RePReL: A Unified Framework for Integrating Relational Planning and Reinforcement Learning for Effective Abstraction in Discrete and Continuous Domains

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

[1]The University of Texas at Dallas, USA.
[2]Robert Bosch Centre for Data Science and Artificial Intelligence, Indian Institute of Technology Madras, India.
[3] Oregon State University, USA.

*Corresponding author(s). E-mail(s): hkokel@utdallas.edu;
Contributing authors: Sriraam.Natarajan@utdallas.edu; ravi@cse.iitm.ac.in; tadepall@eecs.oregonstate.edu;

Abstract

We propose a hybrid planner-(deep)reinforcement learning (RL) architecture, RePReL, that leverages a relational planner to efficiently provide useful state abstractions. State abstractions have a tremendous advantage for better generalization and transfer in RL. Our framework takes an important step toward constructing these abstractions. Specifically, the framework enables multi-level abstractions by leveraging a high-level planner to communicate with a low-level (deep) reinforcement learner. Our empirical results demonstrate the generalization and transfer capabilities of the framework in both discrete and continuous domains with rich structures (objects and relations between these objects). A key aspect of RePReL is that it can be seen as a plug-and-play framework where different planners can be used in combination with different (deep) RL agents.

Keywords: Planning, Reinforcement Learning, Abstractions

1 Introduction

Planning and Reinforcement Learning (RL) have been two major thrusts of AI aimed at sequential decision-making. While classical relational planning focuses on composing sequences of actions offline before any execution, RL interleaves planning and execution and is typically associated with reactive domains with unknown dynamics. We describe an integrated architecture that we call "RePReL," which combines *relational planning* (RePP) and *reinforcement learning* (ReL) in a way exploiting their complementary strengths, accelerating the convergence compared to a traditional RL solution, and enabling effective transfer 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 [4, 16] or online in look-ahead search [43]. The main motivation for the combination comes from the benefits of efficiency and cost savings due to offline computation or look-ahead search. Critically, both planning and RL components employ the same state space.

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 enabling 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 taking "Exit 205," while the low-level state space consists of a more precise location and velocity of the car and actions such as turning the steering wheel or braking. 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.

In this work, we aim to combine planning and RL with the motivation of exploiting the hierarchy in the domain and inducing effective task-specific abstractions for efficient learning. **An important assumption that we make is the availability of a relational planner designed by a human expert**. If the planner is a hierarchical one, the human expert is used to construct the task-subtask hierarchy, otherwise the human expert's guidance is used to define the sequence of high-level tasks for solving the problem. In addition, as we explain later, knowledge from humans is employed to provide safe and effective abstractions.

Our key contribution is the RePReL architecture that enables multi-level abstractions. The second contribution is the adaptation of first-order conditional influence (FOCI) statements [35, 36] to determine safe and effective abstractions. Finally, we demonstrate the effective transfer of learned skills from one task to another.

This work builds on our prior work [27] and extends it significantly in a few different directions. First, the prior work only handled tabular Q-learning in discrete domains. We have *extended the formalism and presented results on using neural RL to handle both discrete and continuous states and actions.* To this effect, we employ Double Deep Q-Network [19] and Soft-Actor Critic (SAC) [18] as base learners. Another key difference is that while our earlier work generated the ground state abstractions by unrolling the D-FOCI to a fixed depth (which limits its use), we *allow for richer recursion and relational state representations by employing graph neural networks.* The final difference is that while the prior work can handle only online RL updates, *our current approach allows for both online and batch RL updates.*

The rest of the paper is organized as follows. Section 2 summarizes the required background and the relevant related work in planning and RL. Section 3 describes the RePReL architecture in detail. Section 4 describes empirical evaluations on four different domains that demonstrate generalization and transfer across multiple tasks. Finally, Section 5 concludes and discusses future directions.

2 Background and Related Work

This section introduces some basic first-order logic formalisms and notations. Then it briefly covers the required background in the areas of RL and planning, including hierarchical approaches. Then it presents a brief summary of the related work at the intersection of RL and planning.

2.1 First-order Logic

A *first-order language* $\mathcal{L}$ consists of a finite set of constants, predicates, and free variables. Each *predicate* takes a fixed number of arguments—arity α—and is represented as `predicate/`α. Following Prolog notation, we represent free variables in uppercase and constants in lowercase. A substitution θ maps variables to constants. An *atom* is a predicate symbol followed by a parenthesized list of terms, `predicate(term`$_1$`, term`$_2$`, ···)`. A *literal* is an atom or negation of an atom. If all the terms in an atom are constants it is called a grounded atom, otherwise, it is called a lifted atom. A lifted atom can be grounded by substitution. For example, substitution $\theta = \{X/\texttt{p1}\}$ grounds atom $a = \texttt{taxi\_at}(X)$ to $a\theta = \texttt{taxi\_at}(\texttt{p1})$

2.2 Reinforcement Learning

A Markov decision process (MDP) is defined as a tuple $\langle S, A, T, R, \gamma\rangle$, with a set of states S, a set of actions A, a transition function $T : S \times A \times S \rightarrow [0, 1]$, a reward function $R : S \times A \times S \rightarrow \mathbb{R}$, and a discount factor γ. The goal of an agent is to interact with the environment and find an optimal policy $\pi : S \times A \rightarrow [0, 1]$ maximizing the expected cumulative discounted reward.

While RL is generally successful, it suffers from a curse of dimensionality especially when state features are very large. Hierarchical RL addresses it by introducing temporal abstractions. HRL methods allow for improved exploration in problems with rich structure, are efficient in learning due to their ability to decompose larger tasks into smaller sub-tasks, and are amenable to transfer and generalization [6]. While MDPs conceive time as a discrete step, where a chosen action only persists for a single step, Semi-MDPs (SMDPs) [38, 45] introduce temporal abstractions, where a chosen action persists over a variable period of time. The options framework [44] is one such hierarchical RL approach that models temporally extended actions as *options*. An option $p = \langle I_p, \pi_p, \beta_p\rangle$ consists of three components: a set of states where the option can be initiated I_p, an option policy π_p, and a termination condition $\beta_p : S \rightarrow [0, 1]$ that describes the probability of option termination.

For relational domains, the MDP definition has been extended to the relational MDP [12]. We focus on the relational MDPs, but before introducing them we must make a distinction between the state predicates and the action predicates. State predicates indicate properties or relations between the entities of the world, whereas action predicates indicate an activity, movement, or step. For example, in the taxi domain predicate `east/0` is an action predicate and the atom `east()` moves the taxi in the east direction. For simplicity, we assume the action predicates have zero arity.

Definition 1 Given a first-order language $\mathcal{L}$ consisting of a set of constants C, a set of state predicates P, and a set of action predicates Y, a **relational MDP** (RMDP) is defined as a tuple $\langle S, A, T, R, \gamma\rangle$, where:

- S is a finite set of states defined over all the ground atoms generated by constants C and state predicates P
- A is a finite set of actions defined as a set of ground atoms generated by constants C and action predicates Y
- $T : S \times A \times S \rightarrow [0, 1]$ is a probabilistic transition function
- $R : S \times A \times S \rightarrow \mathbb{R}$ is a reward function
- $\gamma \in [0, 1)$ is the discount factor

2.3 Planning

A *schematic operator* in classical planning [13] is defined as $c = \langle h(c), \mathit{pre}(c), \mathit{eff}^+(c), \mathit{eff}^-(c)\rangle$, consisting of a lifted atom, $h(c)$, often referred to as the *head*; a first-order formula called preconditions $\mathit{pre}(c)$; and two disjoint sets of atoms, $\mathit{eff}^+(c)$ and $\mathit{eff}^-(c)$, describing the positive (add) and negative (delete) effects of executing the operator. All terms that appear in the literals in pre, eff^+, and eff^- also appear as arguments of $h(c)$. Hence, a schematic operator can be grounded by substituting the head atom, $h(c)\theta$. A ground operator $c\theta$ can be applied in a state s if a substitution θ satisfies the precondition $\mathit{pre}(c)$ at s, i.e. $s \models \mathit{pre}(c)\theta$. On applying the ground operator $c\theta$, s transitions to another state $s' = (s \setminus \mathit{eff}^-(c)\theta) \cup \mathit{eff}^+(c)\theta$.

A planning domain $\mathbf{D} = \langle \mathcal{L}, \mathbf{O}\rangle$ consists of a first-order language $\mathcal{L}$ and a finite set of schematic operators $\mathbf{O}$. A classical planning task is defined as a tuple $\Pi = \langle \mathbf{D}, s_0, g\rangle$, where s_0 is an initial state and g is a goal specified as a conjunction of literals. A plan for the planning task is a sequence of ground operators, which when executed in a state s results in a state satisfying g.

A hierarchical planning domain $\mathbf{D} = \langle \mathcal{L}, \mathbf{O}, \mathbf{M}\rangle$ is similar to a classical planning domain with an additional set of methods $\mathbf{M}$. A method is a triple $m = \langle t(m), \mathit{pre}(m), \tau(m)\rangle$, where t is a compound-task, pre is a precondition for the method, and τ is an ordered sequence of compound-tasks or operators. A task $t(m)$ can be decomposed into a sequence of sub-tasks $\tau(m)$ at a state s if a substitution θ satisfies the preconditions in s, i.e. $s \models \mathit{pre}(m)\theta$. A hierarchical planner solves the planning task by recursively decomposing the goal into a sequence of sub-tasks.

2.4 Related Work

Prior research has explored the idea of combining AI planning and RL agents to solve complex problems involving temporally extended actions or task hierarchies [10, 15, 23, 33, 47]. Among these, our RePReL is closely related to the Taskable RL framework of Illanes et al [21]. Similar to Taskable RL, RePReL employs a planner to generate useful instructions (temporally extended operators) for the RL agent. RePReL extends the Taskable RL framework in three key ways: (1) it generalizes the Taskable RL to solving RMDP, (2) it provides an approach to define task-specific state abstraction in this framework, and (3) it can handle both discrete and continuous domains. Thus, Taskable RL is a natural baseline in our evaluations.

RMDPs can either be converted to propositional MDPs and solved using standard RL approaches, or they can be solved using relational RL (RRL) approaches. RRL approaches include symbolic as well as neural approaches. Notable work using symbolic Relational RL (RRL) methods include relational TILDE trees with Q-learning [9], RRL-TG that replaces TILDE with incremental tree learning algorithm [8], approximate policy iteration (API) with decision-lists [12], relational gradient-boosted Q-learning (GBQL) and relational boosted fitted Q-learning (RBFQ) [5]. Neural RRL approaches include neural logic machines (NLM) [7] that showed the ability to solve blocks world with up to 50 blocks, neural logic reinforcement learner (NLRL) [24], off-policy differentiable logic RL (OPDLRL) [49] that uses differentiable ILP (∂ILP) [11], the work by Kimura et al [26] that used Logical Neural Network (LNN) [42] for text-based RL games, and various deep relational RL works that use graph-based neural networks (GNNs) architectures with off-policy learning algorithms [22, 31, 48]. In our experiments, we used a symbolic RRL, a propositional deep RL as well as a GNN-based relational deep RL agent for evaluations.

3 Integration of relational planning and reinforcement learning

This section explains the problem setup and presents the RePReL architecture.

3.1 Problem Setup

To address a multi-task setting, we first restrict the RMDP definition to a goal-directed RMDP (GRMDP), $\mathcal{M} = \langle S, A, T, R, \gamma, G\rangle$, by introducing a set of goals G that the agent may be asked to achieve. Different tasks are formulated in a GRMDP by choosing different goals from the set G. The reward function in this multi-task setting is incognizant of the goal. It provides a domain-specific reward, like a negative reward for invalid actions or step costs. The solution to a GRMDP is a policy $\pi : S \times G \times A \to [0, 1]$, such that when following π from any state s for goal g the probability of eventually reaching the goal is 1.

We assume that partial high-level domain knowledge of the problem is specified in the form of a high-level planner. We now briefly explain the notion of high-level planning. Unlike the classical planning domains $\mathbf{D} = \langle \mathcal{L}, \mathbf{O}\rangle$ where a schematic operator is a single action, in the high-level planning domain $\mathcal{D} = \langle \mathcal{L}, \mathcal{O}\rangle$ schematic operators are temporally extended. This is to say that, in classical planning, applying a grounded operator to state results in a single step or transition from one state to another. In high-level planning, applying a grounded operator to a state usually requires multiple steps or multiple subsequent state transitions before it terminates. The termination condition β is also defined for each schematic operator. Given the resemblance to the options framework [44], we refer to these temporally extended operators as *options*. A high-level plan consists of a sequence of grounded options.

We address a class of GRMDPs that combines a high-level symbolic planner with low-level RL policies. Inspired by an earlier work [21], we define this class of GRMDPs as taskable GRMDPs.

Definition 2 A GRMDP $\langle S, A, T, R, \gamma, G\rangle$ is **taskable** if a high-level planning domain $\mathcal{D}$ can be defined such that all the goals in G can be composed as some combination of the options in $\mathcal{D}$.

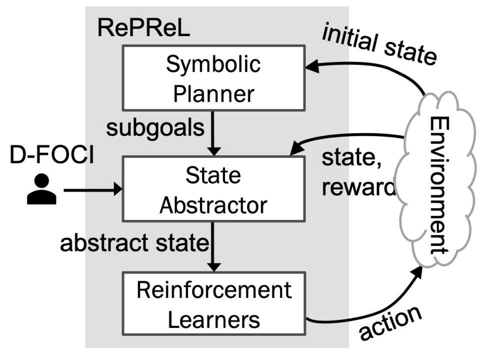


Fig. 1: RePReL architecture.

The key idea here is that high-level symbolic domain knowledge can be leveraged to identify the compositionality in the domain and individual policies can be learned to solve each composition. *So a solution to a taskable GRMDP is a compositional policy.*

3.2 RePReL

Given a taskable GRMDP problem, the RePReL framework obtains the high-level plan from a symbolic planner and then learns RL policies at the low level to achieve each of the options in the high-level plan. The architecture of the RePReL framework is shown in Figure 1. It consists of three stacked modules: *symbolic planner*, *state abstractor*, and *reinforcement learners.*

The *symbolic planner* uses the high-level planning domain $\mathcal{D}$ to decompose the goal into a sequence (or set, if unordered) of grounded options. It must be mentioned that our RePReL framework is **independent of the planner**. If the planner is a hierarchical one, we assume that high-level tasks (methods) are decomposed recursively until low-level options are constructed. If the planner is a non-hierarchical one, the high-level decomposition yields a set of options. The *state abstractor* generates a task-specific abstract state representation. Finally, multiple *reinforcement learners* at the lowest level learn separate RL policies for each option in the abstract state space. For each option $o \in \mathcal{O}$ in the high-level planning domain, we define a subgoal RMDP as follows.

Definition 3 The subgoal RMDP M_o for an option o is a tuple $\langle S, A, P_o, R_o, \gamma\rangle$ consisting of abstract states S, actions A, transition function P_o, reward function R_o, discount factor γ. The action space remains the same as for the original GRMDP. The reward function R_o and transition probability distribution function P_o are defined as follows:

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

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

with R and P from the original GRMDP, and a fixed terminal reward t_R.

This subgoal RMDP is solved by the low-level RL agents in abstract state space. We next describe how the state space is abstracted safely by the state abstractor.

3.2.1 State Abstractor

The state abstractor determines the set of state variables that are necessary and sufficient given the current task and provides a task-specific abstract representation of the state for the low-level RL agents. We adopt the bisimulation framework of Givan et al [14] and Ravindran and Barto [40] to define model agnostic abstraction.

Definition 4 A **model-agnostic abstraction** $\phi(s)$ is such that for any action a and an 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 states that the two states s_1 and s_2 have the same immediate reward distribution with respect to the abstraction, and the second condition states that they have the same transition dynamics. Essentially, the value function of the original MDP is maintained in the abstract MDP. So, the model agnostic abstraction is *safe*. That is, an optimal policy learned with model-agnostic abstraction is also optimal in the original MDP [30].

To define such a safe model agnostic abstraction function ϕ, we need to identify state literals that neither influence the reward function nor the transition function—they are irrelevant. In this framework, we capitalize on explicit domain knowledge to identify the relevant state literals from irrelevant ones. This domain knowledge is captured using a formal language called D-FOCI.

D-FOCIs

First-order conditional influence statements (FOCI), introduced by Natarajan et al [35] consist of statements of the form,

$$\begin{aligned}&\text{IF } \texttt{<condition>}\\ &\text{THEN } \texttt{<influent>} \text{ QINF } \texttt{<resultant>},\end{aligned} \tag{1}$$

where `condition` and `influent` are a finite set of literals and `resultant` is a single literal. It encodes the qualitative influence (QINF) of `influent` on the `resultant`.

Dynamic-FOCI (D-FOCI) *statements* extend FOCI statements by adding representation to capture influence over time. The syntax of a D-FOCI statement is as follows,

$$[\texttt{<option>}] : \texttt{<influent>} \xrightarrow{[+1]} \texttt{<resultant>}, \tag{2}$$

where the `option` is a temporally extended operator from the high-level planning domain, and `influent` and `resultant` are set of literals.

D-FOCI statement states that when executing the given `option`, the `resultant` literal is influenced by literals in `influent`. Following the standard dynamic Bayesian network (DBN) representation of an MDP, we allow action variables in `influent` and reward variables in `resultant`. D-FOCIs denote direct influences between literals in the same time step by an arrow symbol ($\longrightarrow$) and direct influences of the literals in the current time step on the literals in the next time step by a '+1' symbol above the arrow. Non-mandatory components of the D-FOCI statement are denoted within square brackets [ ]. Omitting the `option` encodes the influences between literals is perpetual (for example, Equation 3a encodes that the `action` and `taxi_at` always influence `taxi_at`, regardless of the option).

D-FOCI statements are a class of dynamic relational models [17, 34, 46]. Similar to dynamic relational models, D-FOCIs captures the conditional independence relationships between domain predicates at different time steps [28] and has the additional capability of conditioning on the option being executed.

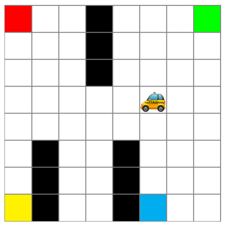


Fig. 2: Taxi domain with multiple passengers and a single taxi.

Example of D-FOCI: Consider a relational taxi domain shown in Figure 2. This domain has one or more passengers and a taxi. Six actions available in this domain are: `east`, `west`, `north`, `south`, `pick`, `drop`. The task is to transport passenger(s) from their current location to their destination location. Only 1 passenger can hire the taxi at a time.

The location of the taxi is influenced by its previous location and the action performed, which is captured in the D-FOCI statement in Equation 3a[1]. Further, when executing the task of picking up a passenger, if we assume that the taxi is going to be empty, then we can safely say that only the passenger's location, the passenger in the taxi, and the taxi's location influence the completion of the task. This influence is captured in Equations 3b and 3c. Similarly, the influence information while dropping passenger P is captured in Equation 3d–3f.

$$\begin{aligned}
&\{\texttt{action},\texttt{taxi\_at}(X)\} \xrightarrow{+1} \texttt{taxi\_at}(X) && \text{(3a)}\\
&\texttt{pick}(P) : \{\texttt{action}, \texttt{taxi\_at}(X), \texttt{at}(P, Y),\\
&\qquad \texttt{in\_taxi}(P)\} \xrightarrow{+1} \texttt{in\_taxi}(P) && \text{(3b)}\\
&\texttt{pick}(P) : \{\texttt{in\_taxi}(P)\} \longrightarrow \mathit{Reward} && \text{(3c)}\\
&\texttt{drop}(P) : \{\texttt{at\_dest}(P)\} \longrightarrow \mathit{Reward} && \text{(3d)}\\
&\texttt{drop}(P) : \{\texttt{at}(P, X), \texttt{dest}(P, D), \texttt{at\_dest}(P)\}\\
&\qquad \longrightarrow \texttt{at\_dest}(P) && \text{(3e)}\\
&\texttt{drop}(P) : \{\texttt{action}, \texttt{taxi\_at}(X), \texttt{at}(P, Y),\\
&\qquad \texttt{in\_taxi}(P)\} \xrightarrow{+1} \texttt{at}(P, K) && \text{(3f)}
\end{aligned}$$

[1]Variables are uppercase. Constants and predicates are lowercase. X, Y, D, K are variables for location and P a variable for passenger.

Abstraction using D-FOCI

The state abstractor in the RePReL architecture (Figure 1) uses D-FOCIs to generate model-agnostic state abstractions. A task-specific model-agnostic abstract representation of the state can be derived from D-FOCI by recursively unrolling and collecting the state literals that influence the relevant state literals, starting from the reward variables. Table 1 illustrates this recursive unrolling process.

Given a state s and a grounded option, the unrolling process begins by grounding the relevant D-FOCI statements[2] that have the *Reward* variable on the right-hand side (RHS). To ground a D-FOCI statement, a substitution θ is identified that unifies the literals on the left-hand side (LHS) with state s. Then the literals in the LHS are collected in a *relevant literals set* $\hat{s}$. Then the relevant D-FOCI statements that have RHS in $\hat{s}$ are grounded and substitution θ is refined and literals on LHS are added to $\hat{s}$. Note that the substitution is a superset of the previous substitution, that is the existing variable assignment remains the same. This process is repeated recursively. The recursion ends when no new literals can be added to $\hat{s}$.

An important difference to the prior RePReL framework [27] must be noted here. The prior work assumed D-FOCI statements were unrolled for fixed depth to obtain propositional abstract state representation with fixed vector size for RL agents. *The current work allows relational state representation for RL agents, that is, graph neural network-based RL agents with the capability of processing relational states are used.* By using the relational agents at RL level, the framework supports state representations of varying sizes and, hence, the framework allows varying the unrolling depth.

Theorem 1 *If the MDP satisfies the influence information of the D-FOCI statements then the above procedure that recursively collects the set of state literals that influence the relevant state literals and reward variables is a model agnostic abstraction.*

Proof (sketch): Complete grounding of the D-FOCI statements would create a propositional DBN with all the grounded literals as variables.

Given: a. D-FOCI statements from Equation 3 b. state $s = \{$ at(p1,r), taxi_at(l3), dest(p1,d1), $\neg$at_dest(p1) $\neg$in_taxi(p1), at(p2,b), $\neg$at_dest(p2), $\neg$in_taxi(p2)$\}$ c. grounded option$o\theta$: pick(P) $\{P$/p1$\}$ **Output:** A set of relevant state literals: $\hat{s}$
Depth 1 unrolling: 1. Find a substitution that grounds relevant D-FOCI statements that have reward on RHS pick(p1): in_taxi(p1) $\longrightarrow$ *Reward* $\theta = \{$P/p1$\}$ 2. Collect LHS in relevant literals set $\hat{s}$ $\hat{s} \leftarrow \{$in_taxi(p1)$\}$
Depth 2 unrolling: 1. Find a substitution that grounds relevant D-FOCI statements that have a relevant literal on RHS pick(P): { action, taxi_at(l3), at(p1, r), in_taxi(p1) } $\longrightarrow$ in_taxi(p1) $\theta = \{$P/p1, X/l3, Y/r$\}$ 2. Collect LHS in set $\hat{s}$ $\hat{s} \leftarrow \{$in_taxi(p1), action, taxi_at(l3), at(p1, r)$\}$
Depth 3 unrolling: 1. Ground applicable D-FOCI statements that have a relevant literal ($\hat{s}$) on RHS {action, taxi_at(l3) } $\xrightarrow{+1}$ taxi_at(l3) pick(p1): { action, taxi_at(l3), at(p1, r), in_taxi(p1) } $\longrightarrow$ in_taxi(p1) $\theta = \{$P/p1, X/l3, Y/r$\}$ 2. Collect LHS in set $\hat{s}$ $\hat{s} \leftarrow \{$in_taxi(p1), action, taxi_at(l3), at(p1, r)$\}$

Table 1: Illustrative example of recursive unrolling of the D-FOCI statements in taxi-domain.

If the MDP satisfies the influence information in this DBN then collecting all the variables influencing the reward and the relevant variables provides model agnostic abstraction [41]. The recursive grounding and unrolling of the D-FOCI statement begins with the reward variable and collects all the grounded literals influencing the reward and the relevant grounded literals. So the set of grounded state literals collected by this process is identical to collecting all the relevant state variables in the propositional DBN. Hence, the recursive grounding and unrolling provide a model-agnostic abstraction.□

3.2.2 Summary

In this problem setup, it is assumed that a high-level symbolic planner is available that can decompose the goal into a high-level plan, i.e. sequence of grounded option, as described in Section 3.1.

[2]with matching or `null` option

In RePReL architecture, the initial state of the world is provided as input to a symbolic planner and a high-level plan is obtained as its output. This high-level plan is executed by different RL agents at the lower level. A separate RL agent is trained for each option. The MDP of each RL agent is defined in Definition 3. For each RL agent, the state abstractor provides a task-specific abstraction, as described in Section 3.2.1. The state abstractor uses the D-FOCI statements to derive model-agnostic abstraction. The next section explains how the RePReL architecture is learned.

Algorithm 1 RePReL Algorithm

```
INPUT: env, goal $g$, domain $\mathcal{D} = \langle \mathcal{L}, \mathcal{O} \rangle$, terminal reward $t_R$, D-FOCI statements $F$, num of iterations $i$, num of episodes in each iteration $k$, batch size $b$
OUTPUT: RL policies $\pi_o, \forall o \in \mathcal{O}$
1:  $\pi_o, \mathcal{B}_o, \forall o \in \mathcal{O}$
        ▷ initialize a policy and a buffer for each option
2:  for iteration $\in i$ do
3:      for episode $\in k$ do
4:          $s \leftarrow$ get state from env
5:          $\Pi \leftarrow$ getPlan($s, g, \mathcal{D}$)
6:          for $o\theta$ in $\Pi$ do
7:              $\pi \leftarrow \pi_o$                          ▷ get resp. policy
8:              $\hat{s} \leftarrow$ GetAbstractState($s, o\theta, F$)
9:              done $\leftarrow \hat{s} \in \beta(o\theta)$
                                              ▷ check terminal state
10:             while not done do
11:                 $a \leftarrow \pi(\hat{s})$                 ▷ get action
12:                 $s' \leftarrow$ env.step($a$) ▷ take a step in env
13:                 $r \leftarrow R(s, a, s')$          ▷ get step reward
14:                 $\hat{s}' \leftarrow$ GetAbstractState($s, o\theta, F$)
15:                 done $\leftarrow \hat{s}' \in \beta(o\theta)$
                                         ▷ check terminal next state
16:                 if done then
17:                     $r = r + t_R$ ▷ add terminal reward
18:                 end if
19:                 $\mathcal{B}_o \leftarrow \mathcal{B}_o \cup \{\hat{s}, a, r, \hat{s}', o\theta\}$
                                               ▷ push to the buffer
20:                 $s, \hat{s} \leftarrow s', \hat{s}'$
21:             end while
22:         end for
23:     end for
24:     for $o \in \mathcal{O}$ do             ▷ Update all the policies
25:         $\pi_o \leftarrow$ UpdatePolicy($\pi_o$, SampleBatch($\mathcal{B}_o, b$))
26:     end for
27: end for
28: return $\pi_o, \forall o \in \mathcal{O}$
```

3.2.3 Learning

Given a taskable GRMDP environment `env` with the high-level planning domain $\mathcal{D}$ and the D-FOCI statements F, we now discuss the RePReL learning procedure from **Algorithm 1**. First, for each option, an RL policy π_o and a replay buffer $\mathcal{B}_O$ are initialized in **line 1**. Next, for the current instance of the GRMDP problem, a high-level plan Π is obtained from the planner in **line 5**. For every grounded option in the plan, training samples are collected in the respective buffer $\mathcal{B}_o$ (**lines 6–22**). An abstract state representation $\hat{s}$ is obtained in **line 8** and if it is not a terminal state then samples are collected in the buffer till a terminal state is reached (**lines 10–21**). To collect samples, an action a is obtained from the current policy, that action is performed, and the next state s' and the reward r are observed. If s' is a terminal state for the ground option $o\theta$, then a terminal reward t_R is added (**line 17**) before pushing it to the buffer (**line 19**). Once enough samples are collected in the buffer, the option policy π_o is updated for each option by sampling a batch from buffer $\mathcal{B}_o$ (**lines 25–28**). The process is repeated for a fixed budget of episodes during evaluation but various other stopping criteria can also be used.

3.3 Discussion

Neural or deep RL approaches are shown to be proficient in approximating the value function as well as the policy function in various games and robotic domains. While in our previous work [27] we proposed an online learning algorithm for RePReL that used tabular Q-learning as the base learner, in this work we present how RePReL agents can be learned in a batch setting. This transition from online to batch learning is crucial to facilitate the use of deep RL agents as the base learners. With batch setting, the RePReL framework can support any off-policy RL algorithm. Additionally, we can adapt the batch RePReL learning algorithm to any of the sampling and replay strategies to speed up the training. In our experiments, we employ a hindsight experience replay buffer [1] with Soft Actor-Critic [18] learner, as well as a naive buffer with Double DQN [19] learner.

The low-level RL agents learn optimal policies for the subgoal RMDP. As the original GRMDP is decomposed into the subgoal RMDP by the high-level planner, the resulting RePReL agent is recursively optimal [6] assuming the high-level planner and the GRMDP decompositions are optimal. Note that recursively optimal policies are not necessarily globally optimal. Rather, recursively optimal policies ensure that lowest level policies are locally optimal for the assigned task without any context of the high-level task [6]. This makes them ideal for multi-task and transfer-learning setups.

Our framework takes an important step in constructing a task-specific abstract representation of the state. Learning such abstract state representation is an active area of research in the field of RL as well as automated planning. As our work is at the integration of RL and Planning, it is closest to Konidaris et al [29], wherein symbols or abstract state representation are learned which are suitable for evaluating the high-level plan. They learn an abstract representation for the high-level planner and assume the low-level skills and trajectories are provided. In our work, we are seeking task-specific abstract representation for efficient learning of low-level agents.

4 Experiments

We now empirically evaluate our approach on four multi-task domains with discrete as well as continuous state and action spaces. We compared the RePReL framework with traditional tabular RL, traditional symbolic RL, deep RL, hierarchical RL, deep relational RL, and planning+RL approaches. We consider *three different representations*: 1. *Predicate*, we first use predicate representation in RePReL with tabular Q-learning as the base learner. 2. *Tensor*, we use tensor representation to evaluate RePReL with double DQN as the base learner. 3. *Graph*, we use graph representation to evaluate RePReL with deep relational RL-based learners. The high-level planner remains the *same* for all these representations. This demonstrates the **generality** of our proposed framework.

All our experiments aim at assessing the sample efficiency and effectiveness of transfer across tasks and generalization across objects. We aggregate results over 5 runs with different random seeds. We employ the Pyhop planner[3] in our experiments. We aim to demonstrate the following key aspects of the proposed framework:

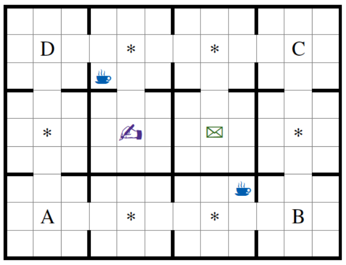


Fig. 3: Office World from Illanes et al [21]

A1. RePReL framework demonstrates a significant advantage in sample efficiency.

A2. RePReL framework facilitates effective transfer across tasks.

A3. RePReL framework efficiently generalizes across multiple objects.

A4. RePReL handles both discrete and continuous state-action spaces, making it a versatile framework for learning in real domains.

A5. RePReL batch learning algorithm adapts easily to different off-policy deep RL algorithms.

4.1 Tabular Reinforcement Learning

We considered three relational multi-task gridworld environments: **Office World**, **Extended Taxi World** and **Relational Box World**. *Office World* is a 9×12 grid (shown in Figure 3) with one office location (indicated with hand), two coffee locations (indicated by mugs), one mailroom (indicated by envelope), and some plants in the office to be avoided (indicated by $*$). The agent can perform 4 actions: *up, down, right,* and *left.* We consider 4 tasks: Task **1.** deliver mail to office; Task **2.** deliver coffee to office; Task **3.** deliver mail and coffee; and Task **4.** visit locations A, B, C, D.

Our second domain is *Extended Taxi World* shown in Figure 2. We extended the Taxi domain by Dietterich [6] in a few ways: the grid size is increased to 8×8; the number of passengers is increased to three; the initial location of the taxi is randomly sampled in each iteration. 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

[3]An HTN planner [37] written in python, https://bitbucket.org/dananau/pyhop

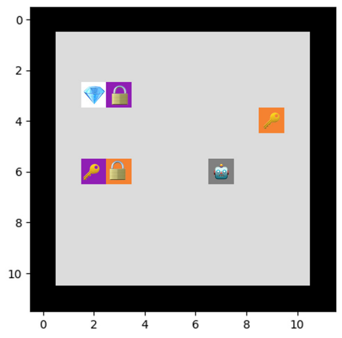


Fig. 4: Relational Box World

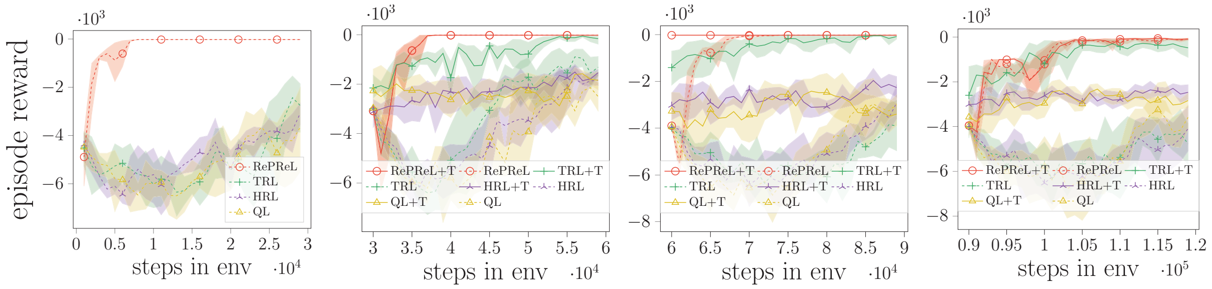
passengers ($p1$, $p2$, and $p3$) to their respective destinations in that order. Here, the taxi location and the passenger pickup and drop locations are randomly sampled at the beginning of each episode. Passenger pickup and drop are restricted to 4 special locations (r, g, b, or y) whereas the taxi can be sampled at any location.

Our third domain, shown in Figure 4, is inspired by the Box World domain in Zambaldi et al [48]. This domain has 4 types of objects: *lock, key, gem, wall* with an associated color for each of them. A lock can be opened with a key of the same color to access the adjacent key. The player is equipped with sensors on each of its 8 directions (northeast,north,east,...), 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, of 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**, the 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.

In these discrete domains, we employed two relational RL base learners: Q-tree [9] and Gradient Boosted Q-Learning [5]. However, these RRL methods could **not** converge to an optimal policy. Consequently, we use propositional tabular Q-learning as the base learner to evaluate the RePReL framework. Propositional state abstractions for all the options are obtained by grounding the parameters, e.g. $p1$ in `pickup(p1)`, to generic objects (Skolem constants in logic) and unrolling the D-FOCI statements. The complete list of D-FOCI statements and the relevant state abstractions are provided in Table 2.

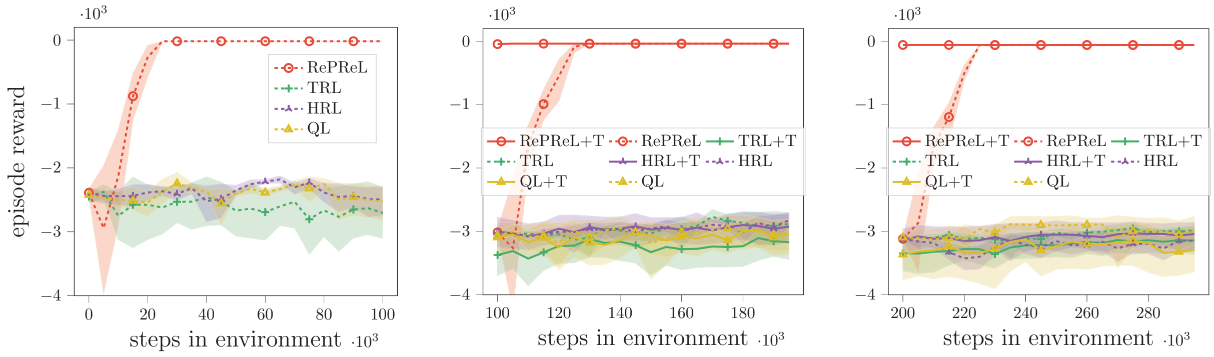
To evaluate the sample efficiency induced by RePReL abstractions, we compare it against the *seq* variant of the Taskable RL [21]. We pick this variant for two reasons: 1. The *seq* variant performed best in all their experiments, and 2. We aim to evaluate the effectiveness of abstractions and thus do not learn the meta-controller introduced by the partially ordered plans. We set a budget on the number of steps for learning in each task and evaluate the performance of RePReL against Taskable RL (TRL), option-based Hierarchical RL (HRL), and Q-Learning (QL). To evaluate the transfer, we modified the RePReL learning algorithm. Specifically, the RL policies were initialized with transferred policy in line 1 of Algorithm 1. We transferred the learned policies from *Task 1* to *Task 2*, refined the policy on *Task 2*, transferred it to *Task 3*, and so on, in increasing order. We indicate the transferred agent performance with a solid line in the plots and append "+T" in the legends.

Figure 5 compares the learning curves of RePReL with TRL, HRL, and QL agents in **Office World** with a 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 option for pickup and visit operation as they have the same preconditions and effects. Fig. 5a shows that RePReL achieves the optimal reward for *Task 1* in less than 10K steps, while the baseline methods TRL, HRL, and QL do not achieve optimality even after 30K steps. Similarly in all the other tasks, Fig. 5b–5d, we see that RePReL consistently outperforms TRL, HRL, and QL by converging to the optimal policy in less than 15K steps. Hence, Fig. 5 demonstrate sample efficiency, aspect A1. Further, Tasks 2 and 3 demonstrate that transferred RePReL policies (RePReL+T) have a distinct advantage in terms of sample efficiency when the task is closely related to a previously learned task. *Task 4* is independent of *Task 1, 2,* and *3*, and thus, the gain due to transfer is not significant. Yet, RePReL+T converges faster than TRL+T. Hence, Fig. 5b and 5c demonstrate the aspect A2, effective transfer across tasks.



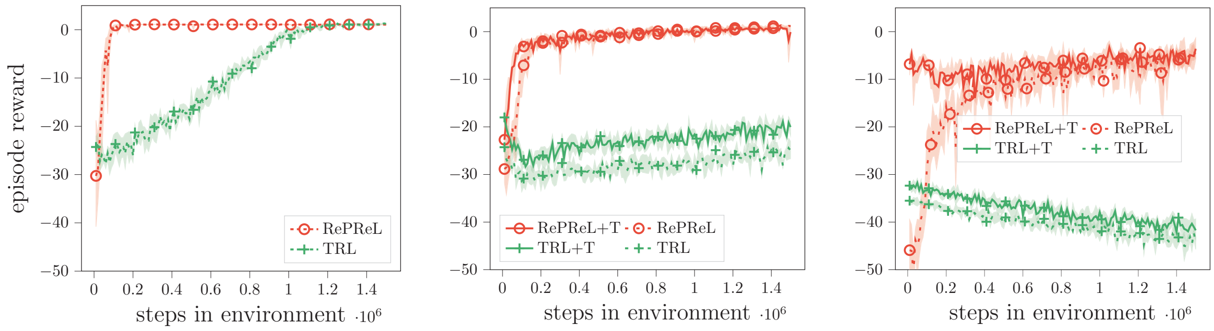
(a) Task 1 (b) Task 2 (c) Task 3 (d) Task 4

Fig. 5: Comparing learning curves of RePReL, TRL, HRL, and QL in the Office World environment. Transfer algorithms are indicated by "+T". **(a)** *Task 1* is deliver mail, **(b)** *Task 2* is deliver coffee, **(c)** *Task 3* is deliver mail and coffee, and **(d)** *Task 4* is visit A, B, C, D. Note that the RePReL and RePReL+T curves in Task 2 are overlapping. The shaded region depicts the standard deviation.



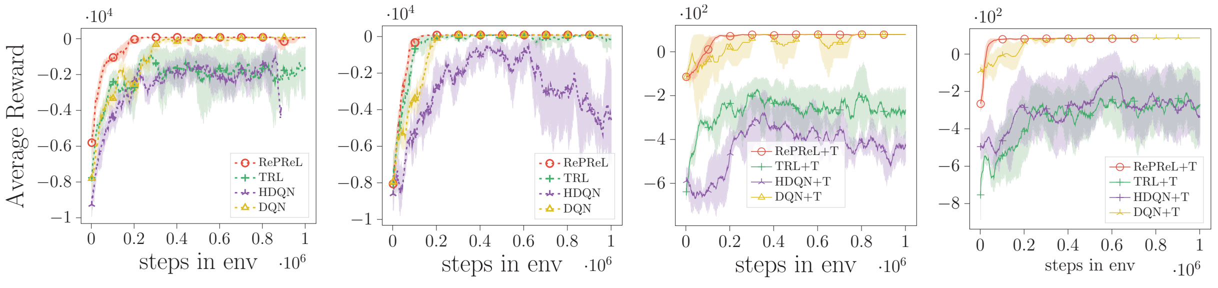
(a) Task 1 (b) Task 2 (c) Task 3

Fig. 6: Comparing learning curves of RePReL, TRL, HRL, and QL in the Extended Taxi World. **(a)** *Task 1* is to drop passenger p1, **(b)** *Task 2* is to drop p1 and p2, **(c)** *Task 3* is to drop p1, p2, p3.



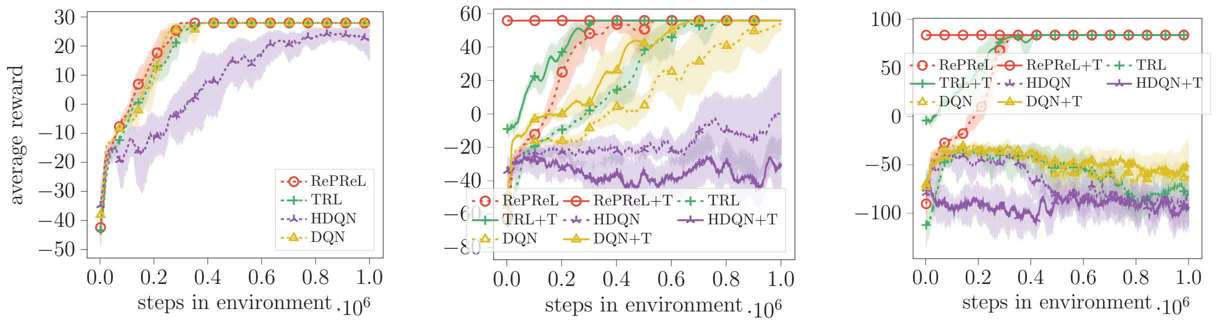
(a) Task 1 (b) Task 2 (c) Task 3

Fig. 7: Comparing learning curves of RePReL and TRL with and without transfer in the Box World environment. **(a)** *Task 1* is to open the lock and reach the gem. **(b)** *Task 2* is to collect the key and then open the lock to reach the gem. **(c)** *Task 3* requires the agent to open two locks in sequence to reach the gem. The shaded region depicts the standard deviation.



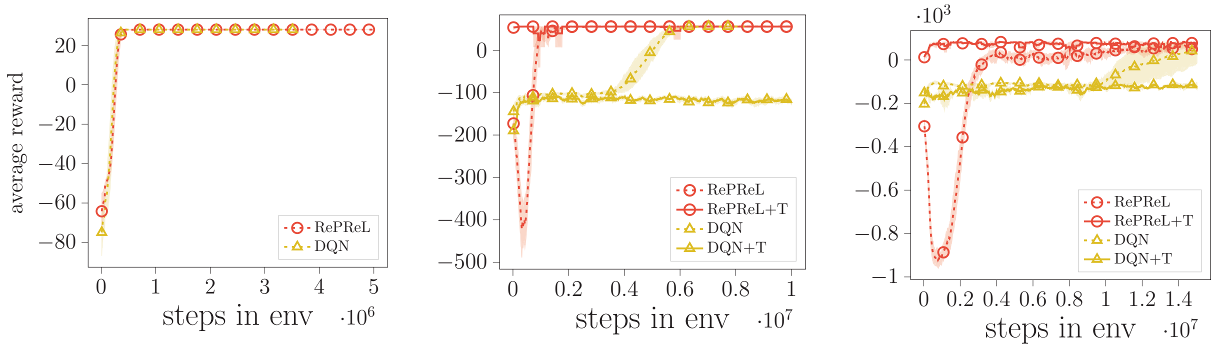
(a) Task 1 (b) Task 2 (c) Transfer to Task 1 (d) Transfer to Task 2

Fig. 8: Comparing learning curves of deep RL based learners in the Office World environment. RePReL compared against TRL, HDQN and DQN in **(a)** *Task 1* deliver mail and **(b)** *Task 2* deliver coffee. Agents are swapped after 1*e*6 steps. RePReL+T compared against TRL+T, HDQN+T and DQN+T in **(c)** *Task 1* and **(d)** *Task 2*. The shaded region depicts the standard deviation.



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

Fig. 9: Comparing learning curves of deep RL-based learners in the Extended Taxi World. **(a)** *Task 1* is to drop passenger p1,**(b)** *Task 2* is to drop p1 and p2, **(c)** *Task 3* is to drop p1, p2, p3. The shaded region depicts the standard deviation.



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

Fig. 10: Comparing learning curves of deep relational RL-based learners in the Extended Taxi World. **(a)** *Task 1* is to drop passenger p1, **(b)** *Task 2* is to drop p1 and p2, **(c)** *Task 3* is to drop p1, p2, and p3. The shaded region depicts the standard deviation.

We present, in Fig. 6, the comparison of the RePReL with baselines in the **Extended Taxi World**. In this domain, both TRL and HRL use 4 options for each location R, G, B, and Y, while RePReL uses only two options, one for picking up and the other for dropping a passenger. The results clearly show that RePReL consistently outperforms both baselines in terms of sample efficiency (A1). 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, aspect A3.

Our experiments in the **Relational Box World** domain are presented in Fig. 7. Here, we use two subtask policies for both the TRL and RePReL: one for opening the lock and another for collecting the 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 $1.5M$ steps for training in each task. We do not present the HRL and QL baselines for this task as they did not converge even after $5M$ training steps. We see that RePReL is significantly more efficient than TRL, in all three tasks (A1). Here, *Task 2* involves opening one box (i.e. collecting the key and opening the lock) to reach the gem and *Task 3* requires opening two boxes. It can be clearly observed that RePReL+T is able to generalize across a number of objects when going from *Task 2* to *Task 3*. These transfer results in Extended Taxi World and Relational Box World demonstrate that RePReL generalizes well across a varying number of objects, aspect A3.

4.2 Deep RL

Next, we aim to assess the RePReL framework with deep RL algorithms. For this, we used double DQN as the base learner, from the implementation available in the RL-kit package[4]. We used *Task 1* and *2* of Office World to evaluate the transfer efficiency and *Task 1–3* of the Extended Taxi World to evaluate generalization capability. We compare our deep RePReL agent against a double *Deep Q-Network (DQN)*, a *Hierarchical DQN (HDQN)*, and a *Taskable-RL (TRL)* agent. Hyperparameters and network architecture were tuned for the DQN agent and used verbatim for other agents. These hyperparameter values are summarized in Table 3. We set a budget of $1e6$ on the number of steps that can be taken in the training environment for learning in each task.

Figures 8 and 9 compare the learning curves of the four agents on two tasks in Office World and three tasks in Extended Taxi World. While the RePReL, TRL and DQN agents achieve comparable performance in *Task 2* of Office world (Fig. 8b) and *Task 1* of Extended Taxi World (Fig. 9a), their performance significantly differs in the remaining tasks. Solid lines with "+T" are transferred agents. We see that all the transferred agents start at a higher average reward, but the RePReL agent has the steepest learning curve. Hence, Figures 8 and 9 demonstrate effective transfer (A2) and generalization across objects (A3), respectively.

4.3 Deep Relational RL

Recently, deep relational RL approaches are proposed that learn policies that can generalize across objects [25, 31, 48]. These approaches use GNN-based architecture and expect the input state representation as a graph. In GNN, a graph is a directed, multi-attribute graph that has nodes, node attributes, directed edges, edge attributes, and global graph attributes [3]. The computational units in GNN are combinations of "aggregate" and "update" functions that take various attributes as input, aggregate them, and update either the node, the edge, or the global attributes of the graph. These computational units are permutation invariant and support different graph sizes in the input. Formally, a graph is represented as a 3-tuple $G = (\mathbf{u}, V, E)$, where $\mathbf{u}$ is a global attribute, $V = \{\mathbf{v_i}\}_{i=1:|V|}$ is a set of nodes and $\mathbf{v_i}$ a node attribute, and $E = \{(\mathbf{e_k}, r_k, s_k)\}_{k=1:|E|}$ is a set of edges with edge attributes ($\mathbf{e_k}$), receiver node index (r_k) and sender node index (s_k).[5] All the computational blocks are some combination of the following aggregate functions (ρ) and update functions (ϕ),

[4]https://github.com/rail-berkeley/rlkit

[5]Boldfont indicates a vector.

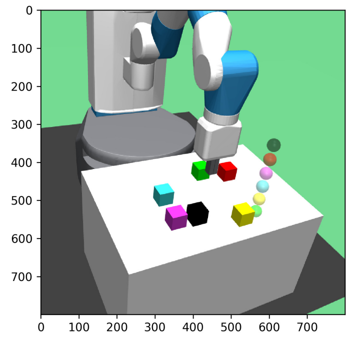


Fig. 11: FetchBlockConstruction

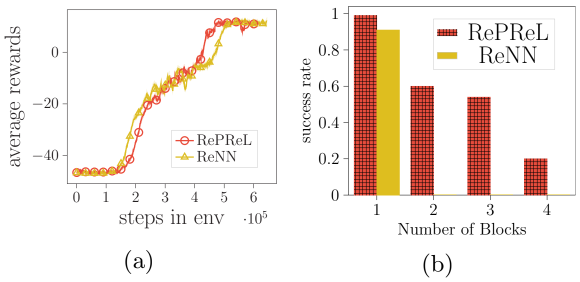


Fig. 12: Comparing deep relational RL-based learners in FetchBlockConstruction. (a) Compares learning curve on the task of moving one block (b) Evaluates generalization for moving 1–4 blocks.

$$\begin{aligned} \mathbf{e}'_k &= \phi^e\left(\mathbf{e}_k, \mathbf{v}_{r_k}, \mathbf{v}_{s_k}, \mathbf{u}\right) & \overline{\mathbf{e}}'_i &= \rho^{e\rightarrow v}\left(E'_i\right) \\ \mathbf{v}'_i &= \phi^v\left(\overline{\mathbf{e}}'_i, \mathbf{v}_i, \mathbf{u}\right) & \overline{\mathbf{e}}' &= \rho^{e\rightarrow u}\left(E'\right) \\ \mathbf{u}' &= \phi^u\left(\overline{\mathbf{e}}', \overline{\mathbf{v}}', \mathbf{u}\right) & \overline{\mathbf{v}}' &= \rho^{v\rightarrow u}\left(V'\right) \end{aligned}$$

$$\begin{aligned} \text{where } E'_i &= \left\{\left(\mathbf{e}'_k, r_k, s_k\right)\right\}_{r_k=i, k=1:|E|}, \\ V' &= \left\{\mathbf{v}'_i\right\}_{i=1:|V|}, \\ \text{and } E' &= \bigcup_i E'_i = \left\{\left(\mathbf{e}'_k, r_k, s_k\right)\right\}_{k=1:|E|}. \end{aligned}$$

Since we aim to compare the generalization capability across varying objects, we evaluate the RePReL framework with deep relational RL as base learners on two domains. The relational state is represented as a *fully-connected graph* in both domains. The first domain is the **Extended Taxi World** described earlier. We represent each passenger as a node in the graph, passenger features as node attributes, and the taxi location as a global attribute of the graph.

All the previous domains used discrete state and action spaces. To demonstrate that RePReL can handle both continuous and discrete spaces (A4) we include a 3-dimensional robotics multi-object manipulation domain with continuous state and action spaces called **FetchBlockConstruction** [31]. FetchBlockConstruction, shown in Figure 11 is based on FetchPickAndPlace [39]. This domain has multiple blocks and the task is to move each block to its goal location. Each block is represented as a node in the fully-connected graph with an 18D vector consisting of block position, orientation, relative position, cartesian velocity, angular velocity, and goal location. Features of the robotic gripper (10D vector) are treated as global attributes of the graph. The action space is 4D, consisting of relative change in the 3D position of the two-fingered parallel jaw gripper and the distance between two fingers of the robotic arm.

We compare the RePReL framework against a double DQN in Extended Taxi World. In FetchBlockConstruction, we use the state-of-the-art deep relational RL called Relational Neural Network (ReNN) by Li et al [31]. ReNN uses a Soft Actor-Critic (SAC) learner with hindsight experience replay (HER) strategy. In both domains, the network architecture consists of an attention-based message passing module, an attentive graph pooling module, and a multi-layer-perceptron module. This also demonstrates that RePReL framework can be adapted to any off-policy RL algorithm, aspect A5. For FetchBlockConstruction we use the same network parameters and experiment settings as Li et al [31], except for one change. Instead of using 35 parallel workers, we used 16 workers because of resource constraints. Table 4 summarizes the network parameters for the Extended Taxi World.

Figure 10 presents the comparison of the RePReL with GNN based DQN agent. Both the agents have comparable performance in *Task 1*, but RePReL shows significant advantage over DQN in the remaining tasks (A1 and A3). The transferred DQN agents took longer to converge than learning DQN from scratch on *Task 2* and *3*. This might be explained by the mechanism called *capacity loss*, whereby networks trained to predict a sequence of target values lose their ability to quickly fit new functions over time [2, 20, 32]. Figure 12a presents the comparison of learning curves of the RePReL agent and the ReNN agent

D-FOCI statements	**Option: Abstract state**
Office World	
{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): {agent-at(L1), at(X, L), with-agent(X), move(Dir)}
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) $\longrightarrow R_o$	deliver(X): {agent-at(L1), with-agent(X), office(L), move(Dir), delivered(X)}
Extended Taxi World	
{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): {taxi-at(L1), at(P,L), in-taxi(P), move(Dir)}
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) $\longrightarrow R_o$	drop(P): {taxi-at(L1), in-taxi(P), dest(P,L), at-dest(P), move(Dir)}
Relational Box World	
{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)	pick_key(K): {neighbor(Dir,C), agent-at(L1), direction(K, Dir2), own(K), move(D)}
unlock(L): open(L) $\longrightarrow R_o$ unlock(L): {agent-at(L1), direction(L, Dir2), open(L)} $\xrightarrow{+1}$ open(L)	unlock(L): {neighbor(Dir,C), agent-at(L1), direction(L, Dir2), open(L),move(D)}
FetchBlockConstruction	
{armfeat1(AF1), ..., armfeat10(AF10) , action(A)} $\xrightarrow{+1}$ armfeat1(AF1), ... {armfeat1(AF1), ..., armfeat10(AF10) , action(A)} $\xrightarrow{+1}$ armfeat10(AF10), place(X): {blockfeat1(X, BF1), ..., blockfeat18(X, BF18) } $\xrightarrow{+1}$ blockfeat1(X, BF1), ... place(X): {blockfeat1(X, BF1), ..., blockfeat18(X, BF18) } $\xrightarrow{+1}$ blockfeat1(X, BF1), place(X): {armfeat1(AF1), ..., armfeat10(AF10), action(A) } $\xrightarrow{+1}$ blockfeat1(X, BF1), ... place(X): {armfeat1(AF1), ..., armfeat10(AF10) , action(A) } $\xrightarrow{+1}$ blockfeat18(X, BF18), place(X): {blockfeat1(X, BF1), ..., blockfeat18(X, BF18) $\longrightarrow R_o$	place(X): { armfeat1(AF1), ... armfeat10(AF10), blockfeat1(X, BF1), ... blockfeat18(X, BF18), action(A)}

Table 2: D-FOCI statments and relevant features (literals) of the state that form the abstract state.

Hyperparameters	**Values**
Learning rate	0.003
Batch size	128
Max steps	1e6
Max buffer size	1e5
Hidden layers	2
Hidden units	256
Discount rate	0.99
Intrinsic reward on subgoals	30
Office World	
Input size (baseline)	11
Input size (RePReL)	4 pickup & 5 drop
Output size (# of Actions)	4
Output size (metacontroller)	3
Max episode length	1000
Epsilon decay	True
Extended Taxi World	
Input size (baseline)	91
Input size (RePReL)	69
Output Size (# of Actions)	6
Output Size (metacontroller)	2
Max episode length (Task 1)	150
Max episode length (Task 2)	200
Max episode length (Task 3)	300
Epsilon decay	false

Table 3: Summary of hyperparameters used in deep RL experiments (Section 4.2).

Hyperparameters	**Values**
Number of graph layers	2
Attention embedding	125
Number of attention heads	3
Number of MLP layers	2
MLP embedding	256
Activation function	Leaky ReLU
Batch size	128
Task 1 max episode length	500
Task 2 max episode length	1000
Task 3 max episode length	1500
Epsilon decay	True
Learning rate	0.003
Buffer size	1e6

Table 4: Summary of hyperparameters used in deep relational RL experiment of Extended Taxi World (Section 4.3).

on the task of moving a block to its goal location. While RePReL and ReNN have comparable performance on this task, we present the generalization results to 2–4 blocks in Figure 12b. As can be seen, the RePReL agent can easily generalize to multiple objects while being statistically significantly better than the baselines (demonstrating aspect A3).

4.4 Discussion

It is important to note one of the significant differences between the RePReL learning algorithm and the Taskable RL learning algorithm. 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.

Collectively, the empirical evaluations clearly demonstrate our central hypothesis that **RePReL allows for better generalization while exploiting effective abstractions for efficient learning**. The results further demonstrate another important aspect of this formalism—the ability to learn in continuous as well as discrete structured domains. Learning in this setting is generally considered a hard task, and our work takes a small step towards this goal. Two important assumptions were made in this work—that both the plan decomposition and the D-FOCI statements were specified in advance by the human expert, and most importantly, they are precise and correct. Relaxing these assumptions is an immediate future research direction.

5 Conclusion

We introduced RePReL, a unified framework that enables efficient learning to act in structured domains. The framework consists of three specific components—a symbolic planner, a state abstractor, and an RL agent. The planner decomposes the tasks into smaller options, the state abstractor computes the appropriate abstractions for the lowest level MDP, and finally, an RL agent learns quickly and effectively given the smaller MDP. Our experiments demonstrate that the RePReL is not only effective and efficient but generalizes to unseen tasks and a larger number of objects. Interesting directions for future research include allowing communication from the RL agent back to the planner, allowing differentiable learning at the high level, and automatically refining the abstraction rules.

Acknowledgments. HK and SN gratefully acknowledge the support of ARO award W911NF2010224 and AFOSR award FA9550-18-1-0462. PT acknowledges the support of DARPA contract N66001-17-2-4030 and NSF grant IIS-1619433. Any opinions, findings, conclusions 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. We also thank the Starling lab members for feedback on the manuscript.

Statements and Declarations

Competing Interests. The research leading to these results received funding from federal grants as mentioned in the acknowledgments. Specifically, HK and SN received the support of ARO (award W911NF2010224) and AFOSR (award FA9550-18-1-0462). PT received the support of DARPA (contract N66001-17-2-4030) and NSF (grant IIS-1619433). No conflict of interest exists with this work.

Code availability.
https://github.com/starling-lab/RePReL

References

[1] Andrychowicz M, Wolski F, Ray A, et al (2017) Hindsight experience replay. In: NeurIPS, pp 5048–5058

[2] Ash JT, Adams RP (2020) On warm-starting neural network training. In: Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual

[3] Battaglia PW, Hamrick JB, Bapst V, et al (2018) Relational inductive biases, deep learning, and graph networks. CoRR abs/1806.01261

[4] Brafman RI, Tennenholtz M (2002) R-max-a general polynomial time algorithm for near-optimal reinforcement learning. JMLR 3:213–231

[5] Das S, Natarajan S, Roy K, et al (2020) Fitted q-learning for relational domains. CoRR abs/2006.05595

[6] Dietterich TG (1998) The maxq method for hierarchical reinforcement learning. In: ICML, pp 118–126

[7] Dong H, Mao J, Lin T, et al (2019) Neural logic machines. In: ICLR

[8] Driessens K, Ramon J, Blockeel H (2001) Speeding up relational reinforcement learning through the use of an incremental first order decision tree learner. In: ECML, pp 97–108

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

[10] Eppe M, Nguyen PDH, 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

[11] Evans R, Grefenstette E (2018) Learning explanatory rules from noisy data. JAIR 61:1–64

[12] Fern A, Yoon S, Givan R (2006) Approximate policy iteration with a policy language bias: Solving relational markov decision processes. JAIR 25:75–118

[13] Ghallab M, Nau D, Traverso P (2004) Automated Planning: theory and practice. Elsevier

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

[15] Grounds M, Kudenko D (2005) Combining reinforcement learning with symbolic planning. In: AAMAS III, pp 75–86

[16] Guestrin C, Patrascu R, Schuurmans D (2002) Algorithm-directed exploration for model-based reinforcement learning in factored mdps. In: ICML, pp 235–242

[17] Guestrin C, Koller D, et al (2003) Generalizing plans to new environments in relational mdps. In: IJCAI, pp 1003–1010

[18] Haarnoja T, Zhou A, Abbeel P, et al (2018) Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In: ICML, pp 1861–1870

[19] van Hasselt H, Guez A, Silver D (2016) Deep reinforcement learning with double q-learning. In: AAAI, pp 2094–2100

[20] Igl M, Farquhar G, Luketina J, et al (2021) Transient non-stationarity and generalisation in deep reinforcement learning. In: International Conference on Learning Representations

[21] Illanes L, Yan X, Icarte RT, et al (2020) Symbolic plans as high-level instructions for reinforcement learning. ICAPS pp 540–550

[22] Janisch J, Pevný T, Lisý V (2021) Symbolic relational deep reinforcement learning based on graph neural networks. RL4RealLife @ ICML

[23] Jiang Y, Yang F, Zhang S, et al (2019) Task-motion planning with reinforcement learning for adaptable mobile service robots. In: IROS, pp 7529–7534

[24] Jiang Z, Luo S (2019) Neural logic reinforcement learning. In: ICML, vol 97. PMLR, pp 3110–3119

[25] Jiang Z, Minervini P, Jiang M, et al (2021) Grid-to-graph: Flexible spatial relational inductive biases for reinforcement learning. In: AAMAS. ACM, pp 674–682

[26] Kimura D, Ono M, Chaudhury S, et al (2021) Neuro-symbolic reinforcement learning with first-order logic. In: EMNLP, pp 3505–3511

[27] Kokel H, Manoharan A, Natarajan S, et al (2021) Reprel: Integrating relational planning and reinforcement learning for effective abstraction. ICAPS 31(1):533–541

[28] Kokel H, Manoharan A, Natarajan S, et al (2021) Dynamic probabilistic logic models for effective abstractions in RL. CoRR abs/2110.08318

[29] Konidaris G, Kaelbling LP, Lozano-Perez T (2018) From skills to symbols: Learning symbolic representations for abstract high-level planning. JAIR

[30] Li L, Walsh TJ, Littman ML (2006) Towards a unified theory of state abstraction for mdps. In: ISAIM, p 5

[31] Li R, Jabri A, Darrell T, et al (2020) Towards practical multi-object manipulation using relational reinforcement learning. In: ICRA. IEEE, pp 4051–4058

[32] Lyle C, Rowland M, Dabney W (2022) Understanding and preventing capacity loss in reinforcement learning. In: International Conference on Learning Representations

[33] Lyu D, Yang F, Liu B, et al (2019) SDRL: Interpretable and data-efficient deep reinforcement learning leveraging symbolic planning. In: AAAI, pp 2970–2977

[34] Manfredotti CE (2009) Modeling and inference with relational dynamic bayesian networks. In: CCAI, pp 287–290

[35] Natarajan S, Tadepalli P, et al (2005) Learning first-order probabilistic models with combining rules. In: ICML, pp 609–616

[36] Natarajan S, Tadepalli P, et al (2008) Learning first-order probabilistic models with combining rules. Ann Math Artif Intell 54(1-3):223–256

[37] Nau D, Cao Y, Lotem A, et al (1999) Shop: Simple hierarchical ordered planner. In: IJCAI, pp 968–975

[38] Parr R, Russell SJ (1998) Reinforcement learning with hierarchies of machines. In: NeurIPS, pp 1043–1049

[39] Plappert M, Andrychowicz M, Ray A, et al (2018) Multi-goal reinforcement learning: Challenging robotics environments and request for research. arXiv preprint arXiv:180209464

[40] Ravindran B, Barto AG (2003) Smdp homomorphisms: An algebraic approach to abstraction in semi markov decision processes. In: IJCAI, pp 1011–1018

[41] Ravindran B, Barto AG (2003) SMDP homomorphisms: An algebraic approach to abstraction in semi-markov decision processes. In: IJCAI. Morgan Kaufmann, pp 1011–1018

[42] Riegel R, Gray AG, Luus FPS, et al (2020) Logical neural networks. CoRR abs/2006.13155

[43] 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

[44] Sutton RS, Precup D, Singh SP (1998) Intra-option learning about temporally abstract actions. In: ICML, pp 556–564

[45] Sutton RS, Precup D, Singh SP (1999) Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. Artif Intell 112(1-2):181–211

[46] Vlasselaer J, Meert W, et al (2014) Efficient probabilistic inference for dynamic relational models. In: StarAI @ AAAI

[47] Yang F, Lyu D, Liu B, et al (2018) Peorl: Integrating symbolic planning and hierarchical reinforcement learning for robust decision-making. IJCAI pp 4860–4866

[48] Zambaldi V, Raposo D, et al (2019) Deep reinforcement learning with relational inductive biases. In: ICLR

[49] Zhang L, Li X, Wang M, et al (2021) Off-policy differentiable logic reinforcement learning. In: ECML PKDD, pp 617–632