ECAI 2020
G.D. Giacomo et al. (Eds.)

doi:10.3233/FAIA200361

Learning First-Order Symbolic Representations for Planning from the Structure of the State Space

Blai Bonet[1] and **Hector Geffner**[2]

Abstract. One of the main obstacles for developing flexible AI systems is the split between data-based learners and model-based solvers. Solvers such as classical planners are very flexible and can deal with a variety of problem instances and goals but require first-order symbolic models. Data-based learners, on the other hand, are robust but do not produce such representations. In this work we address this split by showing how the *first-order symbolic representations* that are used by planners can be learned *from non-symbolic inputs* that encode the structure of the state space. The *representation learning problem* is formulated as the problem of inferring planning instances over a common but unknown first-order domain that account for the structure of the observed state space. This means to infer a complete first-order representation (i.e. general action schemas, relational symbols, and objects) that explains the observed state space structures. The inference problem is cast as a two-level combinatorial search where the outer level searches for values of a small set of hyperparameters and the inner level, solved via SAT, searches for a first-order symbolic model. The framework is shown to produce general and correct first-order representations for standard problems like Gripper, Blocksworld, and Hanoi from input graphs that encode the flat state-space structure of a single instance.

1 INTRODUCTION

Two of the main research threads in AI revolve around the development of *data-based learners* capable of inferring behavior and functions from experience and data, and *model-based solvers* capable of tackling well-defined but intractable models like SAT, classical planning, and Bayesian networks. Learners, and in particular deep learners, have achieved considerable success but result in black boxes that do not have the flexibility, transparency, and generality of their model-based counterparts [26, 27, 32, 12, 17]. Solvers, on the other hand, require models which are hard to build by hand. This work is aimed at bridging this gap by addressing the problem of learning first-order models from data without using any prior symbolic knowledge.

Almost all existing approaches for learning representations for acting and planning fall into two camps to be discussed below. On the one hand, methods that output symbolic representations but which require symbolic representations in the input; on the other, methods that do not require symbolic inputs but which do not produce them either. First-order representations structured in terms of objects and relations like PDDL [29, 21, 18], however, have a number of benefits; in particular, they are easier to understand, and they can be easily reused for defining a variety of new instances and goals. Representations like PDDL, however, are written by hand; the challenge is to learn them from data.

In the proposed formulation, general first-order planning representations are learned from graphs that encode the structure of the state space of one or more problem instances. For this, the *representation learning problem* is formulated as the problem of inferring planning instances P_i over a common, fully unknown, first-order domain D (action schemas and predicate symbols) such that the graphs $G(P_i)$ associated with the instances P_i and the observed graphs G_i are *structurally equivalent*. Since the space of possible domains can be bounded by a number of hyperparameters with small values, such as the number of action schemas, predicates, and arguments, the inference problem is cast as a two-level combinatorial search where the outer level looks for the right value of the hyperparameters and the inner level, formulated and solved via SAT, looks for a first-order representation that fits the hyperparameters and explains the input graphs. Correct and general first-order models for domains like Gripper, Blocksworld, and Hanoi are shown to be learned from graphs that encode the flat state-space structure of a single small instance.

2 RELATED RESEARCH

Object-oriented MDPs [13] and similar work in classical planning [37, 2, 1], build first-order model representations but starting with a first-order symbolic language, or with information about the actions and their arguments [11]. Inductive logic programming methods [31] have been used for learning general policies but from symbolic encodings too [23, 28, 14]. More recently, general policies have been learned using deep learning methods but also starting with PDDL models [36, 9, 22, 6]. The same holds for methods for learning abstract planning representations [7]. Other recent methods produce PDDL models from given macro-actions (options) but these models are propositional and hence do not generalize [24].

Deep reinforcement learning (DRL) methods [30], on the other hand, generate policies over high-dimensional perceptual spaces like images, without using any prior symbolic knowledge [19, 10, 15]. Yet by not constructing first-order representations, DRL methods lose the benefits of transparency, reusability, and compositionality [27, 25]. Recent work in deep symbolic relational reinforcement learning [16] attempts to account for objects and relations through the use of attention mechanisms and loss functions but the semantic and conceptual gap between the low-level techniques and the high-level representations that are required remains just too large. Something similar occurs with work aimed at learning low-dimensional representations that disentangle the factors of variations in the data [35]. The first-order representations used in planning are low dimen-

[1] Universidad Simón Bolívar, Venezuela. Email: bonet@usb.ve
[2] ICREA & Universitat Pompeu Fabra, Spain. Email: hector.geffner@upf.edu

sional but highly structured, and it is not clear that they can be learned in this way. An alternative approach produces first-order representations using a class of variational autoencoders that provide a low-dimensional encoding of the images representing the states [4, 3].

3 FORMULATION

The proposed formulation for learning planning representations from data departs from existing approaches in two fundamental ways. First, unlike deep learning approaches, *the representations are not learned from images associated with states but from the structure of the state space*. Second, unlike other methods that deliver first-order symbolic planning representations, the *proposed method does not assume knowledge of the action schemas, predicate symbols, or objects*; these are all learned from the input. *All the data required* to learn planning representations in the four domains considered in the experiments, Blocksworld, Towers of Hanoi, Gripper, and Grid, is shown in Fig. 1. In each case, the sole input is a labeled directed graph that encodes the structure of the state-space associated with a small problem instance, and the output is a PDDL-like representation made up of a general first-order domain with action schemas and predicate symbols, some of which are possibly static, and instance information describing objects and an initial situation. No goal information, however, is assumed in the input, and no goal information is produced in the output. Further details about the inputs and outputs of the representation learning approach are described below.

3.1 Inputs: Labeled Graphs

The inputs are one or more labeled directed graphs that encode the structure of the state space of one or several problem instances. The nodes of these graphs represent the states and no information about the contents or inner structure of them is provided or needed. Labels in the edges represent action types; e.g., action types in the Gripper domain distinguish three types of actions: moves, pick ups, and drops. In the absence of labels, all edges are assumed to have the same label. No other information is provided as input or in the input graphs; in particular, no node is marked as an initial or goal state (see Fig. 1). Initial states for each input graph are inferred indirectly, but nothing is inferred about goals.

In standard, tabular (model-free) reinforcement learning (RL) schemes [34], the inputs are traces made up of states, actions, and rewards, and the task is to learn a policy for maximizing (discounted) expected reward, not to learn factored symbolic representations of the actions and states. The same inputs are used in model-based reinforcement learning methods where the policy is derived from a flat model that is learned incrementally but which does not transfer to other problems [8].

One way to understand the labeled input graphs used by our method is as collections of RL traces organized as graphs but without information about rewards and with the actions replaced by less specific action types or labels. Indeed, for a given node in an input graph, there may be zero, one, or many outgoing edges labeled with the same action type. We assume however that input graphs are *complete* in the sense that if they do not have an edge (s, l, s') between two nodes s and s' with action label l, it is because there is no trace that contains such a transition. Since the graphs required for building the target representations are not large, a sufficient number of sampled traces can be used to produce such graphs. In the experiments, however, we built the input graphs by systematically expanding the whole state space of a number of small problem instances (in our case just one) from some initial state. Formally, the input graphs are tuples $G = \langle V, E, L\rangle$, where the nodes n in V correspond to the different states, and the edges (n, n') in E with label $l \in L$, denoted (n, l, n'), correspond to state transitions produced by an action with label l. For the sake of the presentation, it is assumed that all nodes in an input graph can be reached from one or more nodes. The formulation, however, does not require this assumption.

3.2 Outputs: First-Order Representations

Given labeled graphs $G_1, \ldots, G_m$ in the input, the learning method produces a corresponding set of planning instances $P_1, \ldots, P_m$ in the output over a common planning domain D (also learned). A (classical) planning instance is a pair $P = \langle D, I\rangle$ where D is a **first-order planning domain** and I is the **instance information**. The planning domain D contains a set of predicate symbols and a set of action schemas with preconditions and effects given by atoms $p(x_1, \ldots, x_k)$ or their negations, where p is a domain predicate and each x_i is a variable representing one of the arguments of the action schema. The instance information is a tuple $I = \langle O, Init, Goal\rangle$ where O is a (finite) set of object names c_i, and $Init$ and $Goal$ are sets of ground atoms $p(c_1, \ldots, c_k)$ or their negations, where p is a predicate symbol in D of arity k. This is the structure of planning problems expressed in PDDL [29, 21] that corresponds to STRIPS schemas with negation. The actual name of the constants in O is irrelevant and can be replaced by numbers in the interval $[1, N]$ where $N = |O|$ is the number of objects in O. Similarly, goals are included in I to keep the notation consistent with planning practice, but they play no role in the formulation.

A problem $P = \langle D, I\rangle$ defines a *labeled* graph $G(P) = \langle V, E, L\rangle$ where the nodes n in V correspond to the states $s(n)$ over P, and there is an edge (n, n') in E with label a, (n, a, n'), if the state transitions $(s(n), s(n'))$ is enabled by a ground instance of the schema a in P. It is thus assumed that the ground instances of the same action schema share the same label, and hence that edges with different labels in the input graphs involve ground instances from different action schemas.

3.3 Inputs to Outputs: Representation Discovery

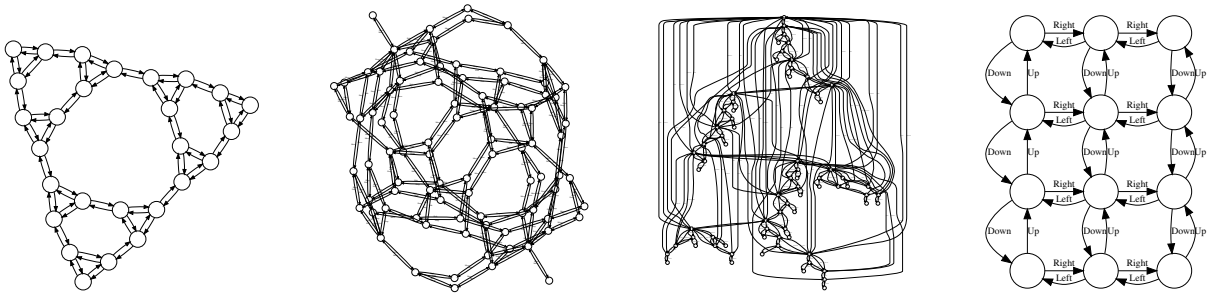
Representation learning in our setting is about finding the (simplest) domain D and instances P_i over D that define graphs $G(P_i)$ that are structurally equivalent (isomorphic) to the input graphs G_i. We formalize the relation between the labeled graphs $G(P_i)$ associated with the instances P_i and the input labeled graphs G_i as follows:

Definition 1. *An instance P **accounts for** a labeled graph G if (n, a, n') is a labeled edge in $G(P)$ iff $(h(n), g(a), h(n'))$ is a labeled edge in G, for some function g between the labels in $G(P)$ and those in G, and a 1-to-1 function h between the nodes in $G(P)$ and those in G.*

The representation learning problem is then:

Definition 2. *The **representation discovery problem** is finding a domain D and instances $P_i = \langle D, I_i\rangle$ that account for the input labeled graphs G_i, $i = 1, \ldots, m$.*

For solving the problem, we take advantage that the space of possible domain representations D is bounded by the values of a small number of domain parameters like the number of action schemas, predicates, and arguments (arities). Likewise, the number of possible



(a) Towers of Hanoi (1 label) (b) Gripper (3 labels) (c) Blocksworld (3 labels) (d) 4×3 Grid (4 labels)

Figure 1. Input data for learning the planning representations in the four domains considered. The proposed formulation accepts one or mode labeled directed graphs encoding the structure of the state space of one or more problem instances as the *sole input*. It then produces symbolic PDDL-like planning representations that account for the input graphs in the form of a general first-order domain with action schemas and predicate symbols, and instance information describing objects and an initial situation. Labels are used to distinguish action types. In each of the four domains, a single input graph corresponding to a single instance (as shown) sufficed to learn the general first-order domain representations. (The graphs can be zoomed in to reveal the labels.)

instances I_i is bounded by the size of the input graphs. As a result, representation discovery becomes a combinatorial problem. The domain parameters define also how complex a domain representation is, with simpler representations involving parameters with smaller values. *Simpler domain representations are preferred* although we do not introduce or deal with functions to rank domains. Instead, we simply bound the value of such parameters.

4 SAT ENCODING

The problem of computing the instances $P_i = \langle D, I_i \rangle$ that account for the observed labeled graphs G_i, $i = 1, \ldots, n$, is mapped into the problem of checking the satisfiability of a propositional theory $T_\alpha(G_{1:n})$ where α is a vector of hyperparameters. The theory $T_\alpha(G_{1:n})$ is the union of formulas or layers

$$T_\alpha(G_{1:n}) = T^0_\alpha \cup \bigcup \{T^i_\alpha : i = 1, 2, \ldots, n\} \qquad (1)$$

where the formula T^0_α is aimed at capturing the domain D, and takes as input the vector α of hyperparameters only, while formula T^i_α is aimed at capturing the instance information I_i, and takes as input the graph G_i as well. The *domain layer* T^0_α involves its own variables, while each *instance layer* T^i_α involves its own variables and those of the domain layer. The encoding of the domain D and the instances P_i can be read (decoded) from the truth assignments that satisfy the theory $T_\alpha(G_{1:n})$ over the variables in the corresponding layer.

The **vector of hyperparameters** α represents the number of action schemas and the arity of each one of them, the number of predicate symbols and the arity of each one of them, the number of different atoms in the schemas, the total number of unary and binary static predicates, and the number of objects in each layer i. We provide a max value on each of these parameters, and then consider the theories $T_\alpha(G_{1:n})$ for each of the α vectors that comply with such bounds. Action schemas a, predicate symbols p, atom names m, arguments ν, unary and binary predicates u and b, and objects o, are all integers that range from 1 to their corresponding number in α. A predicate p is static if all p-atoms are static, meaning that they do not appear in the effects of any action. Static atoms are used as preconditions to control the grounding of action schemas, and in the SAT encoding, they are treated differently than the other (fluent) atoms.

Next we fully define the theory $T_\alpha(G_{1:n})$: first the domain layer T^0_α and then each of the instance layers T^i_α, $i = 1, \ldots, n$. The encoding is not trivial and it is one of the main contributions of this work, along with the formulation of the representation learning problem and the results. For lack of space, we only provide brief explanations for the formulas in the encoding.

4.1 SAT Encoding: Domain Layer T^0_α

The domain layer T^0_α makes use of the following boolean variables, some of which can be regarded as decision variables, and the others, as the variables whose values are determined by them. It defines the space of possible domains D given the value of the hyperparameters α and it does not use the input graphs.

Decision propositions:

- $\text{p0}(a, m)/\text{p1}(a, m)$: m is negative/positive precondition of a,
- $\text{e0}(a, m)/\text{e1}(a, m)$: m is negative/positive effect of a,
- $\text{label}(a, l)$: label of action schema a is l,
- $\text{arity}(p, i)$: arity of predicate symbol p,
- $\text{at}(m, p)$: m is a p-atom,
- $\text{at}(m, i, \nu)$: i-th argument of m is (action) argument ν,
- $\text{un}(u, a, \nu)$: action a uses static unary predicate u on argument ν,
- $\text{bin}(b, a, \nu, \nu')$: a uses static binary pred. b on arguments ν and ν'.

Implied propositions:

- $\text{use}(a, m)$: action a uses atom m,
- $\text{use}(m)$: some action a uses atom m,
- $\text{arg}(a, \nu)$: action a uses argument ν,
- $\text{argval}(a, \nu, m, i) \Leftrightarrow \text{use}(a, m) \wedge \text{at}(m, i, \nu)$,
- $\neg\text{static0}(a, m, p) \Rightarrow \text{at}(m, p) \wedge \text{e0}(a, m)$,
- $\neg\text{static1}(a, m, p) \Rightarrow \text{at}(m, p) \wedge \text{e1}(a, m)$.

4.1.1 Formulas

Atoms in preconditions and effects, and unique action labels:

$$\text{use}(a, m) \Leftrightarrow \text{p0}(a, m) \vee \text{p1}(a, m) \vee \text{e0}(a, m) \vee \text{e1}(a, m) \qquad (2)$$

$$\text{use}(m) \Leftrightarrow \bigvee_a \text{use}(a, m) \qquad (3)$$

$$\neg\text{p0}(a, m) \vee \neg\text{p1}(a, m) \qquad (4)$$

$$\neg\text{e0}(a, m) \vee \neg\text{e1}(a, m) \qquad (5)$$

$$\text{At-Most-1 } \{\text{label}(a, l) : l\} \qquad (6)$$

Effects are non-redundant, and unique predicate arities:

$$\text{e0}(a, m) \Rightarrow \neg\text{p0}(a, m) \qquad (7)$$

$$\text{e1}(a, m) \Rightarrow \neg\text{p1}(a, m) \qquad (8)$$

$$\text{Exactly-1 } \{\text{arity}(p,i) : 0 \le i \le \text{max-arity}\} \quad (9)$$

Structure of atoms: predicate symbols and arguments:

$$\text{Exactly-1 } \{\text{at}(m,p) : p\} \quad (10)$$

$$\text{At-Most-1 } \{\text{at}(m,i,\nu) : \nu\} \quad (11)$$

$$\text{at}(m,p) \wedge \text{at}(m,i,\nu) \Rightarrow \bigvee_{i \le j \le \text{max-arity}} \text{arity}(p,j) \quad (12)$$

$$\text{at}(m,p) \wedge \text{arity}(p,i) \Rightarrow \bigwedge_{1 \le j \le i} \bigvee_{\nu} \text{at}(m,j,\nu) \quad (13)$$

$$\text{at}(m,p) \wedge \text{arity}(p,i) \Rightarrow \bigwedge_{i < j \le \text{max-arity}} \neg\text{at}(m,j,\nu) \quad (14)$$

1-1 map of atom names into possible atom structures: For $\text{vect}(m)$ denoting the boolean vector with components $\text{use}(m)$, $\{\text{at}(m,p)\}_p$, and $\{\text{at}(m,i,\nu)\}_{i,\nu}$, impose the constraint $\text{vect}(m) <_{lex} \text{vect}(m')$ when $m < m'$ for uniqueness.

$$\text{Strict-Lex-Order } \{\text{vect}(m) : m\} \quad (15)$$

Atoms are non-static; static atoms dealt with separately:

$$\bigvee_{a,m} \left[\neg\text{static0}(a,m,p) \vee \neg\text{static1}(a,m,p)\right] \quad (16)$$

$$\neg\text{static0}(a,m,p) \Rightarrow \text{at}(m,p) \wedge \text{p1}(a,m) \wedge \text{e0}(a,m) \quad (17)$$

$$\neg\text{static1}(a,m,p) \Rightarrow \text{at}(m,p) \wedge \text{p0}(a,m) \wedge \text{e1}(a,m) \quad (18)$$

Atom and action arguments:

$$\text{use}(a,m) \wedge \text{at}(m,i,\nu) \Rightarrow \text{arg}(a,\nu) \quad (19)$$

$$\text{arg}(a,\nu) \Rightarrow \bigvee_{m,i} \text{argval}(a,\nu,m,i) \quad (20)$$

$$\text{argval}(a,\nu,m,i) \Leftrightarrow \text{use}(a,m) \wedge \text{at}(m,i,\nu) \quad (21)$$

Arities of action schemas and predicate symbols:

$$\bigwedge_{\nu \ge \text{arity(action } a)} \neg\text{arg}(a,\nu) \quad (22)$$

$$\bigwedge_{0 \le \nu < \text{arity(action } a)} \text{arg}(a,\nu) \quad (23)$$

$$\bigwedge_{i \ne \text{arity(atom } p)} \neg\text{arity}(p,i) \wedge \bigwedge_{i = \text{arity(atom } p)} \text{arity}(p,i) \quad (24)$$

If static predicate on action argument, argument must exist:

$$\text{un}(u,a,\nu) \Rightarrow \text{arg}(a,\nu) \quad (25)$$

$$\text{bin}(b,a,\nu,\nu') \Rightarrow \text{arg}(a,\nu) \wedge \text{arg}(a,\nu') \quad (26)$$

4.2 SAT Encoding: Instance Layer T^i_α

The layers T^i_α of the propositional theory $T_\alpha(G_{1:n})$ make use of the input graphs G_i in the form of a set of states (nodes) s and transitions (edges) t. The source and destination states of a transition t are denoted $t.src$ and $t.dest$, and the label as $t.label$. The layers T^i_α introduce symbols for ground atoms k, objects o, and tuples of objects $\bar{o}$ whose size matches the arity of the context where they are used (action and predicate arguments). The number of ground atoms k is determined by the number of objects, predicate symbols, and arguments, as established by the hyperparameters in α. The index i that refers to the i-th input graph G_i is omitted for readability.

Decision propositions:

- $\text{mp}(t,a)$: transition t is mapped to action schema a,
- $\text{mf}(t,k,m)$: ground atom k is mapped to atom m in transition t,
- $\phi(k,s)$: value of (boolean) ground atom k at state s,
- $\text{gr}(k,p)$: ground atom k refers to predicate symbol p,
- $\text{gr}(k,i,o)$: i-th argument of ground atom k is object o [$i > 0$],
- $r(u,o)$: true if $u(o)$ holds for static unary predicate u,
- $s(b,o,o')$: true if $b(o,o')$ holds for static binary predicate b,
- $\text{gtuple}(a,\bar{o})$: true if $a(\bar{o})$ is a ground instance of a.

Implied propositions:

- $\text{free}(k,t,a)$: ground atom k is unaffected in trans. t mapped to a,
- $\text{g}(k,s,s') \Leftrightarrow \phi(k,s) \oplus \phi(k,s')$ ($\oplus$ is XOR),
- $U(u,a,\nu,o) \Leftrightarrow \text{un}(u,a,\nu) \wedge \neg r(u,o)$,
- $B(b,a,\nu,\nu',o,o') \Leftrightarrow \text{bin}(b,a,\nu,\nu') \wedge \neg s(b,o,o')$,
- $\text{mt}(t,\nu,o)$: argument ν is mapped to object o in transition t,
- $W(t,k,i,\nu) \Rightarrow \left[\text{gr}(k,i,o) \Leftrightarrow \text{mt}(t,\nu,o)\right]$,
- $G(t,a,\bar{o})$: transition t is (ground) instance of $a(\bar{o})$,
- $\text{appl}(a,\bar{o},s)$: ground instance $a(\bar{o})$ is applicable in state s,
- $\text{vio0}(a,\bar{o},s,k)$: k is neg. precond. $p(\bar{o})$ of a that is false in s,
- $\text{vio1}(a,\bar{o},s,k)$: k is pos. precond. $p(\bar{o})$ of a that is false in s,
- $\text{pre0eq}(a,\bar{o},k,m) \Rightarrow \text{p0}(a,m) \wedge \text{eq}(\bar{o},m,k)$,
- $\text{pre1eq}(a,\bar{o},k,m) \Rightarrow \text{p1}(a,m) \wedge \text{eq}(\bar{o},m,k)$,
- $\text{eq}(\bar{o},m,k)$: ground atom k instantiates atom m with tuple $\bar{o}$.

4.2.1 Formulas

Binding transitions with action schemas, and ground atoms with atom schemas:

$$\text{Exactly-1 } \{\text{mp}(t,a) : a\} \quad (27)$$

$$\text{At-Most-1 } \{\text{mf}(t,k,m) : m\} \quad (28)$$

$$\text{At-Most-1 } \{\text{mf}(t,k,m) : k\} \quad (29)$$

Consistency between mappings, labeling, and usage:

$$\text{mp}(t,a) \Rightarrow \text{label}(a, t.label) \quad (30)$$

$$\text{mp}(t,a) \wedge \text{mf}(t,k,m) \Rightarrow \text{use}(a,m) \quad (31)$$

$$\text{mp}(t,a) \wedge \text{use}(a,m) \Rightarrow \bigvee_k \text{mf}(t,k,m) \quad (32)$$

Ground atom unaffected if:

$$\text{mp}(t,a) \wedge \left[\bigwedge_m \neg\text{mf}(t,k,m)\right] \Rightarrow \text{free}(k,t,a) \quad (33)$$

$$\text{mp}(t,a) \wedge \text{mf}(t,k,m) \Rightarrow \left[\neg\text{e0}(a,m) \wedge \neg\text{e1}(a,m) \Leftrightarrow \text{free}(k,t,a)\right] \quad (34)$$

Transitions and inertia:

$$\text{mp}(t,a) \wedge \text{mf}(t,k,m) \wedge \text{p0}(a,m) \Rightarrow \neg\phi(k,t.src) \quad (35)$$

$$\text{mp}(t,a) \wedge \text{mf}(t,k,m) \wedge \text{p1}(a,m) \Rightarrow \phi(k,t.src) \quad (36)$$

$$\text{mp}(t,a) \wedge \text{mf}(t,k,m) \wedge \text{e0}(a,m) \Rightarrow \neg\phi(k,t.dst) \quad (37)$$

$$\text{mp}(t,a) \wedge \text{mf}(t,k,m) \wedge \text{e1}(a,m) \Rightarrow \phi(k,t.dst) \quad (38)$$

$$\text{mp}(t,a) \Rightarrow \left[\text{free}(k,t,a) \Leftrightarrow [\phi(k,t.src) \Leftrightarrow \phi(k,t.dst)]\right] \quad (39)$$

States must differ in value of some ground atom:

$$\text{g}(k,s,s') \Leftrightarrow \phi(k,s) \oplus \phi(k,s') \quad (40)$$

$$\bigwedge_{s<s'} \bigvee_k \text{g}(k,s,s') \quad (41)$$

Predicate symbol and arguments of ground atoms:

$$\text{Exactly-1 } \{\text{gr}(k,p) : p\} \quad (42)$$

$$\text{At-Most-1 } \{\text{gr}(k,i,o) : o\} \quad (43)$$

$$\text{gr}(k,p) \wedge \text{gr}(k,i,o) \Rightarrow \bigvee_{i \le j \le \text{max-arity}} \text{arity}(p,j) \quad (44)$$

$$\text{gr}(k,p) \wedge \text{arity}(p,i) \Rightarrow \bigwedge_{1 \le j \le i} \bigvee_o \text{gr}(k,j,o) \quad (45)$$

$$\text{gr}(k,p) \wedge \text{arity}(p,i) \Rightarrow \bigwedge_{i<j} \neg\text{gr}(k,j,o) \quad (46)$$

1-1 map of ground atoms names to predicates and arguments: For $\text{vect}(k)$ denoting boolean vector with components $\{\text{gr}(k,p)\}_p$ and $\{\text{gr}(k,i,o)\}_{i,o}$, impose constraint $\text{vect}(k) <_{lex} \text{vect}(k')$ for $k < k'$.

$$\text{Strict-Lex-Order } \{\text{vect}(k) : k\} \quad (47)$$

Ground atoms and schema atoms in sync:

$$\text{mf}(t,k,m) \Rightarrow \left[\text{at}(m,p) \Leftrightarrow \text{gr}(k,p)\right] \quad (48)$$

$$\text{mf}(t,k,m) \wedge \text{at}(m,i,\nu) \Rightarrow \bigvee_o \text{gr}(k,i,o) \quad (49)$$

$$\text{mf}(t,k,m) \wedge \text{gr}(k,i,o) \Rightarrow \bigvee_\nu \text{at}(m,i,\nu) \quad (50)$$

Excluded bindings of static predicates:

$$U(u,a,\nu,o) \Leftrightarrow \text{un}(u,a,\nu) \wedge \neg r(u,o) \quad (51)$$

$$B(b,a,\nu,\nu',o,o') \Leftrightarrow \text{bin}(b,a,\nu,\nu') \wedge \neg s(b,o,o') \quad (52)$$

Bindings associated with transitions (part 1):

$$\text{At-Most-1}\ \{\text{mt}(t,\nu,o) : o\} \quad (53)$$

$$\text{mp}(t,a) \wedge \text{arg}(a,\nu) \Rightarrow \bigvee_o \text{mt}(t,\nu,o) \quad (54)$$

$$\text{mp}(t,a) \wedge \text{mt}(t,\nu,o) \Rightarrow \text{arg}(a,\nu) \quad (55)$$

Bindings associated with transitions (part 2):

$$\text{mf}(t,k,m) \wedge \text{at}(m,i,\nu) \Rightarrow W(t,k,i,\nu) \quad (56)$$

$$W(t,k,i,\nu) \Rightarrow \big[\text{gr}(k,i,o) \Leftrightarrow \text{mt}(t,\nu,o)\big] \quad (57)$$

Explanation of non-existing ground actions $\text{gtuple}(a,\bar{o})$:

$$\neg\text{gtuple}(a,\bar{o}) \Rightarrow \bigvee_{o_i>0} \neg\text{arg}(a,\nu_i) \vee \bigvee_{u,i} U(u,a,\nu_i,o_i) \vee \bigvee_{b,i<j} B(b,a,\nu_i,\nu_j,o_i,o_j) \quad (58)$$

Explanation of existing ground actions:

$$\text{mp}(t,a) \wedge \text{mt}(t,\nu,o) \wedge \text{un}(u,a,\nu) \Rightarrow r(u,o) \quad (59)$$

$$\text{mp}(t,a) \wedge \text{mt}(t,\nu,o) \wedge \text{mt}(t,\nu',o') \wedge \text{bin}(b,a,\nu,\nu') \Rightarrow s(b,o,o') \quad (60)(61)$$

Ground actions must be used in some transition:

$$\text{gtuple}(a,\bar{o}) \Rightarrow \bigvee_t G(t,a,\bar{o}) \quad (62)$$

$$G(t,a,\bar{o}) \Rightarrow \text{gtuple}(a,\bar{o}) \quad (63)$$

$$G(t,a,\bar{o}) \Rightarrow \text{mp}(t,a) \wedge \bigwedge_{o_i>0} \text{mt}(t,\nu_i,o_i) \wedge \bigwedge_{o_i=0} \big[\text{arg}(a,\nu_i) \Rightarrow \text{mt}(t,\nu_i,o_i)\big] \quad (64)$$

$$\text{At-Most-1}\ \{G(t,a,\bar{o}) : t.src = s\} \quad (65)$$

$$\text{Exactly-1}\ \{G(t,a,\bar{o}) : a,\bar{o}\} \quad (66)$$

Applicable actions must be applied:

$$G(t,a,\bar{o}) \Rightarrow \text{appl}(a,\bar{o},t.src) \quad (67)$$

$$\text{appl}(a,\bar{o},s) \Rightarrow \bigvee_{t.src=s} G(t,a,\bar{o}) \quad (68)$$

$$\neg\text{appl}(a,\bar{o},s) \Rightarrow \neg\text{gtuple}(a,\bar{o}) \vee \bigvee_k \big[\text{vio0}(a,\bar{o},s,k) \vee \text{vio1}(a,\bar{o},s,k)\big] \quad (69)$$

$$\text{vio0}(a,\bar{o},s,k) \Rightarrow \phi(k,s) \wedge \bigvee_m \text{pre0eq}(a,\bar{o},k,m) \quad (70)$$

$$\text{vio1}(a,\bar{o},s,k) \Rightarrow \neg\phi(k,s) \wedge \bigvee_m \text{pre1eq}(a,\bar{o},k,m) \quad (71)$$

$$\text{pre0eq}(a,\bar{o},k,m) \Rightarrow \text{p0}(a,m) \wedge \text{eq}(\bar{o},m,k) \quad (72)$$

$$\text{pre1eq}(a,\bar{o},k,m) \Rightarrow \text{p1}(a,m) \wedge \text{eq}(\bar{o},m,k) \quad (73)$$

$$\text{eq}(\bar{o},m,k) \Rightarrow \big[\text{at}(m,p) \Leftrightarrow \text{gr}(k,p)\big] \quad (74)$$

$$\text{eq}(\bar{o},m,k) \wedge \text{at}(m,i,\nu_j) \Rightarrow \text{gr}(k,i,o_j) \quad (75)$$

The encoding also contains formulas that reduce the number of redundant, symmetric valuations, which are omitted here for clarity. Such formulas only affect the performance of SAT solvers and do not affect the satisfiability of the theory $T_\alpha(G_{1:n})$.

5 PROPERTIES

The correctness and completeness of the encoding is expressed as:

Theorem 3. *The instances $P_i = \langle D, I_i\rangle$ with parametrization α account for the input labeled graphs $G_1,\ldots,G_n$ applying every ground action at least once iff there is a satisfying assignment of the theory $T_\alpha(G_{1:n})$ that encodes these instances up to renaming.*

This means basically that if the graphs can be generated by some instances, such instances are encoded in one of the models of the SAT encoding. On the other hand, any satisfying assignment of the theory encodes a first-order domain D and instances P_i over D that solve the representation discovery problem for the input graphs.

The parametrization α associated with a set of instances with a shared domain is simply the value of the hyperparameters determined by the instances. The condition that ground actions must be applied at least once follows from (62) and could be relaxed. In the encoding, indeed, if a ground action $a(\bar{o})$ is never applied (i.e., $\text{gtuple}(a,\bar{o})$ is false), it must be because the static predicates filter it out (cf. second and third disjunctions in (58)). On the other hand, the first disjunct in (58) explains inexistent ground actions due to "wrong groundings"; namely, groundings of variables that are not arguments of the action schema.

The extraction of instances $P_i = \langle D, I_i\rangle$, $I_i = \langle O_i, Init_i, Goal_i\rangle$, from a satisfying assignment is direct for the domain D and the objects O_i in each instance I_i. The assignment embeds each node n of the input graph G_i into a first-order state $s(n)$ over the problem P_i. The initial state $Init_i$ can be set to any state $s(n)$ for a node n in G_i that is connected to all other nodes in G_i, while $Goal_i$ plays no role in the structure of the state space and it is left unconstrained.

Finally, observe that the size of the theory is exponential only in the hyperparameter that specifies the max arity of action schemas since the tuples $\bar{o}$ of objects that define grounded actions $a(\bar{o})$ appear explicit in the formulas. However, the arities of action schemas are bounded and small; we use a bound of 3 in all the experiments.

6 VERIFICATION

It is possible to verify the representations learned by leaving apart some input graphs G_k, $k > n$, for **testing only** as it is standard in supervised learning. For this, the learned domain D is verified with respect to each testing graph G_k individually, by checking whether there is an instance $P_k = \langle D, I_k\rangle$ of the learned domain D that accounts for the graph G_k, following Def. 1. This test may be also performed with a SAT solver over a propositional theory $T'(G_k)$ that is a simplified version of the theory $T_\alpha(G_{1:n})$. Indeed, if the domain D was obtained from a satisfying truth assignment σ for the theory $T_\alpha(G_{1:n}) = T^0_\alpha \cup \cup_{i=1,n} T^i_\alpha$, then $T'(G_k) = T^0_\alpha \cup T^k_\alpha \cup \sigma^0$ where σ^0 is the set of literals that captures the valuation σ over the symbols in T^0_α. In words, $T'(G_k)$ treats G_k as an input graph but with the values of the domain literals in layer T^0_α set to the values in σ^0.

7 EXPERIMENTS AND RESULTS

We performed experiments to test the computational feasibility of the approach and the type of first-order representations that are obtained. We considered four domains, Blocksworld, Towers of Hanoi, Grid, and Gripper. For each domain, we selected a single input graph $G = G_1$ of a small instance to build the theory $T_\alpha(G_{1:n})$ with $n = 1$, abbreviated $T_\alpha(G)$, converted it to CNF, and fed it to the SAT solver `glucose-4.1` [5]. The input graphs used in the experiments are shown in Fig. 1. The experiments were performed on Amazon EC2's `c5n.18xlarge` with a limit of 1 hour and 16Gb of memory. If $T_\alpha(G)$, for parameters α was found to be satisfiable, we obtained an instance $P = \langle D, I\rangle$. The size of these graphs in terms of the number of nodes and edges appear in Table 1 as #states and #trans, while #tasks is the number of possible parametrizations α that results from the following bounds:

– max number of action schemas set to number of labels,

Table 1. Instance, # of labels, nodes and edges in graph, # of parametrizations α and theories $T_\alpha(G)$, fraction evaluated, and # found to be indeterminate (SAT solver still running after 1h cutoff), UNSAT, or SAT, with $x+y+z$ meaning that x did not complete verification in time/memory bound, y failed it, and z passed it (solutions). Last columns show avg. sizes and times of theories that produced these solutions.

	Input TS			Statistics SAT Calls					Theory for SAT Tasks (avg.)			
Instance	#labels	#states	#trans.	#tasks	sample	INDET	UNSAT	SAT	#vars	#clauses	time	mem. (Mb)
Blocksworld (4blocks)	3	73	240	19,050	1,905	246	1,642	10+0+7	1,666,705.5	6,033,529.0	1,441.1	860.7
Towers of Hanoi (3disks + 3pegs)	1	27	78	6,390	639	24	614	0+0+1	860,704.0	3,328,492.0	1,691.7	454.5
Gripper (2rooms + 3balls)	3	88	280	19,050	1,905	333	1,564	0+2+6	1,592,358.5	6,176,073.3	1,840.3	873.4
Rectangular grid 4×3	4	12	34	37,800	3,780	55	3,496	10+141+78	321,904.0	1,165,860.6	156.7	164.5
Rectangular grid 4×3	2	12	34	15,120	1,512	36	1,408	2+4+62	343,472.7	1,299,706.1	46.3	175.4
Rectangular grid 4×3	1	12	34	7,560	756	11	715	2+0+28	363,418.2	1,683,392.7	53.4	211.0

– max number of predicate symbols set to 5,
– max arity of action schemas and predicates set to 3 and 2 resp,
– max number of atoms schemas set to 6,[3]
– max number of static predicates set to 5,
– max number of objects in an instance set to 7.

The choice for these bounds is arbitrary, yet for most benchmarks the first five domain parameters do not go much higher, and the last one is compatible with the idea of learning from small examples.

The hyperparameter vector α specifies the *exact* values of the parameters, compatible with the bounds, and the *exact* arities of each action schema and predicate. This is why there are so many parametrizations α and theories $T_\alpha(G)$ to consider (column #tasks). Given our computational resources, for each input, we run the SAT solver on 10% of them randomly chosen. The number of theories that are SAT, UNSAT, or INDET (SAT solver still running after time/memory limit) are shown in the table that also displays the number of solutions verified on the test instances. The last columns show the average sizes of the SAT theories $T_\alpha(G)$ that were solved and verified. For each domain, we chose one solution at random and display it, with the names of predicates and action schemas changed to reflect their meanings (i.e., our interpretation). These solutions are compatible with the hyperparameters but are not necessarily "simplest", as we have not attempted to rank the solutions found.

7.1 Towers of Hanoi

The input graph G is the transition system for Hanoi with 3 disks, 3 pegs, and one action label shown in Fig. 1(a). Only one sampled parametrization α yields a satisfiable theory $T_\alpha(G)$, and the resulting domain passes validation on two test instances, one with 4 disks and 3 pegs; the other with 3 disks and 4 pegs. This solution was found in 1,692 seconds and uses two predicates, `clear(d)` and `Non(x,y)`, to indicate that disk d is clear and that disk x is *not* on disk y respectively. Two binary static predicates are learned as well, `BIGGER` and `NEQ`. The encoding is correct and intuitive although it features negated predicates like `Non` and redundant preconditions like `Non(fr,d)` and `Non(d,fr)`. Still, it is remarkable that this subtle first-order encoding is obtained from the graph of one instance, and that it works for any instance involving any number of pegs and disks.

Hanoi (ref. 530)

```
Move(fr,to,d):
 Static: BIGGER(fr,d),BIGGER(to,d) NEQ(fr,to)
 Pre: -clear(fr),clear(to),clear(d),Non(fr,d),-Non(d,fr),Non(d,to)
 Eff: clear(fr),-clear(to),Non(d,fr),-Non(d,to)
```

[3] The atom schemas are of the form $p(t)$ where p is a predicate symbol and t is a tuple of numbers of the arity of p, with the numbers representing action schema arguments.

7.2 Gripper

The instance used to generate the graph G in Fig. 1(b) involves 2 rooms, 3 named balls, 2 grippers, and 3 action labels for moves, picks, and drops. In this case, 8 encodings are found, 6 of which pass verification over instances with 2 and 4 balls. One of these encodings, randomly chosen from these 6 is shown below. It was found in 863 seconds, and uses the atoms `at(room)`, `hold(gripper,ball)`, `Nfree(gripper)`, and `Nat(room,ball)` to denote the robot position, that `gripper` holds `ball`, that `gripper` holds some ball, and that `ball` is not in `room` respectively. The learned static predicates are both binary, `CONN` and `PAIR`: the first for different rooms, and the second, for a pair formed by a room and a gripper. There also redundant preconditions, but the encoding is correct for any number of rooms, grippers, and balls.

Gripper (ref. 13918)

```
Move(from,to):
 Static: CONN(from,to)
 Pre: at(from),-at(to)
 Eff: -at(from),at(to)

Drop(ball,room,gripper):
 Static: PAIR(room,gripper)
 Pre: at(room),Nfree(gripper),hold(gripper,ball),Nat(room,ball)
 Eff: -Nfree(gripper),-hold(gripper,ball),-Nat(room,ball)

Pick(ball,room,gripper):
 Static: PAIR(room,gripper)
 Pre: at(room),-Nfree(gripper),-hold(gripper,ball),-Nat(room,ball)
 Eff: Nfree(gripper),hold(gripper,ball),Nat(room,ball)
```

7.3 Blocksworld

The instance used to generate the graph in Fig. 1(c) has 4 blocks and 3 action labels to indicate moves to and from the table, and moves among blocks. 17 of the 10% of sampled tasks were SAT, and 7 of them complied with test instances with 2, 3 and 5 blocks. One of these encodings, selected randomly and found in 110 seconds, is shown below. It has the predicates `Nclear(x)` that holds when (block) x is not clear, and `Ntable-OR-Non(x,y)` that holds when x is not on the table for $x=y$, and when block x is on block y for $x\neq y$. The standard human-written encoding for Blocksworld features three predicates instead (`clear`, `ontable`, and `on`). This encoding uses one less predicate but it is more complex due to the disjunction in `Ntable-OR-Non(x,y)`. As before, some of the preconditions in the schemas are redundant, and for the action schema `MoveFromTable` the argument d is redundant.

Blocksworld (ref. 1688)

```
MoveToTable(x,y):
 Static: NEQ(x,y)
 Pre: -Nclear(x),Nclear(y),-Ntable-OR-Non(x,y),Ntable-OR-Non(x,x)
 Eff: -Nclear(y),-Ntable-OR-Non(x,x),Ntable-OR-Non(x,y)

MoveFromTable(x,y,d):
 Static: NEQ(x,y),EQ(y,d)
```

```
 Pre: -Nclear(x),-Nclear(d),-Ntable-OR-Non(x,x),Ntable-OR-Non(x,y)
 Eff: Nclear(d),Ntable-OR-Non(x,x),-Ntable-OR-Non(x,y)

Move(x,z,y):
 Static: NEQ(x,z),NEQ(z,y),NEQ(x,y)
 Pre: -Nclear(x),Nclear(y),-Nclear(z),Ntable-OR-Non(x,x),
      Ntable-OR-Non(x,z),-Ntable-OR-Non(x,y)
 Eff: Nclear(z),-Nclear(y),Ntable-OR-Non(x,y),-Ntable-OR-Non(x,z)
```

7.4 Grid

The graph G in Fig. 1(d) is for an agent that moves in a 4×3 rectangular grid using three classes of labels: (the default shown) 4 labels `Up`, `Right`, `Down`, and `Left`, 2 labels `Horiz` and `Vert`, and a unique label `Move`. Many solutions exist in this problem because the domain is very simple, even though the space of hyperparameters is the same. The randomly chosen solution for the input with four labels is complex and it is not shown. Instead, a simpler and more intuitive hand-picked solution (found in 3 seconds) is displayed, where the x position is encoded as usual (one object per position), but the y position is encoded with a unary counter (count is number of bits that are on).

```
Grid with 4 labels (ref. 4853)
Up(y,ny):
 Static: U0(y),B0(y,ny)
 Pre: -NatY(y)
 Eff: NatY(y),-NatY(ny)

Right(x,nx):
 Static: U1(x),U1(nx),B0(x,nx)
 Pre: unaryEncodingX(x),-unaryEncodingX(nx)
 Eff: unaryEncodingX(nx)

Down(py,y):
 Static: U0(py),B0(py,y)
 Pre: NatY(py),-NatY(y)
 Eff: -NatY(py),NatY(y)

Left(n,nx):
 Static: U0(nx),U1(n),B0(n,nx)
 Pre: unaryEncodingX(n),-unaryEncodingX(nx)
 Eff: -unaryEncodingX(n)
```

To illustrate the flexibility of the approach, we also show below a first-order representation that is learned from the input graph G that only has 2 labels; i.e., the labels `Right` and `Left` are replaced by the label `Horiz`, and the labels `Up` and `Down` by the label `Vert`.

```
Grid with 2 labels (ref. 1713)
Horiz(from,to):
 Static: B2(from,to)
 Pre: atX(from),-atX(to)
 Eff: -atX(from),atX(to)

Vert(to,from):
 Static: B0(to,from)
 Pre: -atY(to),atY(from)
 Eff: atY(to),-atY(from)
```

The inferred static predicates `B2` and `B0` capture the horizontal and vertical adjacency relations respectively.

8 DISCUSSION

We have shown how to learn first-order symbolic representations for planning from graphs that only encode the structure of the state space while providing no information about the structure of states or actions. While the formulation of the representation learning problem and its solution are very different from those used in deep (reinforcement) learning approaches, there are some commonalities: we are fitting a parametric representation in the form of theories $T_\alpha(G_{1:n})$ to data in the form of labeled graphs $G_1, \ldots, G_n$. The parameters come in two forms: as the vector of hyperparameters α that bounds the set of possible first-order planning domains D and the number of objects in each of the instances $P_i = \langle D, I_i \rangle$, and the boolean variables in the theory $T_\alpha(G_{1:n})$ that bound the possible domains D and instances P_i. The formulation makes room and exploits a strong structural prior or bias; namely, that the set of possible domains can be bounded by a small number of hyperparameters with small values (number of action schemas and predicates, arities, etc). Lessons learned, possible extensions, limitations, and challenges are briefly discussed next.

Where (meaningful) symbols come from? We provide a crisp technical answer to this question in the setting of planning where meaningful first-order symbolic representations are obtained from non-symbolic inputs in the form of plain state graphs. In the process, objects and relations that are not given as part of the inputs are learned. The choice of a first-order target language (lifted STRIPS with negation) was crucial. At the beginning of this work, we tried to learn the (propositional) state variables of a single instance from the same inputs, but failed to obtain the intended variables. Indeed, looking for propositional representations that minimize the number of variables or the number of variables that change, result in $O(\log |S|)$ variables (where $|S|$ is the number of states) and so-called Gray codes, that are not meaningful. The reuse of actions and relations as captured by first-order representations did the trick.

Traces vs. complete graphs. The inputs in our formulation are not observed traces but complete graphs. This distinction, however, is not critical when the graphs required for learning are small. Using Pearl's terminology [33], the input graphs can be regarded as defining the complete space of possible causal interventions that allow us to recover the causal structure of the domain in a first-order language. The formulation and the SAT encoding, however, can be adjusted in a simple manner to account for incomplete graphs where only certain nodes are marked or assumed to contain all of their children.

Non-determinism. For learning representations of non-deterministic actions, the inputs must be changed from OR graphs to AND-OR graphs. Then the action schemas that account for transitions linked by AND nodes must be forced to take the same arguments and the same preconditions. In contrast to other approaches for learning stochastic action models [38], this method does not require symbolic inputs but the structure of the space in the form of an AND-OR graph.

Noise. The learning approach produces crisp representations from crisp, noise-free inputs. However, limited amount of "noise" in the form of wrong transitions or labels can be handled at a computational cost by casting the learning task as an optimization problem, solvable with Weighted Max-SAT solvers instead of SAT solvers.

Representation learning vs. grounding. The proposed method learns first-order representations from the structure of the state space, not from the structure of states as displayed for example in images [4, 3]. The latter approaches are less likely to generate crisp representations due to the dependence on images, but at the same time, they deal with two problems at the same time: representation learning and representation (symbol) grounding [20]. Our approach deals with the former problem only; the second problem is for future work.

Learning or synthesis? The SAT formulation is used to learn a representation from one or more input graphs corresponding to one or more domain instances. The resulting first-order domain representation is correct for these instances but not necessarily for other instances. The more compact the domain representation the more likely

that it generalizes to other instances, yet studying the conditions under which this generalization would be correct with high probability is beyond the scope of this work.

9 CONCLUSIONS

We have shown that it is possible to learn first-order symbolic representations for planning from non-symbolic data in the form of graphs that only capture the structure of the state space. Our learning approach is grounded in the simple, crisp, and powerful principle of finding a simplest model that is able to explain the structure of the input graphs. The empirical results show that a number of subtle first-order encodings with static and dynamic predicates can be obtained in this way. We are not aware of other approaches that can derive first-order symbolic representations of this type without some information about the action schemas, relations, or objects. There are many performance improvements to be pursued in particular regarding to the SAT encoding and the scalability of the approach, the search in the (bounded) hyperparameter space, and the ranking and selection of the simplest solutions. Extensions for dealing with partial observations will be pursued as well.

ACKNOWLEDGEMENTS

The work was done while B. Bonet was at the Universidad Carlos III, Madrid, on a sabbatical leave, funded by a Banco Santader–UC3M Chair of Excellence Award. H. Geffner's work is partially funded by grant TIN-2015-67959-P from MINECO, Spain, and a grant from the Knut and Alice Wallenberg (KAW) Foundation, Sweden.

REFERENCES

[1] Diego Aineto, Sergio Jiménez, Eva Onaindia, and Miquel Ramírez, 'Model recognition as planning', in *Proc. ICAPS*, pp. 13–21, (2019).
[2] Ankuj Arora, Humbert Fiorino, Damien Pellier, Marc Métivier, and Sylvie Pesty, 'A review of learning planning action models', *The Knowledge Engineering Review*, **33**, (2018).
[3] Masataro Asai, 'Unsupervised grounding of plannable first-order logic representation from images', in *Proc. ICAPS*, (2019).
[4] Masataro Asai and Alex Fukunaga, 'Classical planning in deep latent space: Bridging the subsymbolic-symbolic boundary', in *AAAI*, (2018).
[5] Gilles Audemard and Laurent Simon, 'Predicting learnt clauses quality in modern SAT solver', in *Proc. IJCAI*, (2019).
[6] Aniket Nick Bajpai, Sankalp Garg, et al., 'Transfer of deep reactive policies for mdp planning', in *Advances in Neural Information Processing Systems*, pp. 10965–10975, (2018).
[7] Blai Bonet, Guillem Francès, and Hector Geffner, 'Learning features and abstract actions for computing generalized plans', in *Proc. AAAI*, (2019).
[8] Ronen I. Brafman and Moshe Tennenholtz, 'R-max-a general polynomial time algorithm for near-optimal reinforcement learning', *The Journal of Machine Learning Research*, **3**, 213–231, (2003).
[9] Thiago P Bueno, Leliane N de Barros, Denis D Mauá, and Scott Sanner, 'Deep reactive policies for planning in stochastic nonlinear domains', in *AAAI*, volume 33, pp. 7530–7537, (2019).
[10] Maxime Chevalier-Boisvert, Dzmitry Bahdanau, Salem Lahlou, Lucas Willems, Chitwan Saharia, Thien Huu Nguyen, and Yoshua Bengio, 'Babyai: A platform to study the sample efficiency of grounded language learning', in *ICLR*, (2019).
[11] Stephen N Cresswell, Thomas L McCluskey, and Margaret M West, 'Acquiring planning domain models using LOCM', *The Knowledge Engineering Review*, **28**(2), 195–213, (2013).
[12] Adnan Darwiche, 'Human-level intelligence or animal-like abilities?', *Communications of the ACM*, **61**(10), 56–67, (2018).
[13] Carlos Diuk, Andre Cohen, and Michael L Littman, 'An object-oriented representation for efficient reinforcement learning', in *Proc. ICML*, pp. 240–247, (2008).
[14] Alan Fern, SungWook Yoon, and Robert Givan, 'Approximate policy iteration with a policy language bias', in *Advances in neural information processing systems*, pp. 847–854, (2004).
[15] Vincent François-Lavet, Yoshua Bengio, Doina Precup, and Joelle Pineau, 'Combined reinforcement learning via abstract representations', in *Proc. AAAI*, volume 33, pp. 3582–3589, (2019).
[16] Marta Garnelo and Murray Shanahan, 'Reconciling deep learning with symbolic artificial intelligence: representing objects and relations', *Current Opinion in Behavioral Sciences*, **29**, 17–23, (2019).
[17] Hector Geffner, 'Model-free, model-based, and general intelligence', in *IJCAI*, (2018).
[18] Hector Geffner and Blai Bonet, *A concise introduction to models and methods for automated planning*, Morgan & Claypool, 2013.
[19] Edward Groshev, Maxwell Goldstein, Aviv Tamar, Siddharth Srivastava, and Pieter Abbeel, 'Learning generalized reactive policies using deep neural networks', in *Proc. ICAPS*, (2018).
[20] Stevan Harnad, 'The symbol grounding problem', *Physica D: Nonlinear Phenomena*, **42**(1-3), 335–346, (1990).
[21] Patrik Haslum, Nir Lipovetzky, Daniele Magazzeni, and Christian Muise, *An Introduction to the Planning Domain Definition Language*, Morgan & Claypool, 2019.
[22] Murugeswari Issakkimuthu, Alan Fern, and Prasad Tadepalli, 'Training deep reactive policies for probabilistic planning problems', in *ICAPS*, (2018).
[23] Roni Khardon, 'Learning action strategies for planning domains', *Artificial Intelligence*, **113**(1-2), 125–148, (1999).
[24] George Konidaris, Leslie Pack Kaelbling, and Tomas Lozano-Perez, 'From skills to symbols: Learning symbolic representations for abstract high-level planning', *Journal of Artificial Intelligence Research*, **61**, 215–289, (2018).
[25] Brenden M Lake and Marco Baroni, 'Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks', *arXiv preprint arXiv:1711.00350*, (2017).
[26] Brenden M Lake, Tomer D Ullman, Joshua B Tenenbaum, and Samuel J Gershman, 'Building machines that learn and think like people', *Behavioral and Brain Sciences*, **40**, (2017).
[27] Gary Marcus, 'Deep learning: A critical appraisal', *arXiv preprint arXiv:1801.00631*, (2018).
[28] Mario Martín and Hector Geffner, 'Learning generalized policies from planning examples using concept languages', *Applied Intelligence*, **20**(1), 9–19, (2004).
[29] Drew McDermott, 'The 1998 AI Planning Systems Competition', *Artificial Intelligence Magazine*, **21**(2), 35–56, (2000).
[30] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al., 'Human-level control through deep reinforcement learning', *Nature*, **518**(7540), 529, (2015).
[31] Stephen Muggleton and Luc De Raedt, 'Inductive logic programming: Theory and methods', *The Journal of Logic Programming*, **19**, 629–679, (1994).
[32] Judea Pearl, 'Theoretical impediments to machine learning with seven sparks from the causal revolution', *arXiv preprint arXiv:1801.04016*, (2018).
[33] Judea Pearl and Dana Mackenzie, *The book of why: the new science of cause and effect*, Basic Books, 2018.
[34] Richard Sutton and Andrew Barto, *Introduction to Reinforcement Learning*, MIT Press, 1998.
[35] Valentin Thomas, Emmanuel Bengio, William Fedus, Jules Pondard, Philippe Beaudoin, Hugo Larochelle, Joelle Pineau, Doina Precup, and Yoshua Bengio, 'Disentangling the independently controllable factors of variation by interacting with the world', *arXiv preprint arXiv:1802.09484*, (2018).
[36] Sam Toyer, Felipe Trevizan, Sylvie Thiébaux, and Lexing Xie, 'Action schema networks: Generalised policies with deep learning', in *AAAI*, (2018).
[37] Qiang Yang, Kangheng Wu, and Yunfei Jiang, 'Learning action models from plan examples using weighted max-sat', *Artificial Intelligence*, **171**(2-3), 107–143, (2007).
[38] Luke S Zettlemoyer, Hanna Pasula, and Leslie Pack Kaelbling, 'Learning planning rules in noisy stochastic worlds', in *AAAI*, pp. 911–918, (2005).