüthrich, Bengio, Schölkopf, & Bauer, 2021).

1.3 Organization of the Paper

The contents of this paper is organized as follows:

Section 2 constitutes the core of the paper, where all relevant components of the learning of declarative action models are introduced. We formalize **the hypothesis space**, i.e. the set of possible action models that can be learned. The section formalizes also **the learning examples**; that is the type of observations of action execution used to feed the learning of declarative action models. Last, the section formalizes **the error functions** that are used to guide the search in the space of possible declarative action models, as well as to evaluate the quality of the learned action models.

Section 3 presents **the learning paradigms** corresponding to the different approaches to the learning of declarative action models. Each learning paradigm presents different aims, produces a different output, and corresponds to a different classic AI task.

Section 4 reviews **the most popular systems** for learning declarative action models, that are implemented in the 15-year period 2005-2020.

Section 5 presents a thorough discussion on various **open challenges** concerning the learning of declarative action models as well as successful examples of **applications** of this learning technology.

Section 6 wraps up this work with our **conclusions**, summarizing the proposed framework and its importance.

2. Learning Declarative Action Models

This section formalizes the inductive learning of declarative action models as a combinatorial search that is structured around three elements: *the hypothesis space* of the possible action models, *the learning examples*, and *the error functions* to guide search and evaluate the learned models. Next, we delve into each of these three elements.

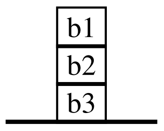
2.1 The Hypothesis Space

This paper focuses on declarative action models represented as action schemes that comprise parameterized preconditions and effects. Learning action schemes that include additional features, such as the execution cost or duration of an action, has been addressed in several works (Lanchas, Jiménez, Fernández, & Borrajo, 2007; Yang, Khandelwal, Leonetti, & Stone, 2014; Gregory & Lindsay, 2016). Nonetheless, this paper focuses on the learning of preconditions and effects as these are the common features addressed by most of the work on learning declarative action models. In the following, we show how to formalize the hypothesis space of an action scheme using the definition of *schematic invariants* introduced by Rintanen (2017).

Definition 1 (Schematic state variable) *Let V be a set of variable symbols. A* schematic state variable *is of the form $z = name(params)$ where $name$ is a symbol identifying the schematic state variable, and $params \subseteq V$ is a list of variable symbols. We denote by Z a finite set of schematic state variables, where each $z \in Z$ is associated with a finite domain $D(z)$*

Given a set of objects O, a schematic state variable $z = name(params)$ induces a set of state variables given by the First Order Logic (FOL) interpretations of the parameters of z over the objects of O. Typification can be applied to both variable symbols V and objects of O in order to constrain the FOL interpretation and keep the set of induced state variables compact. For the sake of simplicity, however, we will not consider typification in this work.

We now show an example that illustrates these notions in the classical AI *blocksworld* domain (Slaney & Thiébaux, 2001). In this example we use the STRIPS fragment of the PDDL language (Fox & Long, 2003) whose variables are Boolean. Let $Z = \{clear(x), handempty(), holding(x), on(x, y), ontable(x)\}$ be the set of five schematic state variables, where each variable is built on two variable symbols, $V = \{x, y\}$. The left part of Figure 1 shows a three-block state from the *blocksworld* that involves three objects, $O = \{b1, b2, b3\}$. In the right part of the figure we can see the schematic state variables of Z, and the induced state variables (corresponding to interpretations of Z over O) that represent the three-block state. State variables are represented in standard PDDL syntax and following the *closed world* assumption where missing state variables are assumed



Schematic state variables	State variables
clear(x)	`(clear b1)`
handempty()	`(handempty)`
holding(x)	-
on(x, y)	`(on b1 b2) (on b2 b3)`
ontable(x)	`(ontable b3)`

Figure 1: Example of a three-block state from the *blocksworld* (left), and its representation formed by the Boolean state variables that are induced from the given set of schematic state variables (table on the right).

to be false. For instance, $clear(b1) = True$ is represented by Boolean variable $(\texttt{clear b1})$ whereas $clear(b2) = False$ and $clear(b3) = False$ are missing because they are evaluated to `False` in this example.

Definition 2 (Schematic state constraint) *Let Z be a finite set of schematic state variables built on a set of variable symbols V. A* schematic state constraint *is a formula over Z, which is inductively defined as follows:*

- *a schematic state variable $z \in Z$.*
- *$\neg\phi$, if ϕ is a schematic state constraint.*
- *conjunction $\phi_1 \wedge \phi_2$, if both ϕ_1 and ϕ_2 are schematic state constraints.*
- *disjunction $\phi_1 \vee \phi_2$, if both ϕ_1 and ϕ_2 are schematic state constraints.*

Once introduced the concepts of schematic state variable and schematic state constraint, we are ready to formalize the concept of *action scheme*.

Definition 3 (Action scheme) *Let Z be a finite set of schematic state variables and V the variable symbols that appear in such variables. An* action scheme *is a tuple $\mu = \langle name, params, pre, eff\rangle$ where:*

- *$name$ is the symbol labeling the action scheme.*
- *$params \subseteq V$ is a list of variable symbols. We denote by Z_μ the subset of schematic state variables $Z_\mu \subseteq Z$ defined over the symbols in $params$.*
- *pre is a schematic state constraint defined over Z_μ, that constrains the value of a subset of state variables at the state* before *the application of the action scheme.*
- *eff is a schematic state constraint defined over Z_μ, that constrains the value of a subset of state variables at the state* after *the application of the action scheme.*

Given a set of objects O, an action scheme μ induces a set of ground actions given by the FOL interpretations of the parameters of μ over the objects of O. Figure 2 shows a *blocksworld* action scheme for stacking a block on top of another block. The action scheme is represented in the

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

Figure 2: *Stack* action scheme from the *blocksworld*, represented in the STRIPS fragment of the PDDL language.

STRIPS fragment of the PDDL language (Fox & Long, 2003) and it exhibits two parameters `(?x ?y)`. In this particular action scheme pre and eff are conjunctions of schematic state variables.

Once introduced the structure of an action scheme, we now formalize the elements of the language that are needed to build an action scheme.

Definition 4 (The alphabet of an action scheme) *Let Z be a finite set of schematic state variables built on variable symbols V. The* alphabet of an action scheme *μ, with signature $\langle name, params\rangle$, is the finite set of Boolean variables $\langle name, pe, \phi\rangle$ such that $pe \in \{\texttt{pre}, \texttt{eff}\}$ and ϕ is a schematic state constraint defined over Z_μ.*

Let $\mathcal{A}$ be the alphabet of an action scheme μ; a full variable assignment of $\mathcal{A}$ defines an action scheme such that variables $\langle name, pe, \phi\rangle$ set to true represent that ϕ belongs to the preconditions/effects of μ. Therefore, a total of $2^{|\mathcal{A}|}$ action schemes can be defined with $\mathcal{A}$.

Let us exemplify the alphabet of an action model. In this example, we look at the STRIPS fragment of the PDDL representation language where pre and eff are conjunctions of schematic state variables and action effects cannot be positive and negative at the same time. Considering the schematic state variables of Figure 1, the alphabet of the action scheme `stack(x, y)` is the finite set of Boolean variables $\langle \texttt{stack}, \texttt{pre}, \phi\rangle$ and $\langle \texttt{stack}, \texttt{eff}, \phi\rangle$ where ϕ belongs to the set

{`clear(x), clear(y), handempty(), holding(x), holding(y)`
`on(x,x), on(x,y), on(y,x), on(y,y),`
`ontable(x), ontable(y)`}

Here, a variable $\langle \texttt{stack}, pe, \phi\rangle$ is representing whether ϕ is a conjunct of the precondition or effects of the `stack` scheme. This alphabet leverages the STRIPS constraints in the following way: (1) if a schematic state variable appears in both the preconditions and the effects then it means it is representing a negative effect, (2) if a schematic state variable only appears in the effects then it means it is representing a positive effect. This encoding results in the most compact possible alphabet as any full assignment of $\mathcal{A}$ defines a valid action scheme that complies with the syntactic constraints of STRIPS.

Figure 3 shows the full variable assignment of the alphabet that defines the `stack` action scheme. In this encoding, $\langle \texttt{stack}, \texttt{eff}, holding(x)\rangle$ and $\langle \texttt{stack}, \texttt{eff}, clear(y)\rangle$ represent negative effects, because $\langle \texttt{stack}, \texttt{pre}, holding(x)\rangle$ and $\langle \texttt{stack}, \texttt{pre}, clear(y)\rangle$ are also set to `True`. The other three effects that are set to `True`, namely $\langle \texttt{stack}, \texttt{eff}, clear(x)\rangle$, $\langle \texttt{stack}, \texttt{eff}, handempty()\rangle$, and $\langle \texttt{stack}, \texttt{eff}, on(x, y)\rangle$, represent positive effects.

$\langle \texttt{stack}, \texttt{pre}, holding(x) \rangle = \texttt{True}$	$\langle \texttt{stack}, \texttt{eff}, holding(x) \rangle = \texttt{True}$
$\langle \texttt{stack}, \texttt{pre}, clear(y) \rangle = \texttt{True}$	$\langle \texttt{stack}, \texttt{eff}, clear(y) \rangle = \texttt{True}$
	$\langle \texttt{stack}, \texttt{eff}, clear(x) \rangle = \texttt{True}$
	$\langle \texttt{stack}, \texttt{eff}, handempty() \rangle = \texttt{True}$
	$\langle \texttt{stack}, \texttt{eff}, on(x,y) \rangle = \texttt{True}$
$\langle \texttt{stack}, \texttt{pre}, holding(y) \rangle = \texttt{False}$	$\langle \texttt{stack}, \texttt{eff}, holding(y) \rangle = \texttt{False}$
$\langle \texttt{stack}, \texttt{pre}, ontable(x) \rangle = \texttt{False}$	$\langle \texttt{stack}, \texttt{eff}, ontable(x) \rangle = \texttt{False}$
$\langle \texttt{stack}, \texttt{pre}, ontable(y) \rangle = \texttt{False}$	$\langle \texttt{stack}, \texttt{eff}, ontable(y) \rangle = \texttt{False}$
$\langle \texttt{stack}, \texttt{pre}, on(x,x) \rangle = \texttt{False}$	$\langle \texttt{stack}, \texttt{eff}, on(x,x) \rangle = \texttt{False}$
$\langle \texttt{stack}, \texttt{pre}, on(y,x) \rangle = \texttt{False}$	$\langle \texttt{stack}, \texttt{eff}, on(y,x) \rangle = \texttt{False}$
$\langle \texttt{stack}, \texttt{pre}, on(y,y) \rangle = \texttt{False}$	$\langle \texttt{stack}, \texttt{eff}, on(y,y) \rangle = \texttt{False}$
$\langle \texttt{stack}, \texttt{pre}, clear(x) \rangle = \texttt{False}$	
$\langle \texttt{stack}, \texttt{pre}, handempty() \rangle = \texttt{False}$	
$\langle \texttt{stack}, \texttt{pre}, on(x,y) \rangle = \texttt{False}$	

Figure 3: Full variable assignment of the alphabet that defines the STRIPS action scheme `stack(x, y)` from the *blocksworld* domain.

Given that a declarative action model is formalized as a finite set of action schemes, the **hypothesis space** of the possible action models is given by the number of action schemes and their respective alphabets.

Definition 5 (The hypothesis space) *Let $\mathcal{A}$ be the alphabet of an action model, built as the union of the alphabets of the schemes that make up the action model. The* space of hypothesis $\mathcal{M}$ *is the finite set of action models such that each $M \in \mathcal{M}$ corresponds to an interpretation in $2^{\mathcal{A}}$.*

The hypothesis space $\mathcal{M}$ includes all the action models that can be defined as full assignments of a given alphabet. As a rule of thumb, more expressive representation languages involve larger alphabets which, in turn, entail larger hypothesis spaces.

Bit-vectors provide a convenient encoding for the hypothesis space of the possible action models: each bit in the vector codes whether a variable in the alphabet of an action scheme is set to true or false. The bit-vector encoding facilitates the implementation of a combinatorial search in the space of possible action models because it allows to: (i) quantify the distance between two different models; (ii) sort the space of possible action models; and (iii) enable the activation (or deactivation) of the bit associated to a schematic constraint when such precondition/effect is added (or removed) from an action scheme (Aineto, Jiménez, Onaindia, & Ramírez, 2019). Learning systems like ARMS, SLAF, ALICE, or FAMA leverage binary encoding of the space of possible action models.

The definitions introduced in this section consider only Boolean schematic state variables and logic formula constraints, as this is generally the scope of the works in the learning of declarative action models. However, our definitions can be naturally extended to consider finite domain and numeric state variables as well as schematic state constraints defined as arithmetic-logic formulae. Similarly, our definition of action scheme can be extended to specify constraints that refer to states other than the *pre-state* or the *post-state* such as, for instance, constraints *over all* the sequence of states traversed during the execution of a durative action (Fox & Long, 2003) in a *durative action* scheme.

2.2 The Learning Examples

In this section, we present the learning examples used as input to the task of learning of declarative action models. Broadly speaking, learning examples contain data collected from the transition system for which we want to build a model. Learning examples are typically presented in the form of trajectories, that contain a sequence of actions and the states traversed in its execution. This section formalizes the concept of trajectory and then extends the discussion to introduce the notion of observation of a trajectory.

Definition 6 (State) *Let X be a set of state variables, where each variable $x \in X$ has domain $D(x)$. A state $s = \langle x_1 = v_1, \ldots, x_{|X|} = v_{|X|}\rangle$ is a full assignment of values $v_i \in D(x_i)$ over the state variables $x_i \in X$. We use S to denote the set of all states.*

Definition 7 (Trajectory) *A* trajectory *$\tau = \langle s_0, a_1, s_1, \ldots, a_n, s_n\rangle$ is an interleaved sequence of states and actions, that results from the execution of an action sequence $\pi = \langle a_1, \ldots, a_n\rangle$, starting from a given* initial state *$s_0 \in S$.*

Given an action model that defines the preconditions and effects of a set of actions, we say that a trajectory $\tau = \langle s_0, a_1, s_1, \ldots, a_n, s_n\rangle$ is *valid* iff, for each $1 \leq i \leq n$, it holds that 1) a_i is applicable in s_{i-1}, i.e., s_{i-1} satisfies the precondition of a_i, and 2) s_i is the state that follows s_{i-1} after applying the effects of a_i. Trajectories are generated by the execution of deterministic actions as they appear in *classical planning*, e.g. when executing a sequential plan to reach the aimed goals (Geffner & Bonet, 2013); or non-deterministic actions as in *MDP solving*, e.g. when executing a policy (Mausam, 2012), or in RL as the output of a learning episode (Sutton & Barto, 2018). Figure 4 shows a trajectory of the *Blocksworld* domain associated to the execution of the six-action sequential plan $\pi = \langle unstack(b1, b2), putdown(b1), unstack(b2, b3), stack(b2, b1), pickup(b3), stack(b3, b2)\rangle$ to invert the 3-block tower of Figure 1. In the example, states are represented following the *closed world* notation of PDDL.

When learning declarative action models, it is not always realistic to assume that trajectories are fully observable. Observations of trajectories are more commonly used in the literature as learning examples. In theory, observations of trajectories may be noisy and/or intermittent, that is, not every state/action of a trajectory is always observable. In practice, most approaches to learning declarative actions models adopt the following two assumptions: (i) the initial state of the trajectory is fully observed, and (ii) the sequence of actions in the trajectory is fully observed. This restrictive view of the observations entails that the set of world objects, the set of state variables, and the actual length of the true trajectory, are actually known.

Definition 8 (State observation) *An state observation $s^o = \langle x_1 = v_1, \ldots, x_n = v_n\rangle$, $n \leq |X|$, is a partial (or empty) assignment over the state variables X.*

Definition 9 (Observation sequence) *An observation sequence $obs = \langle s^o_0, a_1, s^o_1, \ldots, a_m, s^o_m\rangle$ is an interleaved sequence of state observations and actions.*

To illustrate these definitions, Figure 5 shows a learning example that is obtained from observing the trajectory of Figure 4. In this example, all the executed actions, as well as the initial and final states of the trajectory, were fully observed, but no intermediate state was observed. The example follows the *closed world* representation for the initial and final states of the trajectory.

```
(trajectory (:objects b1 b2 b3)
(:state (clear b1) (handempty) (on b1 b2) (on b2 b3) (ontable b3))
(:action (unstack b1 b2))
(:state (clear b2) (holding b1) (on b2 b3) (ontable b3))
(:action (putdown b1))
(:state (clear b1) (clear b2) (handempty) (on b2 b3) (ontable b1) (ontable b3))
(:action (unstack b2 b3))
(:state (clear b1) (clear b3) (holding b2) (ontable b1) (ontable b3))
(:action (stack b2 b1))
(:state (clear b2) (clear b3) (handempty) (on b2 b1)  (ontable b1) (ontable b3))
(:action (pickup b3))
(:state (clear b2) (holding b3) (on b2 b1) (ontable b1))
(:action (stack b3 b2))
(:state (clear b3) (handempty) (on b3 b2) (on b2 b1) (ontable b1)))
```

Figure 4: Example of a trajectory from the *Blocksworld* that inverts a 3-block tower starting from the state depicted by Figure 1.

```
(example (:objects b1 b2 b3)
(:state (clear b1) (handempty) (on b1 b2) (on b2 b3) (ontable b3))
(:action (unstack b1 b2))
(:action (putdown b1))
(:action (unstack b2 b3))
(:action (stack b2 b1))
(:action (pickup b3))
(:action (stack b3 b2))
(:state (clear b3) (handempty) (on b3 b2) (on b2 b1) (ontable b1)))
```

Figure 5: Learning example from observing the trajectory of Figure 4.

While observation sequences are the most commonly used learning examples, we can find a few other types in the literature. Some learning systems use tuples with the pre-state and post-state of an executed action (Mourão, Zettlemoyer, Petrick, & Steedman, 2012). This format of learning example fits one-length observation sequences.

Most distinguishable is the use of graphs that encode a fragment of the transition system, which can be understood as comprising several trajectories with some shared segments (Bonet & Geffner, 2020). Our definitions can be straightforwardly extended to accommodate non-sequential learning examples as the ones represented in a graph.

2.3 The Error Function

This section analyzes different *error functions* for guiding the combinatorial search in the hypothesis space of the possible declarative action models, and for evaluating the quality of the learned action models.

2.3.1 DATA-BASED ERROR FUNCTIONS

A *data-based error function*, denoted as $\delta(obs, M)$, is a scalar function that assesses how qualified the model M is to generate a trajectory that comprises the observation obs. The computation of $\delta(obs, M)$ generally comprises the following two steps:

1. Synthesize with M a trajectory τ whose observation might be obs.

2. Compute the error of τ with respect to obs.

In practice it is usually convenient to compute both the trajectory τ, and its corresponding error with respect to *obs*, in a single integrated step. The complexity of computing $\delta(obs, M)$ is given by the complexity of computing the trajectory τ. The simplest scenario arises when the action model M is deterministic and the observations *obs* provide full knowledge of (i), the initial state of the trajectory and (ii), the sequence of actions in the trajectory. Under these widely adopted assumptions, the trajectory τ can be computed in linear time by executing the given sequence of actions, starting from the given initial state, and following the model M. This is the reason why many learning systems in the literature impose such restrictive assumptions on the learning examples.

More challenging scenarios appear when we relax any of the two assumptions described above. In this case, computing a trajectory τ consistent with *obs* (i.e. such that τ includes all actions and state observations of *obs*) could require a combinatorial search process. Uncertainty about the initial state, missing actions, and an unknown bound for the length of τ, are all factors that increase the complexity of the search process. A computationally cheaper solution, but potentially less informative, is to calculate an approximation/relaxation of a valid trajectory. For instance, leveraging the *relaxed plan* formalism, widely used in automated planning, that can be computed in polynomial time (Bonet & Geffner, 2001).

Once we have a trajectory τ, there exist different approaches to quantify the error of τ with respect to the input observation sequence obs:

- Counting the errors found in τ measured as the number of *unsatisfied preconditions* and *redundant effects* (Yang et al., 2007). This method is generally used when obs contains a complete action sequence π, and τ is just the result of reproducing π with M.

- Computing the cost of fixing the trajectory τ to make it consistent with obs. This can be achieved using *edit* operations that *add* or *remove* ground state variables from the trajectory (Kucera & Barták, 2018).
- Likelihood functions (Aineto et al., 2020) and coverage metrics (Mourão et al., 2012) to asses the fitness of a model for a given input set of learning examples.

The computation of $\delta(obs, M)$ can also be re-interpreted with respect to an auxiliary action model M' such that: (i) M' is obtained by applying *edit operations* to the input model M, and (ii) M' is able to generate a valid trajectory τ that is consistent with the observation sequence obs. This model-centred reinterpretation of the error function is appealing because it allows to quantify the quality of an action model even if it cannot generate a trajectory consistent with obs. This reinterpretation is also less sensitive to flaws in the action model that repeatedly manifest in the learning examples; flaws in the action model are corrected only once, instead of correcting every flaw manifestation in the learning examples (Aineto et al., 2019).

2.3.2 STRUCTURAL ERROR FUNCTIONS

Structural error functions are a different kind of information source about the quality of an action model M. These functions are simply denoted as $\delta(M)$ because they only consider the structure of the actions in the action model, regardless the learning examples. Structural error functions represent regularization penalties, to indicate preference of simpler action models over more complex ones (e.g. when the number and signatures of the action schemes are unknown), or to establish initial preferences for certain action models (i.e. priors on the possible action models). With the aim of defining tractable hypothesis spaces, structural error functions are also useful to bound the size of the search space to a maximum complexity (e.g. action models that contain up to k preconditions/effects, or action models that differ up to k preconditions/effects from a given solution sketch). Examples of structural functions used in the literature are:

- Complexity metrics that measure purely structural features of the action models to give more priority to simpler models. For instance, features such as the number of schematic state variables, their arity, or the number of action schemes and their number of parameters. (Pasula et al., 2007a; Aineto et al., 2019; Bonet & Geffner, 2020; Suárez-Hernández, Segovia-Aguas, Torras, & Alenyá, 2021).
- Weights for expressing preferences on the different formulae that can appear in the preconditions and effects of actions (Yang et al., 2007).

Last but not least, the information source of structural functions can also be used in combination with the previous data-based error functions e.g., for tie-breaking or as a weighted sum (Pasula et al., 2007a).

3. Learning Paradigms

This section describes three different paradigms for the learning of declarative action models. Each learning paradigm corresponds to one classical AI task, namely *constraint satisfaction*, *optimization*, and *constraint propagation*. We formalize these three tasks and explain how they map to the learning of declarative action models.

3.1 Classical Problem Solving Tasks

We define a **constraint satisfaction** problem as a tuple $\langle X, D, C\rangle$ where,

- X is a finite and non-empty set of variables.
- D represents the domain of these variables, such that $D(x)$ is the particular domain for a given variable $x \in X$.
- C is a set of constraints that bounds the possible values of the variables in X.

A solution to a constraint satisfaction problem is a full assignment over the variables X that is consistent with the input set of constraints C. Without a notion of solution quality, different variable assignments that are consistent with C, i.e., different solutions, are equally possible and valid. The family of the constraint satisfaction problems includes SAT, CSP, and SMT problems. Each one of them defines a different type of domain of variables (e.g., Boolean domain in the case of SAT, finite-domain for CSPs, or integer and real domains in the case of SMTs) as well as different types of constraints (e.g. propositional logic formulae, first-order logic formulae, arithmetic-logic expressions).

An **optimization** problem is also defined over a set of variables X, with domain in D and, optionally, over a set of constraints C. Optimization problems additionally require the input of an *objective function* $f(X)$, a scalar function defined over X. A solution to an optimization problem is an assignment of the variables that minimizes the given objective function. By convention, minimization is the standard definition of the optimization problem (maximization problems are straightforward formulated negating the objective function). Optimization problems are also categorized according to the kind of the domain of the variables, typically in a continuous or discrete domain, the kind of constraints that are defined over the variables (e.g. linear, quadratic, non-linear), and the kind of the objective function (e.g. convex, non-convex).

A **constraint propagation** problem is also defined with respect to a set of variables X, with domain in D, and a set of constraints C. The solution to a constraint satisfaction problem is not a single value assignment to the variables in X, but the set of all possible value assignments that are consistent for X accordingly to C. Given a variable $x \in X$, the subset $R(x) \subseteq D(x)$ that is consistent with C is called the *reduced domain* of that variable. If constraint propagation causes an empty reduced domain, for any variable in X, this means that the constraint propagation problem has no solution. Constraint propagation is an umbrella term that actually refers to a large family of tasks under different names. This family of tasks ranges from *logical filtering*, for the setting where X are Boolean variables and constraints in C are logic formulae, to *belief propagation/update/tracking* where variables are scalar and reduced domains are specified as a probability distribution.

3.2 Learning Declarative Action Models as Classical Problem Solving

Roughly speaking, the mapping between the task of learning a declarative action model and the problem solving tasks described in the preceding section is established as follows: (i) X are the variables required for encoding the hypothesis space, so a full variable assignment over X represents a possible action model; (ii) C is the set of syntactic and semantic constraints that a solution action model must satisfy (semantic constraints may include the input set of learning examples); and the objective function $f(X)$, which is a function that evaluates the error of the learned action model with respect to the given set of learning examples.

We formalize the **task of learning a declarative action model** from observations of action executions as a tuple $\langle Obs, \mathcal{M} \rangle$, where:

- Obs is a **set of learning examples**, such that each learning example $obs \in Obs$ is an observation sequence $obs = \langle s_0^o, a_1, s_1^o \ldots, a_m, s_m^o, \rangle$.
- $\mathcal{M}$ is the **hypothesis space**, the set of all possible action models that can be learned.

3.2.1 LEARNING AS SATISFIABILITY

In the *learning as satisfiability* paradigm, a solution to the learning task $\langle Obs, \mathcal{M} \rangle$ is an action model $M^+ \in \mathcal{M}$ such that, for every learning example $obs \in Obs$, M^+ can produce a trajectory consistent with obs. The solution action model M^+ is formalized in terms of an error function $\delta(obs, M)$ that computes the editions required by M to generate a trajectory consistent with obs.

Definition 10 (Solution model M^+) *Let $M^+ \in \mathcal{M}$ be a solution to the learning task $\langle Obs, \mathcal{M} \rangle$. M^+ is a solution generated by a learning as satisfiability approach iff $\forall_{obs \in Obs} \delta(obs, M^+) = 0$.*

An error $\delta(obs, M^+) = 0$ means that the model M^+ is able to generate a trajectory that is consistent with the learning example $obs \in Obs$. A characteristic property of this learning paradigm is that all different solution models $M_1^+, \ldots, M_n^+$ satisfy the learning examples, and that no ranking function is specified to prioritize some solution over others.

If a maximum length bound is set for the trajectories that produced the learning examples (for instance because the length of the learning example matches that of the trajectory), then *learning as satisfiability* can be compiled into a constraint satisfaction problem (Bonet & Geffner, 2020; Garrido & Jiménez, 2020). Otherwise, if the maximum length of the trajectories is unknown, the learning task can be compiled into a planning task, but at the cost of increasing the learning complexity (Aineto et al., 2019).

The main strength of the *learning as satisfiability* paradigm is the maturity of SAT, CSP, and SMT solvers. However, this learning paradigm faces limitations to scale up to problems of large size. Constraint satisfaction solvers are sensitive to the size of the input task, and the number of learning examples affects the size of the problem to solve. In addition, the *learning as satisfiability* paradigm cannot handle noisy learning examples. This, however, is not such a limitation for some particular environments like simulator or games wherein noiseless observations of action executions can be guaranteed.

3.2.2 LEARNING AS OPTIMIZATION

In the *learning as optimization* paradigm, a solution to the $\langle Obs, \mathcal{M} \rangle$ learning task is an action model $M^* \in \mathcal{M}$ that minimizes the error $\delta(obs, M)$ accumulated over the input set of learning examples.

Definition 11 (Solution model M^*) *Let $M^* \in \mathcal{M}$ be a solution to the $\langle Obs, \mathcal{M} \rangle$ learning task. M^* is a solution generated by a learning as optimization approach iff it minimizes the accumulated error $\delta(obs, M)$ for all learning examples $obs \in Obs$.*

In terms of a generative approach to inductive learning, the error returned by $\delta(obs, M)$ is definable as the posterior probability $P(obs|M)P(M)$. This generative interpretation allows us to view the *learning as optimization* paradigm as a Bayesian classification problem, where each action model $M \in \mathcal{M}$ is a class (Aineto et al., 2019); an action model M acts as the corresponding class prototype that encapsulates all possible observations of the trajectories synthesized with M (i.e. the set of all the learning examples that belong to that class). Interestingly, this Bayesian classification view allows introducing the prior probability of each action model in the learning task.

The *learning as optimization* paradigm widens the scope of *learning as satisfiability* as it is able to handle noisy learning examples; that is, examples where the values of the observed variables differ from the actual values of the corresponding state variables (Mourão et al., 2012; Zhuo & Kambhampati, 2013). As a rule of thumb, a learning task cast as a *learning as satisfiability* problem can be translated into an optimization problem. In such a case, the problem is conceived as a task that consists in satisfying the maximum number of constraints or maximizing a weighted combination of the satisfied constraints (Yang et al., 2007).

The main drawback of the *learning as optimization* paradigm is the lack of guarantee that a solution model M^* is able to generate trajectories consistent with the learning examples of Obs. Indeed, it is not even guaranteed that M^* is consistent with a single learning example $obs \in Obs$. As a consequence, for deliberative tasks that demand correct and complete action models, M^* may not be useful.

3.2.3 LEARNING AS BELIEF PROPAGATION

This paradigm is understood as a reformulation of the *learning as satisfiability* paradigm, which produced a concrete output model. The aim of the *learning as belief propagation* consists in calculating the set of all the possible action models that are consistent with the input learning examples. We will refer to this solution set as $\mathcal{M}^+ \subseteq \mathcal{M}$.

Definition 12 (Solution set $\mathcal{M}^+$) *A set of action models $\mathcal{M}^+ \subseteq \mathcal{M}$ is a solution to the $\langle Obs, \mathcal{M}\rangle$ learning task generated by a belief propagation approach iff for every $M^+ \in \mathcal{M}^+$ it holds that $\forall_{obs \in Obs}\delta(obs, M^+) = 0$, and there is no $M \in \{\mathcal{M} \setminus \mathcal{M}^+\}$ such that $\delta(obs, M) = 0$.*

The *learning as belief propagation* paradigm is related to *logical filtering* (Shahaf & Amir, 2006; Amir & Chang, 2008; Shirazi & Amir, 2011) and *belief tracking* (Bonet & Geffner, 2014) since these formalisms update a belief state (a set of possible world states) with a sequence of executed actions and/or perceived observations. In the task of learning a declarative action model, beliefs represent subsets of the hypothesis space that are consistent with the processed learning examples, and this subset is updated as new learning examples are provided. In this setting the learning task is solved first for a single learning example and then, in an online fashion, the set of models $\mathcal{M}^+$ that result from this operation is used as the new hypothesis space for the subsequent learning example. This procedure is repeated until all the input learning examples are processed. When the procedure ends, the resulting set of actions is the solution set $\mathcal{M}^+$.

Interestingly, subsets of the hypothesis space can be compactly represented with action schemes whose preconditions/effects include disjunctive formulae. This kind of action schemes are also called in the literature *non-deterministic*, or *partially specified*, action schemes (Zhuo, Nguyen, & Kambhampati, 2013). With this regard, *learning as belief propagation* can be understood as transitioning from non-deterministic to deterministic action models, incrementally increasing the amount of determinism. This approach has been successfully implemented using *dynamic epistemic logic* for learning different types of propositional action models (Bolander & Gierasimczuk, 2018).

One advantage of the *learning as belief propagation* paradigm is that it is able to find all the action models consistent with the learning examples. Another important benefit of this paradigm is that the set of solutions $\mathcal{M}^+$ is incrementally updated by individually processing the learning examples and hence, its computational complexity is not exponentially dependent on the size of the set of learning examples.

The main weakness of this learning paradigm is that, for the time being, it has only been successfully implemented at settings where learning examples are noise-free and include the full sequence of executed actions (partial state observations are, however, optional).

The multiple-model output of this approach offers interesting extensions as, for instance, calculating the probability distribution of the possible models. Similar approaches are traditionally followed in probabilistic belief tracking (Bonet & Geffner, 2016), and in the context of *Hidden Markov Models* (HMMs), to compute the *belief state* that result from processing a sequence of observations (Ghahramani, 2001).

4. Implementations

This section reviews relevant systems (found in the AI literature for the 15-year period 2005-2020) that implement the learning of declarative action models. The review is focused on systems that learn action schemes that generalize to different number (and identity) of the world objects. Each system is examined according to these four dimensions:

1. Hypothesis space of possible action models.
2. Typology of the input learning examples.
3. Error function used to evaluate the quality of the action models.
4. Learning paradigm.

ARMS

The *Action-Relation Modelling System* (ARMS) (Yang et al., 2007) is one of the pioneering works in modern action model learning and it has become a popular baseline to evaluate the state-of-the-art. Briefly, ARMS compiles the task of learning STRIPS action models into a weighted MAX-SAT problem.

1. **Hypothesis space:** The hypothesis space is the set of possible STRIPS action schemes that can be built with the given schematic state variables and action signatures.
2. **Learning examples:** Each learning example comprises the *initial* and *final* states, the sequence of actions executed to achieve the final state from the initial state, and optionally, partial observations of the intermediate states. ARMS is one of the first systems to handle partial (or null) observations of intermediate states.
3. **Error functions:** ARMS defines an *error metric* that, given an action model and a learning example, counts the number of unsatisfied preconditions. In addition, ARMS defines a *redundancy metric* that penalizes unnecessary effects.

4. **Learning paradigm:** ARMS poses the learning problem as an optimization task that minimizes the error and redundancy metrics described above with the aim to find correct and concise models. In more detail, ARMS builds a weighted propositional satisfiability problem and solves it with an off-the-shelf MAX-SAT solver. Three types of constraints are considered: i) constraints imposed by the STRIPS syntax, ii) constraints extracted from the distribution of actions in the learning examples, and iii) constraints obtained from the observed traversed states.

SLAF

Simultaneous Learning and Filtering (SLAF) (Amir & Chang, 2008) is another system precursor of the modern algorithms for action-model learning. The main innovative aspect of SLAF is that it is the first system able to learn all action models that are consistent with the learning examples.

1. **Hypothesis space:** The hypothesis space is the set of possible STRIPS action schemes extended with universal quantifiers in the effects that can be built with the given schematic state variables and action signatures.

2. **Learning examples:** Each learning example comprises the full sequence of executed actions and partial observations of every traversed state.

3. **Error function:** All solution models must be consistent with the entire set of learning examples.

4. **Learning paradigm:** SLAF implements the *constraint propagation* approach; first, SLAF builds a CNF formula representing a belief state of the set of possible action models (the hypothesis space). Then SLAF updates, in an online fashion, the CNF formula with every action and observation in the given set of learning examples; removing from the belief state any action model that is inconsistent. The formula returned at the end defines the set of all consistent action models. Concrete models can be extracted from the returned formula using a SAT solver.

LAMP AND AMAN

Here we review two approaches strongly connected to ARMS and that extend its scope in two different directions. *Learning Action Models from Plan traces* (LAMP) (Zhuo, Yang, Hu, & Li, 2010) extends ARMS expressiveness to learn action models with universal and existential quantifiers, as well as with logical implications.

1. **Hypothesis space:** The hypothesis space is a subset of the possible ADL action models (Pednault, 1994) that can be built using: preconditions with logical implications, and conditional effects with universal and existential quantifiers, defined over the given action signatures and schematic state variables.

2. **Learning examples:** Learning examples comprise the initial and final states, the exact sequence of executed actions, and partial observations of the traversed states.

3. **Error function:** LAMP defines a Weighted Pseudo Log-Likelihood score that measures the likelihood of all weighted candidate formulas for a given set of learning examples.

4. **Learning paradigm:** Like ARMS, LAMP also implements the *learning as optimization* paradigm. In this case the weights of the candidate formulas that encode the action models (essentially, the alphabet) are computed learning a Markov Logic Network from the learning examples. The solution model is given by the formulas that surpass some given threshold.

Action Model Acquisition from Noisy plan traces (AMAN) (Zhuo & Kambhampati, 2013). The AMAN system is able to learn STRIPS action models when the actions in the learning examples have a probability of being observed incorrectly.

1. **Hypothesis space:** The hypothesis space is the set of STRIPS action models consistent with the given action signatures and state variables.
2. **Learning examples:** Each learning example comprises the initial and final states, a sequence of possibly noisy actions, and partial observations of the intermediate states.
3. **Error function:** AMAN defines a reward function defined in terms of the percentage of actions successfully executed and the percentage of goal propositions achieved after the last successfully executed action.
4. **Learning paradigm:** AMAN also follows the *learning as optimization* paradigm. AMAN builds a graphical model to capture the relations between states, executed actions, observed actions, and the action models. The parameters of the graphical model are learned computing, at the same time, the probability distribution of each candidate action model.

ALICE

Action Learning In Complex Environments (ALICE) (Mourão, Petrick, & Steedman, 2010; Mourão et al., 2012) is a system for learning STRIPS action schemes that can handle noisy state observations.

1. **Hypothesis space:** The hypothesis space is the set of possible STRIPS action schemes that can be built with the given schematic state variables and action signatures.
2. **Learning examples:** A learning example is a state transition (i.e. a pair of *pre-state* and *post-state*) labeled with the corresponding executed action. The provided state observations, of the pre-state or the post-state, may be noisy.
3. **Error function:** ALICE defines a coverage metric that penalizes action models that are inconsistent with the state transitions of an input set of learning examples.
4. **Learning paradigm:** For each action signature, ALICE learns its corresponding effects as a set of Boolean classifiers; ALICE follows the *learning* as optimization paradigm since these classifiers are learned by maximizing the coverage of the learning examples. Each classifier determines whether a particular schematic state variable is set `True`/`False` by the action execution. The classifiers implemented by ALICE were *kernelised voted perceptrons*. To train the classifiers, ALICE filters the learning examples, so they refer exclusively to the fixed set of objects mentioned in the parameters of the executed actions. ALICE extracts the STRIPS action effects from the parameters of the learned classifiers. Preconditions are derived later, at a post-processing procedure. This procedure rejects candidate preconditions that make either precision or recall drop on the input set of learning examples.

LOUGA

Learning Operators Using Genetic Algorithms (LOUGA) (Kucera & Barták, 2018), leverages a genetic algorithm to learn the effects of STRIPS action schemes. Each gene encodes, for each action scheme, whether a particular schematic state variable is a positive effect, negative effect, or none.

1. **Hypothesis space:** The hypothesis space is the set of possible STRIPS action schemes that can be built with the given schematic state variables and action signatures.

2. **Learning examples:** Learning examples comprise the initial and final states, the exact sequence of executed actions, and partial observations of the traversed states.

3. **Error function:** The fitness of a given action model is evaluated with a function that counts: i) the number of preconditions not met by the action model, ii) the number of *redundant effects* of the model (i.e., effects that are not used for achieving the precondition of any later action in a learning example) and iii), the number of assignments of state variables, that are observed in the examples, but that are missing in the trajectory generated by the evaluated model.

4. **Learning paradigm:** LOUGA implements the *learning as optimization* approach with a genetic algorithm. The fitness of an individual is evaluated by reproducing the trajectory, with the model encoded in the individual, on the input set of learning examples. Preconditions are derived at a post-processing procedure, they are inferred by tracking literals that hold before the execution of an action.

FAMA

FAMA (Aineto, Jiménez, & Onaindia, 2018; Aineto et al., 2019) extends the *learning as satisfiability* approach to learn STRIPS action schemes when the observability of the executed actions is minimal. In more detail, FAMA does not require knowing the actual sequence of executed actions that produced a given observation sequence. In the general case, FAMA learns from *gapped* sequences of observations such that, between two consecutive state observations, there may be an unbounded number of missing unobserved states and actions. In the extreme case, FAMA can learn from just initial-final state pairs.

1. **Hypothesis space:** The hypothesis space is the set of possible STRIPS action schemes that can be built with the given schematic state variables and action signatures.

2. **Learning examples:** Each learning example comprises an initial and a final state and optionally, a partially observed sequence of executed actions, as well as partial observations of the traversed states. FAMA can then handle learning examples with gapped observation sequences, where unbound segments of the observed trajectory are missing.

3. **Error function:** FAMA implements a variant of the *learning as SAT* approach and hence, it requires that a solution model can produce trajectories that are consistent with all the learning examples.

4. **Learning paradigm:** Implements the *learning as satisfiability* approach with a classical planning compilation, that is solved by an off-the-shelf planner. A solution plan for the classical

planning task that results from the compilation determines the schematic variables that appears in the precondition/effects of each action scheme, as well as it completes the gapped sequences of observations.

LEARNING SYMBOLIC MODELS OF STOCHASTIC DOMAINS

This is one of the first attempts to learn declarative action models with probabilistic effects (Pasula et al., 2007a). We named this approach LPROB.

1. **Hypothesis space:** The hypothesis space is the set of possible STRIPS action schemes, with probabilistic effects, that can be built with the given sets of schematic state variables and action signatures.

2. **Learning examples:** Examples are sets of state transitions given as fully observed *(pre-state, action, post-state)* tuples.

3. **Error function:** This approach defines a hybrid score that rewards action models that maximize the likelihood of the model to produce the given learning examples. At the same time, the score penalizes action models with complex structure (i.e. to prefer compact action models).

4. **Learning paradigm:** LPROB implements the *learning as optimization* paradigm, with a greedy search in the space of the possible action models, guided by the previous hybrid score. LPROB structures the search space hierarchically, identifying two self-contained subproblems: learning the structure of the action models (preconditions and effects), and learning the probabilities associated to the non-deterministic action effects.

LOCM

Learning Object-Centred Models (LOCM) (Cresswell, McCluskey, & West, 2009; Cresswell et al., 2013), is possibly the most distinctive of the previous examined systems, due to its ability of learning declarative action models without knowing the actual state variables. LOCM requires however knowing the full sequence of executed actions.

1. **Hypothesis space:** The hypothesis space is the set of STRIPS action schemes that are consistent with the action signatures of the given sequences of actions.

2. **Learning examples:** A learning example in LOCM is a fully observed sequence of executed actions, i.e. a sequential plan where no gaps with missing actions are allowed.

3. **Error function:** A solution model must be able of producing trajectories consistent with all the given learning examples.

4. **Learning paradigm:** LOCM follows the *learning as satisfiability* paradigm but it overcomes the lack of available state information making reasonable assumptions about the structure of actions. LOCM assumes that objects found in the same position in the parameters of an action belong to the same class (*sort*), and that these objects share a set of possible states that is captured by a parameterized Finite State Machine (FSM). LOCM2, extends this approach to a wider range of domains, by using multiple FSMs to represent the states of the objects

that belong to a class (Cresswell & Gregory, 2011). LOP (LOCM with Optimized Plans), the last contribution of the LOCM family, addresses the problem of inducing preconditions that correspond to static predicates (Gregory & Cresswell, 2016). LOP applies a post-processing step, after the LOCM analysis, that requires that learning examples are optimal sequences of action executions.

LEARNING FIRST-ORDER ACTION MODELS FROM THE STRUCTURE OF THE STATE SPACE

We call this approach LSSS, since it learns declarative action models from the structure of the state space (Bonet & Geffner, 2020). Like LOCM, this approach does not require knowing the actual set of state variables. Inputs are however different, since LSSS learns from one (or more) labeled directed-graph, that encodes the structure of the complete state-space of a small problem instance (or several small problem instances).

1. **Hypothesis space:** The set of possible STRIPS action models that can be built with a maximum number of action signatures, maximum arity for these actions, a maximum number of schematic state variables, and maximum arity for these variables. These bounds are parameters of LSSS.
2. **Learning examples:** A learning example is a labeled directed-graph that encodes the structure of the complete state-space of a small problem instance. Each graph node is a black-box representation of a ground state, while labels in the edges indicate the name of the action that produced the corresponding state transition.
3. **Error function:** This approach guarantees that the solution action model is consistent with all the transitions in the learning examples.
4. **Learning paradigm:** LSSS implements the *learning as satisfiability* approach with a two-level combinatorial search where:
   - The *outer level* searches for the simplest values for the LSSS parameters (number and arity of the action signatures and schematic state variables).
   - *The inner level* computes a set of STRIPS action schemes that is consistent with the observed labeled graphs and given input parameters. In addition, the SAT formulation also produces the simplest set of classical planning problems that are consistent with the input labeled graphs, given a maximum number of objects as another input parameter.

Figure 6 shows a summary table that compares all the reviewed systems. The table evidences that the *error function* and the *learning paradigm* are coupled dimensions; satisfaction approaches and propagation approaches require that the learned action models cover the full set of learning examples. Next we briefly summarize the main similarities and differences of the reviewed systems. ARMS learns STRIPS action schemes with a MAXSAT compilation that implements the *learning as optimization* paradigm. SLAF follows a different learning paradigm and implements a SAT encoding that compute all the action models consistent with the input examples. LAMP extends the ARMS approach to learn models with quantifiers and logical implications, AMAN extends it to deal with noisy learning examples. LOUGA follows the optimization approach of ARMS but implements this approach with genetic algorithms. ALICE allows to learn from noisy examples leveraging off-the-shelf ML tools. To do so, ALICE uses a *deictic* state representation that exclusively focuses on

	Hypothesis space	Learning examples	Error function	Paradigm
ARMS	STRIPS	Full plan + Partial trajectory	Incorrect precs/effs	Optimization
SLAF	STRIPS	Full plan + Partial trajectory	Coverage	Propagation
LAMP	ADL	Full plan + Partial trajectory	Likelihood	Optimization
AMAN	STRIPS	Noisy plan + Partial trajectory	Likelihood	Optimization
ALICE	STRIPS	(Pre-state, Action, Post-state)	Incorrect precs/effs	Optimization
LOUGA	STRIPS	Full plan + Partial trajectory	Incorrect precs/effs	Optimization
FAMA	STRIPS	Partial trajectory	Coverage	Satisfaction
LPROB	Prob. STRIPS	(Pre-state, Action, Post-state)	Likelihood	Optimization
LOCM	STRIPS	Full plan	Coverage	Satisfaction
LSSS	STRIPS	Structure of the state space	Coverage	Satisfaction

Figure 6: Summary table that briefly compares all the reviewed systems.

the objects that are relevant for each executed action. FAMA extends the *learning as satisfiability* approach to learn from examples that have an unbound number of missing executed actions. LPROB is the only system that can learn action schemes with probabilistic effects. All the previous systems required knowledge about the set of state variables, LOCM is able to learn declarative action models exclusively from sequential plans. Last but not least, LSSS follows a similar approach but unlike LOCM, state transitions are only labeled with the name of the action that produced the transition. LOCM required transitions to be labeled with the corresponding ground action (i.e. the action name plus the corresponding ground parameters).

5. Applications and Open Challenges

In this section we first discuss some successful applications that include a core component for learning declarative action models. Subsequently, we summarize some open challenges in the learning of declarative action models.

The current quest for explainable AI systems (Miller, 2019) evidences the need for declarative action models. As a rule of thumb, any AI system in which there is human-in-the-loop interaction, is susceptible of requiring a declarative action model. Plans, policies, explanations, or counterfactuals, are all built with respect to model-based dynamics of a transition system. If these models are to be communicated to humans, they should be compact, general, and object-oriented, like declarative action models. A neat example is the work on *model-reconciliation* (Sreedharan, Chakraborti, & Kambhampati, 2017) which relies on STRIPS actions schemes to handle human-in-the-loop scenarios, where autonomous robots have (and learn) domain and task models that may differ from the humans.

Many different types of applications have brought to light the need of learning action models, specifically in applications that lack a formal description of a transition model or when a translation between different formalisms is needed. In this sense, learning declarative action models has been applied to induce the rules of board games such as *Peg Solitaire* and *Checkers* (Gregory, Schumann, Björnsson, & Schiffel, 2015). Interestingly, this work learns the interactions between the pieces in the games from example sequences of the moves made in playing those games. Another successful field of application of action model learning is narrative generation for story telling (Lindsay, Read, Ferreira, Hayton, Porteous, & Gregory, 2017b; Hayton, Porteous, Ferreira, & Lindsay, 2020). Differently from the previous applications, the examples in narrative generation are natural language descriptions of activity sequences. Learning declarative action models has also been successfully applied to web services for automatic web-service composition (Walsh & Littman, 2008a) or in the acquisition of social rules for multi-agent coordination (Nir, Shleyfman, & Karpas, 2020).

Declarative action models are also useful to compactly represent the update rules of *cellular automata* (Sanner, 2010; Hoffmann, Fates, & Palacios, 2010). Cellular automata have many real-world applications such as modeling the propagation dynamics of wildfires and viruses, modeling fluids dynamics, population growth, and more (Wolfram, 2002). The learning of declarative action models is strongly connected to the ambitious aims of *general problem solving*, that is studied since the early days of AI. Computing a *generalized policy*, that is able to solve an infinite family of problems with some common structure, can be understood as a problem of learning a declarative action model (Francès et al., 2021); more specifically, as a problem of learning the extra preconditions of a given declarative action model so that it exactly captures the situations where actions should be applied. *Generalized policies* are a powerful formalism for solution representation that have been applied to compute solutions for sets of planning problems (Jiménez et al., 2019), as well as for MDP/POMDP solving (Fern, Yoon, & Givan, 2006; Bonet, Palacios, & Geffner, 2010).

Most notably, autonomous car driving emerges as one of the most exciting real-world applications in which action model learning plays a key role. This is particularly relevant for vehicles that learn how to drive from sensor data and human driving (Schwarting, Alonso-Mora, & Rus, 2018), with interesting initiatives that focus on teaching autonomous car how to drive out of data collected from professional drivers[2]. In this regard, mechanisms for action-based representation are exploited for learning a mapping from sensor data directly into driving actions (Xiao, Codevilla, Pal, & Lopez, 2020).

Last but not least, a promising application of the learning of declarative action model is the development of AI systems for the automatic discovery of scientific knowledge (Langley, 2000; Džeroski, Langley, & Todorovski, 2007). Scientific discovery requires declarative models to describe the phenomena to be explained, as well as to encode prior theories obtained from earlier stages of the discovery process. With this regard, scientists must define first the basic entities and then, tackle the discovering of laws that characterize the behavior of these entities. This process strongly resemblances the learning of declarative models from raw sensor data.

The learning of declarative action models is then a well-studied problem, with interesting applications, formal contributions that analyze theoretical aspects of the learning task – for instance, via the notion of sample complexity (Walsh & Littman, 2008b; Stern & Juba, 2017); and with several systems that succeed to learn different action models from diverse types of input knowledge (Jiménez et al., 2012; Arora et al., 2018). However, there is still work to be done to achieve mature tools that are straightforward applicable to more real-world problems. Some of the **open challenges** we identify in this section, like the need for effective exploration algorithms or the synthesis of effective abstract representations, are not exclusive of the learning of action models as they also appear in other AI tasks. Following we summarize some of the most relevant open challenges in the learning of declarative action models:

- **Expressive action models:** Most of the reviewed systems put the focus on learning action models whose preconditions/effects are represented as logic formulae over Boolean state variables.

2. `https://humandrive.co.uk/`

  - *Procedural action models.* The reviewed systems disregard the order in which the preconditions and effects of the actions schemes are validated. For many problems, however, preconditions and effects of actions are more naturally defined as procedures. For instance, in the chess game, the possible moves of a knight are naturally defined as follows: *the knight moves two squares horizontally and then one square vertically*, or *it moves one square horizontally and then two squares vertically*. More examples of procedural structures for action modeling are LTL preconditions represented as finite automata (Baier & McIlraith, 2006), sequence of effects executed in a particular order that may also include execution flow structures for branching and looping (Segovia-Aguas, Jiménez, & Jonsson, 2019), or hierarchies (Gopalakrishnan, Muñoz-Avila, & Kuter, 2018; Segovia-Aguas, Jiménez, & Jonsson, 2018).
  - *Hybrid action models:* Real-world applications such as robotics or autonomous vehicles demand learning of hybrid representations, where state variables are discrete or continuous (Xu & Laird, 2011), and time functions can also be continuous (Ramirez, Papasimeon, Benke, Lipovetzky, Miller, & Pearce, 2017). For instance, the equation for *linear movement with constant acceleration*, $x(t) = x_0 + v_0t + \frac{1}{2}at^2$, where $x(t)$ indicates position at time t, x_0 the initial position, v_0 the initial velocity, and a the acceleration. Defining expressive arithmetic-logic preconditions/effects that constrain the value of hybrid state spaces can easily yield intractable hypothesis spaces. The computation of candidates of tractable subsets of the hypothesis space, e.g. by leveraging input observations as in *regression tree* learning, seems a promising approach to effectively handle these hypothesis spaces.

- **Representative learning examples:** Learning complete action models requires learning examples that are representative of the state space. The collection of representative sets of learning examples depends on the following factors:

  - *Complexity of the state space:* Ideally, representative examples are obtained by collecting all the different state trajectories that are generated when traversing the entire state space (Bonet & Geffner, 2020). Unfortunately the size of a factored state space grows exponentially with the number of state variables and size of their domain. As the number of world objects grows, generating the entire set of trajectories becomes intractable. To illustrate this, the state space of a 3-block *blocksworld* problem contains thirteen states, and grows to above five hundreds states for the 5-blocks case (Slaney & Thiébaux, 2001). Besides the size, the topology of the state space also affects the gathering of representative learning examples. It is easier to obtain representative examples in strongly connected state spaces than in spaces with non-reversible actions and dead-ends. While the theoretical efficiency of ML and RL algorithms has been widely studied in terms of *sample complexity*, most of the work in the learning of declarative action models report only empirical results on this issue. Nevertheless, the sample complexity, for learning STRIPS action schemes in fully observable environments has been successfully characterized in terms of *plan-prediction mistakes* (Walsh & Littman, 2008b). This work requires having a pool of solvable planning problems, which is a too strong assumption for many scenarios. Further research is needed for bounding sample complexity when examples are not fully observable, are intermittent, or noisy.

  - *Exploration algorithms:* Classical blind search algorithms can be used for the systematic exploration of a given state space but they present limitations. *Breadth First Search* requires exponential memory and so it is limited to small state spaces; *Depth First Search* can handle larger state spaces but it cannot guarantee exploration diversity; and *Iterative Deepening* provides exploration diversity with low memory consume but it still demands exponential running time. *Novelty-based* algorithms are an appealing family of systematic exploration algorithms since they determine the states to be explored according to their newness. What is more, polynomial running time algorithms, like the IW(1), gracefully scale-up with the number of objects (Lipovetzky & Geffner, 2012; Lipovetzky, Ramirez, & Geffner, 2015). Stochastic sampling, like the Monte-Carlo exploration algorithms commonly used in RL, may require too much sampling to reach portions of the state space that are only accessible after following a particular sequence of actions. RL is known to suffer limitations in scenarios where rewards are scarce, such as tasks where rewards are only provided at final goal-states (common in automated planning (Icarte, Waldie, Klassen, Valenzano, Castro, & McIlraith, 2019)). An illustrative example is a planning task in the *blocksworld* domain, where no intermediate reward is provided, subgoals are only useful if they are properly sequenced, and computing the right sequence of subgoals is as hard as solving the whole planning problem (Hoffmann, Porteous, & Sebastia, 2004). With this regard, some works on relational RL suggest using different notions of novelty as well as actions implemented with relational count functions (Lang et al., 2012).

- **Realistic learning examples:** Most of the reviewed works on the learning of declarative action models assume that the sequence of executed actions of the observed trajectories is known. This assumption simplifies the learning task because computing the fitness of an action model to an observation sequence can be done in linear time. For many real-world applications this is a too strong assumption. Learning examples are usually collected via sensors that only report the value of some observable state variables at certain time-stamps. It is thus more realistic to assume that input observations correspond to the value of a given set of observable variables. Further, the observable variables usually differ from the actual state variables that rule the actions dynamics. Formalisms like HMMs, which have been successfully applied to numerous real-world problems, follow this approach. In addition, HMMs can also deal with noise and leverage sensor models when available (Aineto et al., 2020). Another unrealistic assumption concerns the kind of available input knowledge. Most of the reviewed systems assume that an abstract state representation is available. Again this is a too strong assumption for many real-world applications, where images (Asai & Fukunaga, 2018; Asai & Muise, 2020), or natural language text, are the only available learning examples (Lindsay, Read, Ferreira, Hayton, Porteous, & Gregory, 2017a; Hayton et al., 2020). The computation of abstract state representations for effective problem solving is actually a core problem in AI that appears in many different forms such as at the computation of *generalized plans* (Lotinac, Segovia-Aguas, Jiménez, & Jonsson, 2016; Bonet, Frances, & Geffner, 2019), *predicate invention* for relational learning (Kok & Domingos, 2007), *feature discovering* from scene labeling (Farabet, Couprie, Najman, & LeCun, 2013) or in RL (Konidaris et al., 2018).

- **Scalable algorithms for action model learning:** Another issue is the lack of learning algorithms that can effectively deal with the satisfaction of diverse hard constraints, the optimization of an objective error function and, at the same time, scale up for large input sets of learning examples. All the reviewed systems (except for SLAF) follow a batch learning approach and, when the set of examples is large enough, the batch approach becomes unfeasible. *Online learning* seems a more promising research direction to scale up to large data sets. Further, the online setting allows to dynamically adapt to new patterns in the learning examples. A popular approach for online learning is using mini-batches, that is, processing a small batch of learning examples at a time. At each step, the learning algorithm starts from the model learned at the prior step. Implementing this approach requires error functions that, besides maximizing the fitness to the given learning examples, they also minimize the changes that need to be applied to the models learned at prior steps (Aineto et al., 2019; Suárez-Hernández et al., 2021).

Last but not least, an important motivation for learning declarative action models is their application in deliberative tasks that compute explanations and counterfactuals (Pearl, 2019). This is the case of tasks such as *plan synthesis*, *intention recognition*, *system diagnosis*, *model-checking*, or *model reconciliation*. The mainstream approaches for addressing these deliberative tasks usually require input action models that are correct and complete. Unfortunately, systems for learning declarative action models hardly guarantee full correctness and completeness of the learned models. Approaches to reasoning under uncertainty, given that they can reason about actions whose effects or preconditions are not known with certainty, can also be useful to close the gap between model-based reasoning and the learning of declarative action models (Kambhampati, 2007).

6. Conclusions

In this paper we have presented a comprehensive scheme for learning declarative action models in model-based transition systems. Our proposal revolves around four axes: the hypothesis space of the action models, the learning examples, the error function used to guide the search, and the learning paradigm.

The hypothesis space of a learning task encompasses all the action models that are learnable, and its size is generally linked to the expressiveness of the target representation language. We have shown that a hypothesis space can be represented with a finite set of variables, named alphabet, and formalized it for STRIPS action models. Nevertheless, this notion can be extended to more expressive action models at the cost of larger alphabets and, therefore, larger hypothesis spaces.

The learning examples constitute the input data to the task of learning a declarative action model. Learning examples are usually formalized in the form of interleaved sequences of states and actions that contain partial information about the states visited by some agent and the actions it took. From this data, it is possible to extract the set of state variables of the domain which give rise to the schematic state variables and in turn to the schematic state constraints that will form the preconditions and the effects of the action schemes. We have also seen that partial observability, noise, and whether or not the length of the true trajectory is known are all features of the learning examples that affect the complexity of the learning task.

The error function is used to guide the search across the hypothesis space. Error functions can be used to characterize what we consider the desired solution. For instance, we may be interested in action models that can reproduce the learning examples without error, or in obtaining the simplest model with the minimum number of preconditions and effects.

Lastly, the learning paradigm determines how the learning problem is formulated, as well as the solver required to find a solution. We have established the connection between the paradigms followed in action model learning and canonical AI tasks, namely constraint satisfaction, optimization and constraint propagation, all of which have strong scientific communities backing the development of powerful solvers that can be directly applied to the learning task. The output of the learning task is also dependant on the learning paradigm, as it dictates whether we will obtain one or several learned models, and whether the learned models will satisfy the learning examples or minimize an error function.

We have shown that the state-of-the-art approaches to the learning of declarative action models fall nicely on the comprehensive framework built upon these four axes. Further, our framework provides analytical grounds to study the advantages and shortcomings of current and future approaches beyond their empirical performance. This analytical assessment also serves to highlight current issues that plague the field of action model learning and to outline future lines of research.

Acknowledgments

This work is supported by the Spanish MINECO project TIN2017-88476-C2-1-R and partially supported by the EU ICT-48 2020 project TAILOR (No. 952215). D. Aineto is partially supported by the *FPU16/03184* and S. Jiménez by the *RYC15/18009*.

References

Abbeel, P., & Ng, A. Y. (2005). Learning first-order markov models for control. *Advances in neural information processing systems*, *17*, 1–8.

Ahmed, O., Träuble, F., Goyal, A., Neitz, A., Wüthrich, M., Bengio, Y., Schölkopf, B., & Bauer, S. (2021). CausalWorld: A Robotic Manipulation Benchmark for Causal Structure and Transfer Learning. In *International Conference on Learning Representations (ICLR-21)*, pp. 3–7.

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

Aineto, D., Jiménez, S., & Onaindia, E. (2019). Learning action models with minimal observability. *Artificial Intelligence Journal*, *275*, 104–137.

Aineto, D., Jiménez, S., & Onaindia, E. (2020). Observation decoding with sensor models: recognition tasks via classical planning. In *International Conference on Automated Planning and Scheduling, (ICAPS-20)*.

Aineto, D., Jiménez, S., Onaindia, E., & Ramírez, M. (2019). Model Recognition as Planning. In *International Conference on Automated Planning and Scheduling, (ICAPS-19)*, pp. 13–21.

Amir, E., & Chang, A. (2008). Learning partially observable deterministic action models. *Journal of Artificial Intelligence Research*, *33*, 349–402.

Ardón, P., Pairet, E., Lohan, K. S., Ramamoorthy, S., & Petrick, R. (2021). Building affordance relations for robotic agents-a review. In *International Joint Conference on Artificial Intelligence, (IJCAI-17)*, pp. 4302–4311.

Arora, A., Fiorino, H., Pellier, D., Etivier, M., & Pesty, S. (2018). A review of learning planning action models. *Knowledge Engineering Review*, *33*.

Asai, M., & Fukunaga, A. (2018). Classical planning in deep latent space: Bridging the subsymbolic-symbolic boundary. In *National Conference on Artificial Intelligence, AAAI-18*.

Asai, M., & Muise, C. (2020). Learning neural-symbolic descriptive planning models via cube-space priors: The voyage home (to strips). In *International Joint Conference on Artificial Intelligence*.

Baier, J. A., & McIlraith, S. A. (2006). Planning with first-order temporally extended goals using heuristic search. In *AAAI Conference on Artificial Intelligence, AAAI-06*, p. 788.

Bergadano, F., & Gunetti, D. (1996). *Inductive Logic Programming: from machine learning to software engineering*. MIT press.

Bolander, T., & Gierasimczuk, N. (2018). Learning to act: qualitative learning of deterministic action models. *Journal of Logic and Computation*, *28*(2), 337–365.

Bonet, B., Frances, G., & Geffner, H. (2019). Learning features and abstract actions for computing generalized plans. In *AAAI Conference on Artificial Intelligence*, Vol. 33, pp. 2703–2710.

Bonet, B., & Geffner, H. (2001). Planning as heuristic search. *Artificial Intelligence*, *129*(1-2), 5–33.

Bonet, B., & Geffner, H. (2014). Belief tracking for planning with sensing: Width, complexity and approximations. *Journal of Artificial Intelligence Research*, *50*, 923–970.

Bonet, B., & Geffner, H. (2016). Factored probabilistic belief tracking. In *International Joint Conference on Artificial Intelligence*.

Bonet, B., & Geffner, H. (2020). Learning First-Order Symbolic Representations for Planning from the Structure of the State Space. In *European Conference on Artificial Intelligence*.

Bonet, B., Palacios, H., & Geffner, H. (2010). Automatic derivation of finite-state machines for behavior control. In *National Conference on Artificial Intelligence, (AAAI-10)*.

Chakraborti, T., Sreedharan, S., Zhang, Y., & Kambhampati, S. (2017). Plan explanations as model reconciliation: Moving beyond explanation as soliloquy. In *International Joint Conference on Artificial Intelligence, (IJCAI-17)*, pp. 156–163.

Cresswell, S., & Gregory, P. (2011). Generalised Domain Model Acquisition from Action Traces. In *International Conference on Automated Planning and Scheduling, ICAPS-11*. AAAI Press.

Cresswell, S., McCluskey, T. L., & West, M. M. (2009). Acquisition of Object-Centred Domain Models from Planning Examples. In *International Conference on Automated Planning and Scheduling, (ICAPS-09)*. AAAI Press.

Cresswell, S. N., McCluskey, T. L., & West, M. M. (2013). Acquiring planning domain models using LOCM. *The Knowledge Engineering Review*, *28*(02), 195–213.

De la Rosa, T., Jiménez, S., Fuentetaja, R., & Borrajo, D. (2011). Scaling up heuristic planning with relational decision trees. *Journal of Artificial Intelligence Research*, *40*, 767–813.

De Raedt, L. (2008). *Logical and relational learning*. Springer Science & Business Media.

De Raedt, L., & Kersting, K. (2017). Statistical relational learning. In Sammut, C., & Webb, G. I. (Eds.), *Encyclopedia of Machine Learning and Data Mining*, pp. 1177–1187. Springer.

Diuk, C., Cohen, A., & Littman, M. L. (2008). An object-oriented representation for efficient reinforcement learning. In *International Conference on Machine Learning*, pp. 240–247.

Džeroski, S., De Raedt, L., & Driessens, K. (2001). Relational reinforcement learning. *Machine learning*, *43*(1-2), 7–52.

Džeroski, S., Langley, P., & Todorovski, L. (2007). Computational discovery of scientific knowledge. In *Computational Discovery of Scientific Knowledge*, pp. 1–14. Springer.

Farabet, C., Couprie, C., Najman, L., & LeCun, Y. (2013). Learning hierarchical features for scene labeling. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *35*(8), 1915–1929.

Fern, A., Yoon, S., & Givan, R. (2006). Approximate policy iteration with a policy language bias: Solving relational markov decision processes. *Journal of Artificial Intelligence Research*, *25*, 75–118.

Fikes, R. E., Hart, P. E., & Nilsson, N. J. (1972). Learning and executing generalized robot plans. *Artificial intelligence*, *3*, 251–288.

Finney, S., Gardiol, N., Kaelbling, L. P., & Oates, T. (2002). The thing that we tried didn't work very well : Deictic representation in reinforcement learning. In *Conference on Uncertainty in Artificial Intelligence*.

Fox, M., & Long, D. (2003). PDDL2.1: An extension to PDDL for expressing temporal planning domains.. *Journal of Artificial Intelligence Research*, *20*, 61–124.

Francès, G., Bonet, B., & Geffner, H. (2021). Learning general policies from small examples without supervision. In *AAAI Conference on Artificial Intelligence*.

Garrido, A., & Jiménez, S. (2020). Learning temporal action models via constraint programming. In *European Conference on Artificial Intelligence*.

Geffner, H., & Bonet, B. (2013). *A Concise Introduction to Models and Methods for Automated Planning*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers.

Getoor, L., & Taskar, B. (2007). *Introduction to Statistical Relational Learning*. The MIT Press.

Ghahramani, Z. (2001). An Introduction to Hidden Markov Models and Bayesian Networks. *Journal of Pattern Recognition and Artificial Intelligence*, *15*(1), 9–42.

Ghallab, M., Nau, D., & Traverso, P. (2004). *Automated Planning: theory and practice*. Elsevier.

Gopalakrishnan, S., Muñoz-Avila, H., & Kuter, U. (2018). Learning task hierarchies using statistical semantics and goal reasoning. *AI Communications*, *31*(2), 167–180.

Grastien, A., Haslum, P., & Thiébaux, S. (2012). Conflict-based diagnosis of discrete event systems: theory and practice. In *International Conference on the Principles of Knowledge Representation and Reasoning*.

Gregory, P., & Cresswell, S. (2016). Domain model acquisition in the presence of static relations in the LOP system. In *International Joint Conference on Artificial Intelligence, (IJCAI-16)*, pp. 4160–4164.

Gregory, P., & Lindsay, A. (2016). Domain model acquisition in domains with action costs. In *International Conference on Automated Planning and Scheduling*.

Gregory, P., Schumann, H. C., Björnsson, Y., & Schiffel, S. (2015). The *GRL* system: Learning Board Game Rules With Piece-Move Interactions. In *Computer games*, pp. 130–148. Springer.

Hayton, T., Porteous, J., Ferreira, J. F., & Lindsay, A. (2020). Narrative planning model acquisition from text summaries and descriptions.. In *AAAI Conference on Artificial Intelligence*, pp. 1709–1716.

Hoffmann, J., Fates, N., & Palacios, H. (2010). Brothers in arms? on ai planning and cellular automata.. In *European Conference on Artificial Intelligence*, pp. 223–228.

Hoffmann, J., Porteous, J., & Sebastia, L. (2004). Ordered landmarks in planning. *Journal of Artificial Intelligence Research*, *22*, 215–278.

Hu, Y., & De Giacomo, G. (2011). Generalized planning: Synthesizing plans that work for multiple environments. In *International Joint Conference on Artificial Intelligence*, Vol. 22.

Icarte, R. T., Waldie, E., Klassen, T., Valenzano, R., Castro, M., & McIlraith, S. (2019). Learning reward machines for partially observable reinforcement learning. In *Advances in Neural Information Processing Systems*, pp. 15497–15508.

Jiménez, S., De La Rosa, T., Fernández, S., Fernández, F., & Borrajo, D. (2012). A review of machine learning for automated planning. *The Knowledge Engineering Review*, *27*(04), 433–467.

Jiménez, S., Segovia-Aguas, J., & Jonsson, A. (2019). A review of generalized planning. *The Knowledge Engineering Review*, *34*(e5), 1–28.

Kaelbling, L. P., Littman, M. L., & Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of artificial intelligence research*, *4*, 237–285.

Kambhampati, S. (2007). Model-lite planning for the web age masses: The challenges of planning with incomplete and evolving domain models. In *National Conference on Artificial Intelligence, (AAAI-07)*.

Kearns, M., & Koller, D. (1999). Efficient reinforcement learning in factored mdps. In *International Joint Conference on Artificial Intelligence*, Vol. 16, pp. 740–747.

Khardon, R. (1999). Learning action strategies for planning domains. *Artificial Intelligence*, *113*(1-2), 125–148.

Kok, S., & Domingos, P. (2007). Statistical predicate invention. In *International Conference on Machine Learning*, pp. 433–440.

Kolobov, A. (2012). Planning with markov decision processes: An ai perspective. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, *6*(1), 1–210.

Konidaris, G., Kaelbling, L. P., & Lozano-Pérez, T. (2018). From skills to symbols: Learning symbolic representations for abstract high-level planning. *Journal of Artificial Intelligence Research*, *61*, 215–289.

Kucera, J., & Barták, R. (2018). LOUGA: learning planning operators using genetic algorithms. In *Pacific Rim Knowledge Acquisition Workshop, PKAW-18*, pp. 124–138.

Lanchas, J., Jiménez, S., Fernández, F., & Borrajo, D. (2007). Learning action durations from executions. In *ICAPS'07 Workshop on AI Planning and Learning*.

Lang, T., Toussaint, M., & Kersting, K. (2012). Exploration in relational domains for model-based reinforcement learning. *J. Mach. Learn. Res.*, *13*, 3725–3768.

Langley, P. (2000). The computational support of scientific discovery. *International Journal of Human-Computer Studies*, *53*(3), 393–410.

Lindsay, A., Read, J., Ferreira, J., Hayton, T., Porteous, J., & Gregory, P. (2017a). Framer: Planning models from natural language action descriptions. In *International Conference on Automated Planning and Scheduling*.

Lindsay, A., Read, J., Ferreira, J. F., Hayton, T., Porteous, J., & Gregory, P. (2017b). Framer: Planning models from natural language action descriptions. In Barbulescu, L., Frank, J., Mausam, & Smith, S. F. (Eds.), *International Conference on Automated Planning and Scheduling, ICAPS 2017*, pp. 434–442. AAAI Press.

Lipovetzky, N., & Geffner, H. (2012). Width and serialization of classical planning problems. In *European Conference on Artificial Intelligence*, pp. 540–545.

Lipovetzky, N., Ramirez, M., & Geffner, H. (2015). Classical planning with simulators: Results on the atari video games. In *International Joint Conference on Artificial Intelligence*, pp. 1610–1616.

Lotinac, D., Segovia-Aguas, J., Jiménez, S., & Jonsson, A. (2016). Automatic generation of high-level state features for generalized planning. In *International Joint Conference on Artificial Intelligence, (IJCAI-16)*, pp. 3199–3205.

Martin, M., & Geffner, H. (2004). Learning generalized policies from planning examples using concept languages. *Applied Intelligence*, *20*(1), 9–19.

Martínez, D. M., Alenyà, G., Ribeiro, T., Inoue, K., & Torras, C. (2017). Relational Reinforcement Learning for Planning with Exogenous Effects. *J. Mach. Learn. Res.*, *18*, 78:1–78:44.

Mausam, A. K. (2012). *Planning with markov decision processes: an ai perspective*. Morgan & Claypool Publishers.

Miller, T. (2019). Explanation in artificial intelligence: Insights from the social sciences. *Artificial Intelligence*, *267*, 1–38.

Mourão, K., Petrick, R. P., & Steedman, M. (2010). Learning action effects in partially observable domains.. In *European Conference on Artificial Intelligence*, pp. 973–974.

Mourão, K., Zettlemoyer, L. S., Petrick, R. P. A., & Steedman, M. (2012). Learning STRIPS operators from noisy and incomplete observations. In *Conference on Uncertainty in Artificial Intelligence, UAI-12*, pp. 614–623.

Muggleton, S. (1992). *Inductive logic programming*. Morgan Kaufmann.

Nir, R., Shleyfman, A., & Karpas, E. (2020). Automated synthesis of social laws in strips. In *AAAI Conference on Artificial Intelligence*, pp. 9941–9948.

Pasula, H. M., Zettlemoyer, L. S., & Kaelbling, L. P. (2007a). Learning symbolic models of stochastic domains. *Journal of Artificial Intelligence Research*, *29*, 309–352.

Pasula, H. M., Zettlemoyer, L. S., & Kaelbling, L. P. (2007b). Learning Symbolic Models of Stochastic Domains. *J. Artif. Intell. Res.*, *29*, 309–352.

Pearl, J. (2019). The seven tools of causal inference, with reflections on machine learning. *Communications of the ACM*, *62*(3), 54–60.

Pednault, E. P. (1994). Adl and the state-transition model of action. *Journal of logic and computation*, *4*(5), 467–512.

Ramírez, M. (2012). *Plan recognition as planning*. Ph.D. thesis, Universitat Pompeu Fabra.

Ramirez, M., Papasimeon, M., Benke, L., Lipovetzky, N., Miller, T., & Pearce, A. R. (2017). Real-time uav maneuvering via automated planning in simulations.. In *International Joint Conference on Artificial Intelligence*, pp. 5243–5245.

Sallans, B., & Hinton, G. E. (2004). Reinforcement learning with factored states and actions. *Journal of Machine Learning Research*, *5*(Aug), 1063–1088.

Sanner, S. (2010). Relational dynamic influence diagram language (rddl): Language description. *Unpublished ms. Australian National University*, *32*.

Sanner, S., & Boutilier, C. (2009). Practical solution techniques for first-order mdps. *Artif. Intell.*, *173*(5-6), 748–788.

Schwarting, W., Alonso-Mora, J., & Rus, D. (2018). Planning and Decision-Making for Autonomous Vehicles. *Annual Review of Control, Robotics, and Autonomous Systems*, *1*(1), 187–210.

Segovia-Aguas, J., Jiménez, S., & Jonsson, A. (2018). Computing hierarchical finite state controllers with classical planning. *Journal of Artificial Intelligence Research*, *62*, 755–797.

Segovia-Aguas, J., Jiménez, S., & Jonsson, A. (2019). Computing programs for generalized planning using a classical planner. *Artificial Intelligence*, *272*, 52–85.

Shahaf, D., & Amir, E. (2006). Learning partially observable action schemas. In *AAAI Conference on Artificial Intelligence*.

Shani, G., Brafman, R. I., & Shimony, S. E. (2005). Model-based online learning of pomdps. In *European Conference on Machine Learning*, pp. 353–364. Springer.

Shani, G., Pineau, J., & Kaplow, R. (2013). A survey of point-based pomdp solvers. *Autonomous Agents and Multi-Agent Systems*, *27*(1), 1–51.

Shirazi, A., & Amir, E. (2011). First-order logical filtering. *Artificial Intelligence*, *175*(1), 193–219.

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

Sreedharan, S., Chakraborti, T., & Kambhampati, S. (2017). Explanations as model reconciliation-a multi-agent perspective.. In *AAAI Fall Symposia*, pp. 277–283.

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

Strehl, A. L., Diuk, C., & Littman, M. L. (2007). Efficient structure learning in factored-state mdps. In *AAAI Conference on Artificial Intelligence*, Vol. 7, pp. 645–650.

Suárez-Hernández, A., Segovia-Aguas, J., Torras, C., & Alenyá, G. (2021). Online action recognition. In *AAAI Conference on Artificial Intelligence*.

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning - An Introduction*. Adaptive computation and machine learning. MIT Press.

Tadepalli, P., Givan, R., & Driessens, K. (2004). Relational reinforcement learning: An overview. In *Proceedings of the ICML-2004 workshop on relational reinforcement learning*, pp. 1–9.

Walsh, T., Goschin, S., & Littman, M. (2010). Integrating sample-based planning and model-based reinforcement learning. In *AAAI Conference on Artificial Intelligence*.

Walsh, T. J., & Littman, M. L. (2008a). Efficient learning of action schemas and web-service descriptions. In *National Conference on Artificial Intelligence, AAAI-08*, pp. 714–719.

Walsh, T. J., & Littman, M. L. (2008b). Efficient learning of action schemas and web-service descriptions. In *AAAI Conference on Artificial Intelligence*, Vol. 8, pp. 714–719.

Wang, C., & Khardon, R. (2010). Relational Partially Observable MDPs. In Fox, M., & Poole, D. (Eds.), *AAAI Conference on Artificial Intelligence*.

Winston, P. H. (1970). Learning structural descriptions from examples. Tech. rep., Massachusetts Institute of Technology, USA.

Wolfram, S. (2002). *A new kind of science*, Vol. 5. Wolfram media Champaign, IL.

Xiao, Y., Codevilla, F., Pal, C., & Lopez, A. M. (2020). Action-based Representation Learning for Autonomous Driving. In *Conference on Robot Learning, (CoRL 2020)*, Vol. 155 of *Proceedings of Machine Learning Research*, pp. 232–246.

Xu, J. Z., & Laird, J. E. (2011). Combining Learned Discrete and Continuous Action Models. In Burgard, W., & Roth, D. (Eds.), *AAAI Conference on Artificial Intelligence, AAAI 2011*. AAAI Press.

Yang, F., Khandelwal, P., Leonetti, M., & Stone, P. (2014). Planning in answer set programming while learning action costs for mobile robots. In *Knowledge Representation and Reasoning in Robotics*, pp. 71–78. AAAI.

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

Yoon, S., Fern, A., & Givan, R. (2008). Learning control knowledge for forward search planning.. *Journal of Machine Learning Research*, *9*(4).

Zhuo, H. H., & Kambhampati, S. (2013). Action-model acquisition from noisy plan traces. In *International Joint Conference on Artificial Intelligence, IJCAI-13*, pp. 2444–2450.

Zhuo, H. H., Nguyen, T., & Kambhampati, S. (2013). Refining incomplete planning domain models through plan traces. In *International joint conference on artificial intelligence*.

Zhuo, H. H., Yang, Q., Hu, D. H., & Li, L. (2010). Learning complex action models with quantifiers and logical implications. *Artificial Intelligence*, *174*(18), 1540–1569.