Proceedings of the Thirty-First International Conference on Automated Planning and Scheduling (ICAPS 2021)

RePReL : Integrating Relational Planning and Reinforcement Learning for Effective Abstraction

Harsha Kokel,[1] **Arjun Manoharan,**[2] **Sriraam Natarajan,**[1]
Balaraman Ravindran,[2] **Prasad Tadepalli** [3]

[1]The University of Texas at Dallas,
[2]Robert Bosch Centre for Data Science and Artificial Intelligence at Indian Institute of Technology Madras,
[3]Oregon State University
{hkokel,Sriraam.Natarajan}@utdallas.edu, {arjunman,ravi}@cse.iitm.ac.in, tadepall@eecs.oregonstate.edu

Abstract

State abstraction is necessary for better task transfer in complex reinforcement learning environments. Inspired by the benefit of state abstraction in MAXQ and building upon hybrid planner-RL architectures, we propose RePReL, a hierarchical framework that leverages a relational planner to provide useful state abstractions. Our experiments demonstrate that the abstractions enable faster learning and efficient transfer across tasks. More importantly, our framework enables the application of standard RL approaches for learning in structured domains. The benefit of using the state abstractions is critical in relational settings, where the number and/or types of objects are not fixed apriori. Our experiments clearly show that RePReL framework not only achieves better performance and efficient learning on the task at hand but also demonstrates better generalization to unseen tasks.

Introduction

Planning and Reinforcement Learning have been two major thrusts of AI aimed at sequential decision making. While classical relational planning focuses on composing sequences of high level actions offline before any execution, reinforcement learning interleaves planning and execution and is typically associated with reactive domains with unknown dynamics. We describe an integrated architecture we call "RePReL," which combines relational planning (RP) and reinforcement learning (RL) in a way that exploits their complementary strengths and not only speeds up the convergence compared to a traditional RL solution but also enables effective transfer of the solutions over multiple tasks.

Most prior work in combining planning and RL falls under the general paradigm of "model-based reinforcement learning" (MBRL). Here explicit dynamic models of actions are learned via exploration and used either offline to compute approximately optimal policies (Brafman and Tennenholtz 2002; Guestrin, Patrascu, and Schuurmans 2002) or online in look-ahead search (Silver, Hubert et al. 2018). Critically, both planning and reinforcement learning components employ the same state-space, while the main motivation for the combination comes from the benefits of efficiency and cost-savings due to offline computation or look-ahead search.

However, in many real world domains, e.g., driving, the state space of offline planning is rather different from the state space of online execution. Planning typically occurs at the level of deciding the route, while online execution needs to take into account dynamic conditions such as locations of other cars and traffic lights. Indeed, the agent typically does not have access to the dynamic part of the state at the planning time, e.g., future locations of other cars, nor does it have the computational resources to plan an optimal policy in advance that works for all possible traffic events.

The key principle that enables agents to deal with these informational and computational challenges is *abstraction*. In the driving example, the high level state space consists of coarse locations such as "O'hare airport" and high level actions such as take "Exit 205," while the lower level state space consists of a more precise location and velocity of the car and actions such as turning the steering wheel by some amount and applying brakes. Importantly, excepting occasional unforeseen failures, the two levels operate independently of each other and depend on different kinds of information available at different times. This allows the agent to tractably plan at a high level without needing to know the exact state at the time of the execution, and behave appropriately during plan execution by only paying attention to a small dynamic part of the state.

The power of abstraction and hierarchies have been studied both in hierarchical planning (Nau et al. 1999; Georgievski and Aiello 2015) and hierarchical reinforcement learning (Dietterich 2000; Sutton, Precup, and Singh 1998; Kulkarni et al. 2016). However, while the planning methods typically emphasize relational representations and deterministic worlds, the RL approaches typically employ propositional representations and stochastic environments. Another important difference is that planning requires knowing a goal, while the RL agent assumes a fixed unknown reward function. The key contribution of the current paper is the RePReL architecture, which consists of a high level relational planner and a low level reinforcement learner. The high level planner is itself hierarchical that allows it to further take advantage of multi-level abstractions. The relational level planner plans to achieve its goal using a

sequence of subgoals, which are passed onto the reinforcement learning agent. The reinforcement learning agent then tries to reach its assigned sub-goal with minimum path cost.

One of the contributions of our work is the adaptation of first-order conditional influence (FOCI) statements (Natarajan et al. 2008) to specify bisimilarity conditions of MDPs (Givan, Dean, and Greig 2003), which in turn help justify safe and effective abstractions for reinforcement learning (Ravindran and Barto 2003). Secondly, since the RL agent learns to optimize policies to achieve subgoals that occur in multiple high-level tasks, it naturally leads to effective transfer of learned skills from one task to another. Finally, the use of relational representation for planning allows appropriate generalization including the number and types of objects in the domain without excessive feature engineering. Our results in 4 compelling domains show that RePReL significantly outperforms the state-of-the-art Planner+RL combination (Illanes et al. 2020) while achieving better generalization and transfer.

The rest of the paper is organized as follows: Next section summarizes the relevant related work in planning and reinforcement learning. Then we describe the RePReL architecture in detail. Empirical Evaluation section describes experiments in 4 different domains that demonstrate generalization and transfer across multiple tasks. The final section concludes and presents some future directions.

Related Work

Planner and RL combination The RePReL framework consists of a symbolic planner at the higher level and various RL agents at the ground level. Several prior works have explored the idea of combining a planner and RL agents to solve complex problems which have some notion of temporally extended actions or task hierarchies (Grounds and Kudenko 2005; Yang et al. 2018; Lyu et al. 2019; Jiang et al. 2019; Eppe, Nguyen, and Wermter 2019). Among these, RePReL is closely related to the Taskable RL framework of Illanes et al. (2020). Similar to Taskable RL , RePReL employs a planner to generate useful instructions (task definitions) for the RL agent. RePReL extends the Taskable RL framework in two key ways: first, we generalize the Taskable RL to a relational MDP representation and second, we propose an approach to define task-specific state abstraction in this framework. As shown in our experiments, both these aspects are critical for effective transfer and generalization.

Abstraction Safe and efficient state abstraction techniques have been studied extensively in RL (Li, Walsh, and Littman 2006). They have been particularly useful for multi-task and transfer learning problems (Walsh, Li, and Littman 2006; Sorg and Singh 2009; Abel et al. 2018). We are inspired by the task-specific, model-agnostic state abstractions of MAXQ (Dietterich 2000) and the bisimulation conditions (Ravindran and Barto 2003; Givan, Dean, and Greig 2003) to define abstractions in relational settings using first-order probabilistic models (Raedt et al. 2016). Andre and Russell (2002) introduces ALisp language to assert (ir)relevant variable. Our work is diverges from their work as we consider relational setting. Another work by Finzi and Lukasiewicz (2006) leverages situation calculus for explicating the abstraction in relational setting. We, however, approach this problem using relational probabilistic model.

Relational Reinforcement Learning (RRL) RRL methods aim to learn optimal behavior in relational worlds described by objects and relations (Price and Boutilier 2001; Sanner and Boutilier 2009; Tadepalli, , and Driessens 2004; Das et al. 2020; Guestrin et al. 2003). While specific methodologies differ, most of these methods derive an approximate value function or policy over relational representations of states either through relational induction, symbolic dynamic programming, or linear programming alternatives. While expressive, learning these models is a very cumbersome task and thus their adaptation to any mildly complex domains is still modest. While these methods employ a relational learner to learn a single policy, our 2-level architecture employs the relational planner and provides state abstractions for RL at the lower level.

Relational Planning and Reinforcement Learning

We consider the problem of learning to act in relational domains with varying number of tasks and interacting objects. Before explaining the formulation of the RePReL framework, we motivate it with a simple toy example in Figure 1. Consider a taxi domain where the goal is to transport the different passengers ($p1, p2$) to their respective destinations ($d1, d2$). In RePReL, the high-level planner first decomposes the goal into appropriate subgoals. In the example, the goal of transporting $p1$ and $p2$ is decomposed into 4 subgoals: pickup $p1$, drop $p1$, pickup $p2$, and drop $p2$. An RL policy at lower level then achieves these subgoals by navigating in the grid and picking or dropping the passenger. *The symbolic planner receives high-level domain knowledge, and does not see the complete map of the domain. The RL agent sees the map but is agnostic to other irrelevant information.* For e.g., the RL policy that is performing pickup $p1$ subgoal, needs to know the location of $p1$ and whether the taxi is free, while passenger $p2$ and destination of $p1$ are irrelevant. Similarly, the RL policy performing drop $p2$ needs to ensure the taxi is hired by $p2$ while the pickup location of $p2$ is irrelevant. It has been argued that for human-level general intelligence, the ability to detect compositional structure in the domain (Lake, Salakhutdinov, and Tenenbaum 2015) and form task-specific abstractions (Konidaris 2019) are necessary. With RePReL's hierarchical planner and task-specific abstractions, we propose a step in that direction and formalize the framework next.

Relational Hierarchical Planning

We extend the relational MDP (RMDP) definition of Fern, Yoon, and Givan (2006) for goal-oriented domains.

Definition 1. *A goal-directed relational MDP (GRMDP) $\mathcal{M}$ is represented by $\langle S, A, P, R, \gamma, G\rangle$, where the states S and the actions A are represented by a set of objects E, a set of predicates Q, and action types Y. P is a probability transition function $S \times A \times S \rightarrow [0, 1]$, R is a reward function*

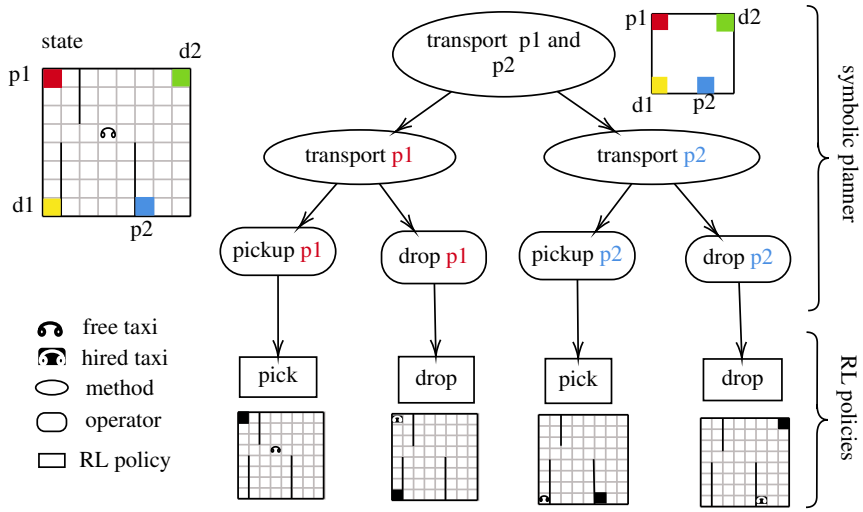


Figure 1: Example of RePReL framework in Taxi domain. The domain has two passengers $p1$ and $p2$ and task it to transport both of them to their respective destination, $d1$ and $d2$ respectively. The initial state is shown on the top left.

$S \times A \times S \to \Re$, *and* $\gamma \in [0, 1)$ *is the discount factor. G is set of goals that the agent may be asked to achieve.*

Different tasks can be formulated by choosing different goals from the potentially infinite set G. The reward function R provides the reward (or cost) of taking a step in the environment, regardless of the goal. A problem instance for a GRMDP is defined by a pair $\langle s \in S, g \in G \rangle$, where s is a state and g is the set of goal condition, both represented using sets of permitted literals, i.e. positive and/or negative atoms. A solution is a policy that starts from s and ends in a state satisfying g with probability 1.

One novelty of our framework is that the proposed GRMDPs are solved using a combination of hierarchical planning and reinforcement learning. The hierarchical planner decomposes goals into a sequence of subgoals using decomposition rules or *methods*. The subgoals are recursively decomposed further until the level of *planning operators*, yielding a *hierarchical plan*. The planning operators (called "operators" from now on) are in turn implemented by the reinforcement learner as a policy that maps concrete states to actions. Importantly, the hierarchical planner assumes a relational deterministic model of operators, whereas the reinforcement learner allows stochastic actions. This is similar to the real world example of driving, where the route planning assumes deterministic dynamics, whereas actual driving needs to react to stochastic events. Due to the potentially infinite set of goals and the nature of hierarchical planning, we cannot guarantee optimality of overall policies that result from the combination. However, under suitable safe abstraction conditions, we can guarantee that the RL policies that implement the operators are optimal.

An operator $o \in O$ is a tuple $\langle \mathtt{p}(o), \mathtt{pre}(o), \mathtt{eff}(o), \beta(o) \rangle$, where $\mathtt{p}$ is a primitive task and $\mathtt{pre}$, $\mathtt{eff}$, and β are first-order formulae for preconditions, effects, and termination condition of the operator o, respectively. The positive and negative conditions in these formulae are represented using superscript $^+$ and $^-$, respectively. For instance, $\mathtt{pre}^+(o)$ is set of predicates in precondition of operator o that must be $\mathtt{True}$, and $\mathtt{pre}^-(o)$ must be $\mathtt{False}$. A standard assumption in planning is that all the literals in the formula $\mathtt{eff}(o)$ and $\beta(o)$ appear either as existential quantifier in the $\mathtt{pre}(o)$ or as parameters of $\mathtt{p}(o)$.

An operator o can be applied in a state s if a substitution θ satisfies the first-order formula $\mathtt{pre}(o)$ at s. On applying a grounded operator o_g, s transitions to state $s' = s \backslash \mathtt{eff}^-(o)\theta \cup \mathtt{eff}^+(o)\theta$. A *plan* for a problem $\langle s \in S, g \in G \rangle$ is a sequence of grounded operators, which when executed in state s result in a state satisfying g. We formally define the hierarchical planner that produces such plans.

Definition 2. *A hierarchical planner assumes a set of domain predicates Q, a set of methods M, and a set of operators O. A method* (m) *is a triple* $\langle c(m), \mathtt{pre}(m), \tau(m) \rangle$ *where* c *is a compound task,* $\mathtt{pre}$ *is a method pre-condition, and* τ *is a task sequence. Each task in* τ *can be a compound task or an operator in O. A hierarchical planner for a problem* $\langle s, g \rangle$ *constructs a hierarchical plan by recursively decomposing the goal task using the methods in* M *to find a sequence of grounded operators in O that achieves g.*

To decompose some task using method m, planner needs a substitution θ satisfying preconditions $\mathtt{pre}(m)$ in s. Once found, planner recursively decomposes $\tau(m)\theta$. To summarise, the planner takes a planning problem $\mathcal{P} = \langle \mathcal{D}, s, g \rangle$, where domain $\mathcal{D} = \langle Q, O, M \rangle$, and returns a sequence of grounded operators that achieves g starting from s. Figure 2 shows an example with the planner methods and operators, along with a sample initial state s, goal condition g and plan Π.[1] *A key difference to typical hierarchical planners is that in our case, lowest level operators do not execute an atomic action. Instead, these operators are tasks that requires an RL agent to solve them by executing a policy.*

[1]We use uppercase to denote variables (for ex., $\exists X$, $\mathtt{in\text{-}taxi}(X)$) and lower case to denote constants (for ex., $\mathtt{in\text{-}taxi}(p1)$)

Methods:
⟨**c:** transport(G) # PASSENGER IN TAXI
pre: ∃X, (at-dest(X) ∈ G)
τ: {drop(X), transport(G)} ⟩
⟨**c:** transport(G) # NOT AT DESTINATION
pre: ∃X, at-dest(X) ∈ G ∧¬at-dest(X)
τ: {pickup(X), transport(G)} ⟩
Operators:
⟨ **p:** pickup(X)
pre: ∃L, at(X, L) ∧ ¬in-taxi(X)
eff: taxi-at(L) ∧ in-taxi(X)
β: in-taxi(X) ⟩
⟨ **p:** drop(X)
pre: ∃L, in-taxi(X) ∧ dest(X, L),
eff: taxi-at(L) ∧ at-dest(X)
∧¬in-taxi(X)
β: at-dest(X) ⟩
Initial State (s): {at(p1,r),
taxi-at(l3), dest(p1,d1),
¬at-dest(p1), ¬in-taxi(p1)}
Goal Set (g): {at-dest(p1)}
Plan (Π): [⟨pickup(X), {X/p1,L/r}⟩,
⟨drop(X), {X/p1, L/d1}⟩]

Figure 2: Example of a taxi domain. Planner declarations, Initial state, Goal set, and Plan.

Safe State Abstraction for RL

We now address the topic of safely abstracting the state to allow the RL agent to implement operators. First, we define subgoal RMDPs which speciailize GRMDPs to operators.

Definition 3. *The subgoal RMDP M_o for each operator o is defined by the tuple $\langle S, A, P_o, R_o, \gamma \rangle$ consisting of states S, actions A, transition function P_o, reward function R_o, and discount factor γ. State and Actions remain same as the original RMDP. The reward function R_o and transition probability distribution function P_o are defined as follows:*

$$R_o(s,a,s') = \begin{cases} t_R + R(s,a,s') & \text{if } s' \in \beta(o) \text{ and } s \notin \beta(o) \\ 0 & \text{if } s' \in \beta(o) \text{ and } s \in \beta(o) \\ R(s,a,s') & \text{otherwise} \end{cases}$$

$$P_o(s,a,s') = \begin{cases} 0 & \text{if } s \in \beta(o) \text{ and } s' \notin \beta(o) \\ 1 & \text{if } s \in \beta(o) \text{ and } s' \in \beta(o) \\ P(s,a,s') & \text{otherwise} \end{cases}$$

with $R(s,a,s')$ indicating the reward function from the original GRMDP definition. t_R is a fixed terminal reward.

In short, the reward function in the original GRMDP would correspond to the step cost function which applies to all operators, and reward R_o is the only goal-specific reward.

Several versions of safe state abstraction have been studied in the MDP literature (Givan, Dean, and Greig 2003; Dietterich 2000; Ravindran and Barto 2003; Li, Walsh, and Littman 2006). We adopt the bisimulation framework of Givan, Dean, and Greig (2003) and Ravindran and Barto (2003), which has been called "model agnostic abstraction" in Li, Walsh, and Littman (2006).

Definition 4 (Li, Walsh, and Littman (2006))**.** *A model-agnostic abstraction $\phi(s)$ is such that for any action a and abstract state $\overline{s}$, $\phi(s_1) = \phi(s_2)$ if and only if*

$$\sum_{\{s_1' | \phi(s_1') = \overline{s}\}} R_o(s_1, a, s_1') = \sum_{\{s_2' | \phi(s_2') = \overline{s}\}} R_o(s_2, a, s_2')$$

$$\sum_{\{s_1' | \phi(s_1') = \overline{s}\}} P_o(s_1, a, s_1') = \sum_{\{s_2' | \phi(s_2') = \overline{s}\}} P_o(s_2, a, s_2')$$

The first condition above states that the two states s_1 and s_2 have the same immediate reward distribution with respect to the abstraction whereas the second condition states that they have the same transition dynamics. It is proven that Q-learning with such abstraction results in an optimal policy for ground MDP (Li, Walsh, and Littman 2006).

Since states are conjunctions of literals in RMDPs, we need to reason about how the actions influence the state predicates and how rewards are influenced by goal predicates and actions to decide which literals should be included and excluded in the abstraction. We capture this knowledge using First-Order Conditional Influence (FOCI) statements (Natarajan et al. 2008), one of the many variants of statistical relational learning languages (Getoor and Taskar 2007; Raedt et al. 2016). Each FOCI statement is of the form: "if `condition` then $\mathtt{X}_1$ influence $\mathtt{X}_2$", where, `condition` and $\mathtt{X}_1$ are a set of first-order literals and $\mathtt{X}_2$ is a single literal. It encodes the information that literal $\mathtt{X}_2$ is influenced only by the literals in $\mathtt{X}_1$ when the stated `condition` is satisfied. For RePReL, we simplify the syntax and extend FOCI to dynamic FOCI (D-FOCI) statements. In addition to direct influences in the same time step, D-FOCI statements also describe the direct influences of the literals in the current time step on the literals in the next time step. To distinguish the two kinds of influences, we show a $+1$ on the arrow between the sets of literals to capture a temporal interaction, as shown below.

$$\mathtt{operator} : \{\mathrm{p}(X_1), q(X_1)\} \xrightarrow{+1} \mathrm{q}(X_1)$$

It says that, for the given `operator`, the literal $q(\mathtt{X}_1)$ in the next time step is directly influenced only by the literals $\{p(\mathtt{X}_1), q(\mathtt{X}_1)\}$. Following the standard DBN representation of MDP, we allow action variables and the reward variables in the two sets of literals. To represent unconditional influences between state predicates, we skip the `operator`.

The D-FOCI statements can be viewed as relational versions of dynamic Bayesian networks (DBNs) and have a similar function of capturing the conditional independence relationships between domain predicates at different time steps. For example, the D-FOCI statement in the taxi domain for `pickup` operator can be expressed as,

$$\mathtt{pickup(P)} : \{\mathtt{taxi\text{-}at(L1)}, \mathtt{at(P,\ L)}\} \xrightarrow{+1} \mathtt{in\text{-}taxi(P)}$$

This denotes that while the task `pickup(P)` is being performed, only the taxi location and the passenger location (`taxi-at(L1)` and `at(P,L2)`) influence `in-taxi(P)`. This means that the `in-taxi(P)` is independent of `dest(P,D)` and `at-dest(P)`. Note that

contrary to this, when the passenger is being dropped, the `dest(P,D)` will influence `in-taxi(P)`.

Using the substitution θ of the grounded operator determined by the planner, the D-FOCI statements would be partially grounded. For example, using the $\theta = \{X/p1, L/r\}$ for `pickup(X)` in Figure 2, above D-FOCI would become

$$\texttt{pickup}(p1) : \{\texttt{taxi-at(L1)}, \texttt{at}(p1, r)\} \xrightarrow{+1} \texttt{in-taxi}(p1)$$

Hence, only the pickup location of passenger $p1$ (i.e., $\texttt{at}(p1, r)$) and current taxi location `taxi-at(L1)` (i.e., the current location `L1`) are relevant for grounded operator $\texttt{pickup}(p1)$. If there is another passenger, say $p2$, then the state variable $\texttt{at}(p2, \cdot)$ would not be relevant.

Learning Optimal Q-Functions

While the planner works in relational representations, the reinforcement learning operates at a propositional level. The gap is bridged by computing an appropriate propositional abstraction of the state for each operator with the parameters, e.g., $p1$, bound to generic objects (Skolem constants in logic). To keep the size of the propositional representation bounded, we bound the depth of inference chain through D-FOCI statements to k.

The propositionalization proceeds by instantiating each D-FOCI statement with generic objects yielding a structure equivalent to a propositional DBN. A model-agnostic abstraction is derived for each operator by iteratively adding the literals that influence the relevant literals through all actions starting with the reward variables R and R_o. Refer appendix for an example.[2] We limit this unrolling process to at most $k = 2$ levels and at most 1 time step to keep the size of the propositionalization bounded. The set of all such literals form the final abstraction. The unrolling depth will impact the time complexity of computing the abstractions.

Theorem 1. *If the MDP satisfies the D-FOCI statements with a fixed depth unrolling, then the corresponding model-agnostic abstraction has the same optimal value function as the fully instantiated MDP.*

Proof (sketch): The fixed depth unrolling assumption ensures that the propositional DBN obtained by unrolling the D-FOCI statements for a fixed depth forms a model-agnostic abstraction of the MDP. The result then follows from the fact that model-agnostic abstractions of the MDP preserve its value function (Li, Walsh, and Littman 2006). □

The complete list of D-FOCI statements and relevant state abstractions derived from it are provided in Table 1.

Given the GRMDP environment env, planner $\mathfrak{P}$ and FOCI statements F, we now discuss the RePReL learning procedure from **Algorithm 1**. First for each operator, an RL policy is initialized in **line 1**. Next for each episode, in **line 4**, we get the high level plan Π from the planner $\mathfrak{P}$, we employ the SHOP planner (Nau et al. 1999). For every ground operator in the plan, we train the RL policy π_o (**lines 6–20**). To train the RL policy, we get an abstract propositional state representation $\hat{s}$ from state s as described in previous paragraphs. In **lines 10–19**, we obtain action a from the current policy, perform that action, observe the next state s' and reward. If s' is a terminal state for the ground operator o_g, then we add a terminal reward t_R (**line 16**) before updating the policy/q-value (**line 18**). In our experiments, we employ tabular q-learning over the propositional state space for updating the values and consequently, the policy. We repeat the process for fixed budget of episodes for our evaluation but various other stopping criteria can also be used.

Algorithm 1 RePReL Learning Algorithm

INPUT: $\mathfrak{P}(O, M)$, goal set g, env, t_R, F
OUTPUT: RL policies $\pi_o, \forall o \in O$

1: $\pi_o \leftarrow 0, \forall o \in O$ ▷ initialize RL policy for each operator
2: **for** each episode **do**
3: $s \leftarrow$ get state from env
4: $\Pi \leftarrow \mathfrak{P}(s, g)$ ▷ get high-level plan
5: **for** o_g in Π **do**
6: $\pi \leftarrow \pi_o$ ▷ get resp. RL policy
7: $\hat{s} \leftarrow$ GetAbstractState(s, o_g, F)
8: done $\leftarrow \hat{s} \in \beta(o_g)$ ▷ check terminal state
9: **while** not done **do**
10: $a \leftarrow \pi(\hat{s})$ ▷ get action
11: $s' \leftarrow$ env.step(a) ▷ take step in env
12: $r \leftarrow R(s, a, s')$ ▷ get step reward
13: $\hat{s}' \leftarrow$ GetAbstractState(s, o_g, F)
14: done $\leftarrow \hat{s}' \in \beta(o_g)$ ▷ check terminal next state
15: **if** done **then**
16: $r = r + t_R$ ▷ add terminal reward
17: **end if**
18: $\pi.update(\hat{s}, a, \hat{s}', r)$ ▷ update policy
19: $s, \hat{s} \leftarrow s', \hat{s}'$
20: **end while**
21: **end for**
22: **end for**
23: **return** $\pi_o, \forall o \in O$

Empirical Evaluation

We aim to empirically answer the following questions.

Q1: Sample Efficiency: Do the abstractions induced in RePReL improve sample efficiency?

Q2: Transfer: Do these abstractions allow for effective transfer?

Q3: Generalization: Does RePReL efficiently generalize to varying number of objects?

We evaluate RePReL on the following 4 environments.[3]

1. Craft World: This is a Minecraft inspired multitask grid-world from Andreas, Klein, and Levine (2017). It has some raw materials and three work locations. Tasks are: **1.** get wood and iron, **2.** make sticks, **3.** make axe, **4.** mine gem. These tasks can be viewed as *curriculum learning* since they are incremental. To mine the gem, agent needs axe, and an axe can be obtained by visiting the tool-shed with a stick and iron. Stick is obtained by first collecting wood and then visiting the workbench.

2. Office World: This is a multitask grid-based environment from Illanes et al. (2020). It has inaccessible and accessible

[2] link to appendix: https://starling.utdallas.edu/papers/RePReL.

[3] refer appendix for code, environment details and visuals.

Domain	D-FOCI statements	Operators	Set of relevant state predicates
CraftsWorld	{agent-at(L1), move(Dir)} $\xrightarrow{+1}$ agent-at(L2) {agent-at(L1), move(Dir)} $\longrightarrow R$ {with-agent(Y), require(X, Y)} $\xrightarrow{+1}$ require(X, Y) pickup(X):with-agent(X) $\longrightarrow R_o$ pickup(X):{agent-at(L1), at(X, L), with-agent(X)} $\longrightarrow$ with-agent(X) build(X): {agent-at(L1), build-at(X, L), require(X,Y), with-agent(Y)} $\xrightarrow{+1}$ with-agent(X) build(X): with-agent(X) $\longrightarrow R_o$	pickup(X)	{agent-at(L1), at(X, L), with-agent(X), move(Dir)}
		build(X)	{agent-at(L1), with-agent(X), build-at(X, L), require(X, Y), with-agent(Y), move(Dir)}
OfficeWorld	{agent-at(L1), move(Dir)} $\xrightarrow{+1}$ agent-at(L2) {agent-at(L1), move(Dir)} $\longrightarrow R$ pickup(X): {agent-at(L1), at(X, L), with-agent(X)} $\xrightarrow{+1}$ with-agent(X) pickup(X): with-agent(X) $\longrightarrow R_o$ deliver(X): {agent-at(L1), with-agent(X), office(L), delivered(X)} $\xrightarrow{+1}$ delivered(X) deliver(X): delivered(X) $\longrightarrow R_o$	pickup(X)	{agent-at(L1), at(X, L), with-agent(X), move(Dir)}
		deliver(X)	{agent-at(L1), with-agent(X), office(L), move(Dir), delivered(X)}
Taxi	{taxi-at(L1), move(Dir)} $\xrightarrow{+1}$ taxi-at(L2) {taxi-at(L1), move(Dir)} $\longrightarrow R$ pickup(P): {taxi-at(L1), at(P, L), in-taxi(P)} $\xrightarrow{+1}$ in-taxi(P) pickup(P): in-taxi(P) $\longrightarrow R_o$ drop(P): {taxi-at(L1), in-taxi(P), dest(P,L), at-dest(P)} $\xrightarrow{+1}$ at-dest(P) drop(P): at-dest(P) $\longrightarrow R_o$	pickup(P)	{taxi-at(L1), at(P,L), in-taxi(P), move(Dir)}
		drop(P)	{taxi-at(L1), in-taxi(P), dest(P,L), at-dest(P), move(Dir)}
BoxWorld	{neighbor(Dir,C), agent-at(L2), move(D)} $\xrightarrow{+1}$ agent-at(L1) {neighbor(Dir,C), agent-at(L1), move(D)} $\longrightarrow R$ pick_key(K): own(K) $\longrightarrow R_o$ pick_key(K): {agent-at(L1), direction(K, Dir2), own(K)} $\xrightarrow{+1}$ own(K) unlock(L): open(L) $\longrightarrow R_o$ unlock(L): {agent-at(L1), direction(L, Dir2), open(L)} $\xrightarrow{+1}$ open(L)	pick_key(K)	{neighbor(Dir,C), agent-at(L1), direction(K, Dir2), own(K), move(D)}
		unlock(L)	{neighbor(Dir,C), agent-at(L1), direction(L, Dir2), open(L),move(D)}

Table 1: D-FOCI statments and relevant state predicates for all the domains

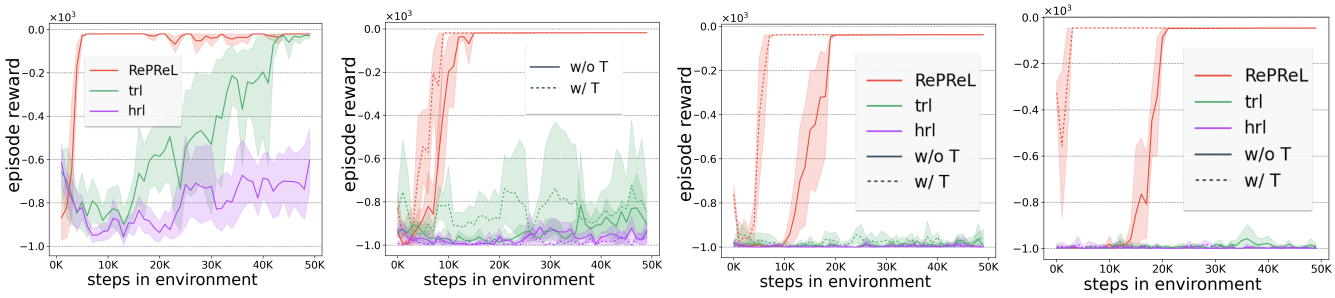
locations. The 4 tasks include: **1.** deliver mail to office, **2.** deliver coffee to office, **3.** deliver mail and coffee, and **4.** visit locations A, B, C, D.

3. Extended Taxi World: We extend the Taxi domain by Dietterich (2000) with three passengers and the relational representation as shown in the earlier example. The taxi can be hired by one passenger at a time. Task **1** is to drop one passenger ($p1$), Task **2** is to drop two passengers ($p1$, $p2$), and Task **3** is to drop all three passengers ($p1$, $p2$, and $p3$) to their respective destinations in that order. This domain has *higher complexity* than previous domains. In both the Craft and Office World, every object except player has a fixed location. Here, along with the player/taxi location, the passenger pick-up and drop locations are also randomly sampled at the beginning of each episode from R, G, B, or Y.

4. Relational Box World: This environment is inspired by Box World from Zambaldi et al. (2019). There are 4 types of objects: lock, key, gem, wall with an associated color. A lock can be opened with a key of the same color and the player has to open a lock to reach the key inside that box. The player is equipped with sensors on each of its 8 directions ($NE, E, S, ...$), which detects the relative direction of the objects. Unlike the image representation, in our setting, *the complete grid is not visible to the agent*. The goal, in each task, is to collect the gem. Task **1** has a lock containing the gem, the agent is initialized with the key to open the lock. In Task **2** agent has to first collect the key and then open the lock to collect the gem. Finally, Task **3** requires the agent to open two locks in sequence to reach the gem. This is a combinatorially complex domain, with 18 possible colors. The color and the location of the locks and keys are sampled at the beginning of each episode.

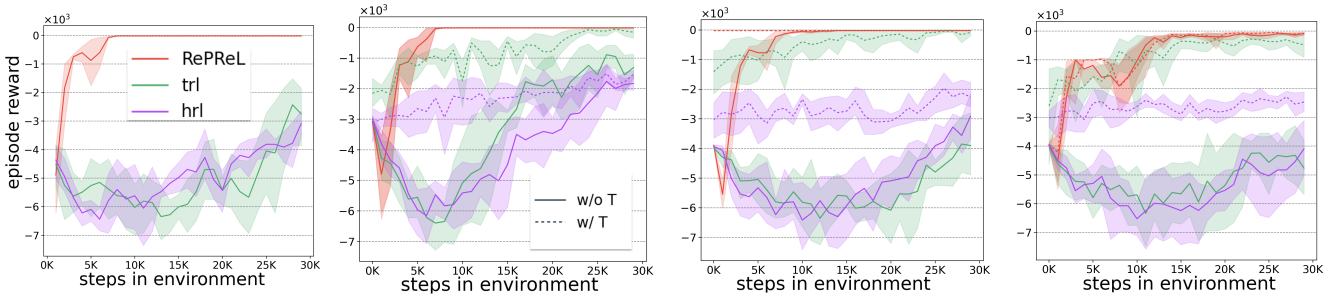
For each of these domains, the operators and the relevant state variables are shown in the Table. 1

[Q1] Sample Efficiency: To evaluate the effect of RePReL abstractions, we compare it against the seq variant of the Taskable RL. We pick this variant for two reasons: 1. seq variant performed best in all their experiments, 2. We aim to evaluate the effectiveness of abstractions and thus do not learn the meta-controller introduced by the partially ordered



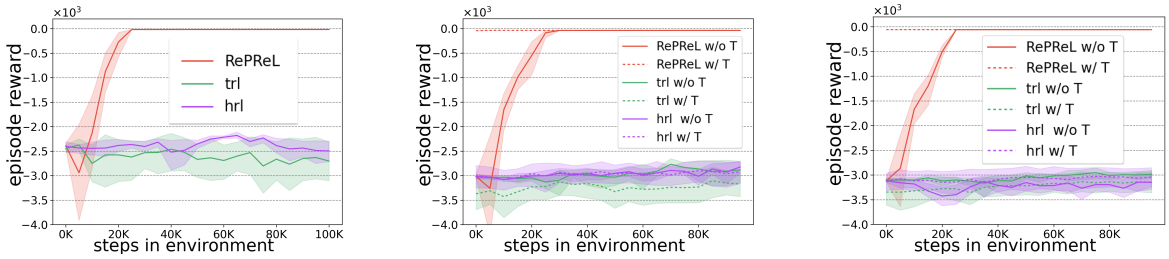
(a) Task 1 (b) Task 2 (c) Task 3 (d) Task 4

Figure 3: Comparing learning curves of RePReL, trl, and hrl in Craft World environment. Results with transfer (w/ T) are indicated by dashed lines. *Task 1* is to get wood and iron, *Task 2* is make stick, *Task 3* is make axe, *Task 4* is mine gem.



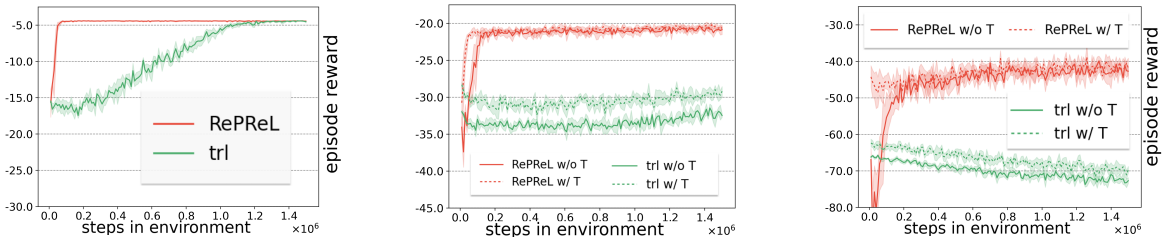
(a) Task 1 (b) Task 2 (c) Task 3 (d) Task 4

Figure 4: Comparing learning curves of RePReL, trl, and hrl in Office World environment. Transfer results (w/ T) are indicated by dashed lines. *Task 1* is deliver mail, *Task 2* is deliver coffee, *Task 3* is deliver mail and coffee, and *Task 4* is visit A, B, C, D. Note that the RePReL with and without transfer curves in Task 2 are overlapping.



(a) Task 1 (b) Task 2 (c) Task 3

Figure 5: Comparing learning curves of RePReL, trl, and hrl in the Extended Taxi World. *Task 1* is to drop passenger p1, *Task 2* is to drop p1 and p2, *Task 3* is to drop p1, p2, p3.



(a) Task 1 (b) Task 2 (c) Task 3

Figure 6: Comparing learning curves of RePReL and trl with and without transfer in Box World environment. Goal in all the tasks is to collect the gem, we increase the number of objects to reach the gem in each task.

plans. We set a budget on number of steps for learning in each task and evaluate the performance of RePReL against Taskable RL (trl) and option-based Hierarchical RL (hrl).

Fig. 3 presents the performance of RePReL and the baselines in **Craft World** environment with $50K$ steps budget (consider only the solid lines in the plots, they are without transfer (w/o T)). All the Figures report aggregated results over 5 runs. Here, trl and hrl have 8 distinct options, one for each location in the domain. RePReL has two policies; one for each operator shown in Tab. 1. RePReL **significantly outperforms the baselines in all the tasks**.

Figure 4 compares the learning curves of RePReL with trl and hrl agents in **Office World** with $30K$ budget. While trl and hrl use 7 different options for traveling to each location in the domain, our RePReL framework uses **only two** options; one each for pickup and deliver operator. We define a common operator for pickup and visit operation as they do not have any preconditions and have the same effects. Fig. 4a shows that RePReL achieves the optimal reward for *Task 1* in less than $10K$ steps, while the baseline methods trl and hrl do not achieve optimality even after $30K$ steps. Similarly in all the other tasks, we see that RePReL consistently outperforms trl and hrl by converging to the optimal policy in less than $15K$ steps. Hence, we answer **Q1** affirmatively in that RePReL abstractions help in statistically significantly outperforming the state-of-the-art hybrid planner-RL architecture, Taskable RL.

[Q2] Transfer: To evaluate the transfer, we modify the RePReL learning algorithm. Specifically, the RL policies are not initialized with 0 (line 1 in Algorithm 1), and instead, we transfer the learned policies from *Task 1* to *Task 2*, refine the policy on *Task 2*, transfer it to *Task 3*, and so on, in increasing order. We present the agent performance with transfer (w/ T) in the plots with dashed lines. We use the same budget as previous set of experiments.

Our experiments (Fig. 3 and 4) clearly demonstrate that transferring the RePReL policies across different tasks has a distinct advantage in terms of sample efficiency. This advantage is more pronounced in tasks that are closely related to prior tasks. For e.g., in Craft World, tasks are incremental in nature and consequently, the significant advantage of RePReL w/ T over RePReL and other baselines in Tasks 2, 3 and 4 can be clearly observed. In Office World, the *Task 4* is independent of *Task 1, 2,* and *3*, and thus, the gain due to transfer is not significant. Yet, RePReL w/ T converges faster than trl w/ T due to the state abstractions. This allows us to answer **Q2** affirmatively.

The significant advantage of RePReL over trl in Craft World when compared to the Office World, is due to the fact that Craft World has more objects (11 vs 9). We hypothesize and verify next that the advantage of state abstraction is more apparent in the relational domains where the number of objects is higher and effective transfer requires generalization across objects.

[Q3] Generalization: We present, in Fig. 5, the comparison of the RePReL with baselines in the **Extended Taxi World**. Here, both trl and hrl use 4 options for each location R, G, B, and Y, while RePReL uses **only two options**, one for each operator: pickup and drop a passenger. The results clearly show that RePReL consistently outperforms both the baselines. RePReL with transfer can perform *Task 2* and *3* seamlessly without any additional learning. This demonstrates the generalization capability of the RePReL agent across different passengers.

Our experiments in **Relational Box World** domain are presented in Fig. 6. Here, we use two subtask policies for both the trl and RePReL: one for opening lock and another for collecting key or gem. Since the locations of the lock and key are not fixed, we cannot use different options for each location in taskable RL. Each learning agent is provided a budget of $1.5M$ training steps in each task. We see that RePReL is significantly more efficient than trl, in all the three tasks. Here, *Task 2* involves opening one box (i.e. collecting key and opening lock) to reach the gem and *Task 3* requires opening two boxes. It can be clearly observed that RePReL w/ T is able to generalize across number of objects when going from *Task 2* to *Task 3*. These transfer results in Extended Taxi World and Relational Box World allows us to answer **Q3** affirmatively in that RePReL allows for generalizing across varying number of objects and is best suited for relational domains.

We also employed two relational RL baselines: Q-tree (Džeroski, De Raedt, and Driessens 2001) and Gradient Boosted Q-Learning (Das et al. 2020). However, with the number of time-steps that we used for the other planner-based methods, these RRL methods could not converge to an optimal policy. We hypothesize that this is due to the fact that they learn on fixed goal domains while our domains have varying goals. Investigating the extension of RRL to goal-directed methods is an interesting future direction.

Discussion and Future Work

It is important to note that taskable RL learns a different policy for each goal location R, G, B, Y i.e. $\pi_R(s), \pi_G(s), \pi_B(s), \pi_Y(s)$, while RePReL learns policy for pickup and drop operators i.e. $\pi_{pickup}(s), \pi_{drop}(s)$. These policies are equivalent (i.e., the action selected by trl would be same as RePReL), the difference lies in the way state s is represented. Taskable RL uses the complete state representation while RePReL generates abstract state representations. Another significant difference in RePReL learning algorithm proposed here is that the Taskable RL updates all the subtask policies for every step in the environment, while RePReL only updates the active subtask policy for each step. With these observations, our empirical results should make stronger cases for sample efficiency and generalization abilities for RePReL.

While our empirical evaluation of RePReL is quite successful, there are several interesting future avenues to pursue. First is combining the planner with Deep RL frameworks to handle hybrid (discrete-continuous) domains. More expressive propositional and relational representations can be explored at the RL level. Another direction is to allow for rich human interaction during learning (in a preference elicitation manner). Finally, scaling up the evaluation to more real-world domains is an interesting future direction.

Acknowledgements

HK & SN gratefully acknowledge the support of ARO award W911NF2010224. SN acknowledges AFOSR award FA9550-18-1-0462. PT acknowledges support of DARPA contract N66001-17-2-4030 and NSF grant IIS-1619433. AM gratefully acknowledge the travel grant from RBCD-SAI. Any opinions, findings, conclusion or recommendations expressed in this material are those of the authors and do not necessarily reflect the view of the ARO, AFOSR, NSF, DARPA or the US government. We sincerely thank Illanes et al. (2020) for sharing the Taskable RL code for baselines.

References

Abel, D.; Arumugam, D.; Lehnert, L.; and Littman, M. 2018. State abstractions for lifelong reinforcement learning. In *ICML*, volume 80, 10–19.

Andre, D.; and Russell, S. J. 2002. State Abstraction for Programmable Reinforcement Learning Agents. In *AAAI*, 119–125.

Andreas, J.; Klein, D.; and Levine, S. 2017. Modular multitask reinforcement learning with policy sketches. In *ICML*, volume 70, 166–175.

Brafman, R. I.; and Tennenholtz, M. 2002. R-max-a general polynomial time algorithm for near-optimal reinforcement learning. *JMLR* 3: 213–231.

Das, S.; Natarajan, S.; Roy, K.; Parr, R.; and Kersting, K. 2020. Fitted Q-Learning for Relational Domains. *CoRR* abs/2006.05595.

Dietterich, T. G. 2000. State abstraction in MAXQ hierarchical reinforcement learning. In *NeurIPS*, 994–1000.

Džeroski, S.; De Raedt, L.; and Driessens, K. 2001. Relational reinforcement learning. *Machine learning* 43(1/2): 7–52.

Eppe, M.; Nguyen, P. D. H.; and Wermter, S. 2019. From Semantics to Execution: Integrating Action Planning With Reinforcement Learning for Robotic Causal Problem-Solving. *Frontiers in Robotics and AI* 6: 123.

Fern, A.; Yoon, S.; and Givan, R. 2006. Approximate policy iteration with a policy language bias: Solving relational Markov decision processes. *JAIR* 25: 75–118.

Finzi, A.; and Lukasiewicz, T. 2006. Adaptive Multi-agent Programming in GTGolog. In *KI*, volume 4314, 389–403.

Georgievski, I.; and Aiello, M. 2015. HTN planning: Overview, comparison, and beyond. *Artificial Intelligence* 222: 124–156.

Getoor, L.; and Taskar, B. 2007. *Introduction to Statistical Relational Learning*. The MIT Press.

Givan, R.; Dean, T.; and Greig, M. 2003. Equivalence notions and model minimization in Markov decision processes. *Artificial Intelligence* 147(1-2): 163–223.

Grounds, M.; and Kudenko, D. 2005. Combining reinforcement learning with symbolic planning. In *AAMAS III*, volume 4865, 75–86.

Guestrin, C.; Koller, D.; Gearhart, C.; and Kanodia, N. 2003. Generalizing plans to new environments in relational MDPs. In *IJCAI*, 1003–1010.

Guestrin, C.; Patrascu, R.; and Schuurmans, D. 2002. Algorithm-directed exploration for model-based reinforcement learning in factored MDPs. In *ICML*, 235–242.

Illanes, L.; Yan, X.; Icarte, R. T.; and McIlraith, S. A. 2020. Symbolic Plans as High-Level Instructions for Reinforcement Learning. *ICAPS* 540–550.

Jiang, Y.; Yang, F.; Zhang, S.; and Stone, P. 2019. Task-Motion Planning with Reinforcement Learning for Adaptable Mobile Service Robots. In *IROS*, 7529–7534.

Konidaris, G. 2019. On the necessity of abstraction. *Current Opinion in Behavioral Sciences* 29: 1–7.

Kulkarni, T. D.; Narasimhan, K.; Saeedi, A.; and Tenenbaum, J. 2016. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In *NeurIPS*, 3675–3683.

Lake, B. M.; Salakhutdinov, R.; and Tenenbaum, J. B. 2015. Human-level concept learning through probabilistic program induction. *Science* 350(6266): 1332–1338.

Li, L.; Walsh, T. J.; and Littman, M. L. 2006. Towards a Unified Theory of State Abstraction for MDPs. In *ISAIM*, volume 4, 5.

Lyu, D.; Yang, F.; Liu, B.; and Gustafson, S. 2019. SDRL: Interpretable and data-efficient deep reinforcement learning leveraging symbolic planning. In *AAAI*, 2970–2977.

Natarajan, S.; Tadepalli, P.; Dietterich, T. G.; and Fern, A. 2008. Learning first-order probabilistic models with combining rules. *Ann. Math. Artif. Intell.* 54(1-3): 223–256.

Nau, D.; Cao, Y.; Lotem, A.; and Munoz-Avila, H. 1999. SHOP: Simple hierarchical ordered planner. In *IJCAI*, 968–975.

Price, B.; and Boutilier, C. 2001. Imitation and reinforcement learning in agents with heterogeneous actions. In *Conference of the Canadian Society for Computational Studies of Intelligence*, volume 2056, 111–120.

Raedt, L. D.; Kersting, K.; Natarajan, S.; and Poole, D. 2016. Statistical relational artificial intelligence: Logic, probability, and computation. *Synthesis Lectures on Artificial Intelligence and Machine Learning* 10(2): 1–189.

Ravindran, B.; and Barto, A. G. 2003. SMDP Homomorphisms: An Algebraic Approach to Abstraction in Semi Markov Decision Processes. In *IJCAI*, 1011–1018.

Sanner, S.; and Boutilier, C. 2009. Practical solution techniques for first-order MDPs. *Artificial Intelligence* 173(5-6): 748–788.

Silver, D.; Hubert, T.; et al. 2018. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science* 362(6419): 1140–1144.

Sorg, J.; and Singh, S. 2009. Transfer via soft homomorphisms. In *AAMAS*, 741–748.

Sutton, R. S.; Precup, D.; and Singh, S. P. 1998. Intra-Option Learning about Temporally Abstract Actions. In *ICML*, 556–564.

Tadepalli, P.; , R.; and Driessens, K. 2004. Relational reinforcement learning: An overview. In *ICML workshop on relational reinforcement learning*, 1–9.

Walsh, T. J.; Li, L.; and Littman, M. L. 2006. Transferring state abstractions between MDPs. In *ICML Workshop on Structural Knowledge Transfer for Machine Learning*.

Yang, F.; Lyu, D.; Liu, B.; and Gustafson, S. 2018. PEORL: Integrating symbolic planning and hierarchical reinforcement learning for robust decision-making. *IJCAI* 4860–4866.

Zambaldi, V.; Raposo, D.; Santoro, A.; Bapst, V.; Li, Y.; Babuschkin, I.; Tuyls, K.; Reichert, D.; Lillicrap, T.; Lockhart, E.; Shanahan, M.; Langston, V.; Pascanu, R.; Botvinick, M.; Vinyals, O.; and Battaglia, P. 2019. Deep reinforcement learning with relational inductive biases. In *ICLR*.