Deep RePReL–Combining Planning and Deep RL for acting in relational domains

Anonymous Author(s)
Affiliation
Address
email

Abstract

We consider the problem of combining a symbolic planner and a Deep RL agent to achieve the best of both worlds – the generalization ability of the planner with the effective learning ability of Deep RL. To this effect, we extend a previous work of Kokel et al. [17], RePReL, to Deep RL. As we demonstrate in experiments in two relational worlds, this combination enables effective learning, transfer and generalization when compared to the use of only Deep RL.

1 Introduction

The two sequential decision making directions of Reinforcement Learning (RL) and AI Planning are complimentary. While Deep RL agents are able to solve complex games such as Go, Atari etc. [32, 24], they are not only data hungry (requiring 100M examples) but also suffer from the lack of generalization to related tasks and domains. Generalization is essential for the deployment of RL agents. As an example, an agent trained to stack 5 red blocks as a pyramid should be able to stack 10 blue blocks as a pyramid. Contrary to RL approaches, symbolic AI planning methods [12] have focused on developing domain-independent approaches that can be utilized in different domains and are able to scale with varying objects. However, most AI planning methods rely on complete (symbolic) description of the domain to guide the search. This limits the usage of Planning approaches in complex domains where it is difficult for humans to articulate the complete transition functions, the pre- and post-conditions of the Planner. In summary, the symbolic representation of Planners allow for efficient generalization while Deep RL methods perform efficient and effective learning.

Consequently, various methods have combined RL and Planning to improve sample efficiency and generalization [14, 16, 7, 36, 22, 23, 19, 10]. We aim to highlight the advantages of combining Planning with RL and present the capability of one such framework, RePReL [17], to learn Deep RL agents. RePReL framework uses a relational hierarchical planner at the higher-level to decompose a task into sub-tasks (options), and learns RL policies for options at lower level while using task-specific state representations. This provides two key advantages. First, the hierarchical planner effectively captures the symbolic, generalized, higher-level task knowledge, which is more general than simply providing intrinsic rewards. Second, learning separate policy for each option at lower level allows for task-specific abstract state representations. By incorporating Deep RL agent in the RePReL framework, we provide a third advantage. We empower lower level options in RePReL to learn complex functions capable to act in continuous state and action spaces. In the proposed Deep ReRPeL framework, the planner serves as the higher level, symbolic, deliberate slow reasoner while the Deep RL is the fast, effective neural learner, thus achieving the two-level neurosymbolic system, a key goal of many AI/ML researchers.

We focus on relational domains, where the number of entities and relations are not fixed in advance. To learn to act in such relational domains, it is important for the agent to identify relevant entities for

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

the given task and focus only on the applicable properties and relations of these entities. For example, while learning to stack all the blocks in a single tower, the agent should not over-fit to the color of the blocks. In RePReL framework, we leverage the domain knowledge of humans to infer the relevant properties using first-order conditional influence statements. In this work-in-progress report, we share our initial results on two domains, extended relation taxi domain and office world. We show the capability of Deep RePReL framework to generalize over different objects in the domain and transfer to different task.

2 Proposed Approach

We consider the problem of learning to execute various tasks in relational domains. Thus, we extend the relational MDP (RMDP) definition of Fern et al. [9] 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 $S \times A \times S \rightarrow \mathbb{R}$, 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. We use a seperate intrinsic reward t_R when the terminal state for any goal or subgoal is reached. 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. The RePReL framework proposes that the GRMDPs can be solved using a combination of planning and RL in 3 stages.

Stage 1 Planning: Use the hierarchical planner to decompose the goal of the GRMDP into a sequence of subgoals using decomposition rules or *methods*. These high-level subgoals act as options.

Stage 2 Abstraction: Use human's task knowledge to abstract the state representations by filtering the irrelevant entities, properties and relationships. This provides sub-task specific state representations.

Stage 3 RL: Learn separate RL policies for each option to perform sub-tasks in abstract state space.

We refer to Kokel et al. [17] for details of the hierarchical planner. Here it suffices to say that given a state s and goal g, the hierarchical planner provides a high-level plan $\Pi = [o_1, o_2, ..., o_n]$ which is a sequence of options (or operators). Each option o is defined with initiation set $I(o)$ (preconditions of operator) and termination condition $\beta(o)$ (necessary effects of operator). A subgoal RMDPs M_o is defined for each option o which is solved to obtain the option policy π_o.

Definition 2. *The subgoal RMDP M_o for each option 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:*

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

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

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

Essentially, the reward function in the original GRMDP would correspond to the step cost function which applies to all operators, or options, and reward R_o is the only goal-specific reward. Next, we present how we obtain safe state abstraction using the bisimulation framework of Givan et al. [13] and Ravindran and Barto [29], which has been called "model agnostic abstraction" in Li et al. [20].

Definition 3 (Li et al. [20]). *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. The second condition indicates that these two states have the same transition dynamics. It is proven that Q-learning with such abstraction results in an optimal policy for ground MDP [20].

Since states are conjunctions of literals in RMDPs, to get safe abstraction, we need to infer which predicates influence the rewards and goals of options. We capture this knowledge using First-Order Conditional Influence (FOCI) statements [25], one of the many variants of statistical relational learning languages [11, 28]. 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 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 from the literals in the current time step to the literals in the next time step. To distinguish the two kinds of influences, we add $^{+1}$ on the arrow between the sets of literals to capture a temporal interaction, as `option` : $\{\mathtt{p}(X_1), q(X_1)\} \xrightarrow{+1} \mathtt{q}(X_1)$. It says that, for the given `option`, 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. `option` may be skipped to represent unconditional influences.

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` option can be expressed as, `pickup(P)` : $\{$`taxi-at(L1)`, `at(P, L2)`$\} \xrightarrow{+1}$ `in-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 option determined by the planner, the D-FOCI statements would be partially grounded. For example, using the $\theta = \{X/p1, L/r\}$ for `pickup(X)` the above D-FOCI would become `pickup`$(p1)$: $\{$`taxi-at(L1)`, `at`$(p1, r)\} \xrightarrow{+1}$ `in-taxi`$(p1)$. Hence, only the pickup location of passenger $p1$ (i.e., `at`$(p1, r)$) and current taxi location `taxi-at(L1)` (i.e., the current location `L1`) are relevant for grounded operator `pickup`$(p1)$. If there is another passenger, say $p2$, then the state variable `at`$(p2, \cdot)$ would not be relevant.

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 option 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. 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. We defer to Kokel et al. [17] for the proof and unrolling process. We present the learning procedure next.

Given the GRMDP environment env, planner $\mathfrak{P}$ and FOCI statements F, we now discuss the Deep RePReL learning procedure from **Algorithm 1**. First for each option, an RL policy π_o and a replay buffer $\mathcal{D}_O$ is initialized in **line 1, 2**. Next for each episode, in **line 6**, we get the high level plan Π from the planner $\mathfrak{P}$, we employ the SHOP planner Nau et al. [26]. For every grounded option in the plan, we collect the training sample in buffer $\mathcal{D}_o$ (**lines 7–23**). To collect samples, we get an abstract propositional state representation $\hat{s}$. In **lines 11–22**, 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 option o_g, then we add a terminal reward t_R (**line 18**) before adding it to the buffer (**line 20**). Then for each option o, the option policy π_o is updated after sampling a batch from buffer $\mathcal{D}_o$ (**lines 25–28**).

Algorithm 1 RePReL Algorithm

INPUT: Planner $\mathfrak{P}$, Options O, goal set g, env, terminal reward t, D-FOCI statements F, num of iterations i, num of episodes in each iteration k, batch size b
OUTPUT: RL policies $\pi_o, \forall o \in O$

1: $\pi_o \leftarrow 0, \forall o \in O$ ▷ initialize RL policy for each option
2: $\mathcal{D}_o, \forall o \in O$ ▷ initialize buffers for each option
3: **for** iteration $\in i$ **do**
4: **for** episode $\in k$ **do**
5: $s \leftarrow$ get state from env
6: $\Pi \leftarrow \mathfrak{P}(s, g)$ ▷ get high-level plan
7: **for** o_g in Π **do**
8: $\pi \leftarrow \pi_o^{\epsilon}$ ▷ get resp. RL policy
9: $\hat{s} \leftarrow$ GetAbstractState(s, o_g, F)
10: done $\leftarrow \hat{s} \in \beta(o_g)$ ▷ check terminal state
11: **while** not done **do**
12: $a \leftarrow \pi(\hat{s})$ ▷ get action
13: $s' \leftarrow$ env.step(a) ▷ take step in env
14: $r \leftarrow R(s, a, s')$ ▷ get step reward
15: $\hat{s}' \leftarrow$ GetAbstractState(s, o_g, F)
16: done $\leftarrow \hat{s}' \in \beta(o_g)$ ▷ check terminal next state
17: **if** done **then**
18: $r = r + t_R$ ▷ add terminal reward
19: **end if**
20: $\mathcal{D}_o \leftarrow \mathcal{D}_o \cup \{\hat{s}, a, r, \hat{s}', o_g\}$
21: $s, \hat{s} \leftarrow s', \hat{s}'$
22: **end while**
23: **end for**
24: **end for**
25: **for** each option $o \in O$ **do** ▷ Update all the parameters
26: $\mathcal{D} \leftarrow \text{SampleBatch}(\mathcal{D}_o, b)$ ▷ Sample a batch from corresponding buffer
27: $\pi_o \leftarrow \text{UpdatePolicy}(\pi_o, \mathcal{D})$
28: **end for**
29: **end for**
30: **return** $\pi_o, \forall o \in O$

3 Experiments

We now present our initial results in two domains, Taxi and Office World. We design our experiments to explicitly answer the followings questions.

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

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

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

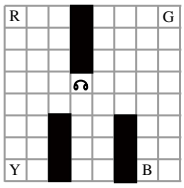


Figure 1: Taxi Domain

Domains: As mentioned earlier, we evaluate our framework on two domains: an extended relational version of the Taxi domain [5] and an Office World [16]. *Extended Taxi domain* is an 8×8 grid, shown in Figure 1, with one taxi and 4 special locations: $\langle R, G, B, Y \rangle$. There can be more than one passenger at a time in the grid; but taxi can be hired only by a single passenger. In every episode, the pickup location and the drop location of each passenger are sampled from the 4 special locations. The agent can perform 6 actions: *up, down, right, left, pick, drop*. Environment provides a reward of

-0.1 for every step and -1 for pick or drop action in wrong locations. We consider three tasks in this domain: *task* 1 is to drop a passenger to its destination, *task* 2 is to drop two passengers, and *task* 3 is to drop three passengers to their destination location.

Office World is a 9×12 grid, shown in Figure 2. It has one office location (indicated with hand), two coffee locations (indicated by mugs), one mail room (indicated by envelope), and some plants in office to be avoided (indicated by $*$). The agent can perform 4 actions: *up, down, right,* and *left*. We consider two tasks in this domain: *task 1* is to deliver coffee to office (which can be picked up by visiting any coffee location) and *task 2* is to deliver mail to office. Environment provides a cost of 1 at every step and cost of 10 on taking invalid actions like stepping on the plant or moving in the walls.

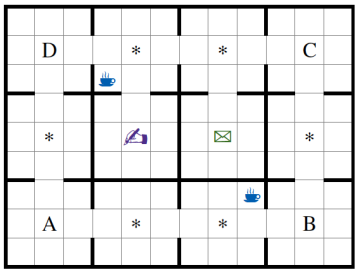


Figure 2: Office World

Baselines We evaluate our RePReL agent against a *Deep Q-Network (DQN)*, an *Hierarchical DQN (HDQN)*, and a *Taskable-RL (TRL)* agent. Mnih et al. [24] proposed **DQN** that uses Q updates to train a Deep RL agent. **HDQN** Kulkarni et al. [18], integrates the temporal abstraction with intrinsic critic by using two-level controller. At the higher level, a meta-controller selects the temporally extended actions, also called *options*, and receives the reward from the environment. While, the lower level controller learns to act in the environment to achieve the option and receives the intrinsic reward from the critic. **TRL** [16] combines symbolic planning with the RL, where the symbolic planner provides the higher-level options and several RL agents are learned for each option at lower level.

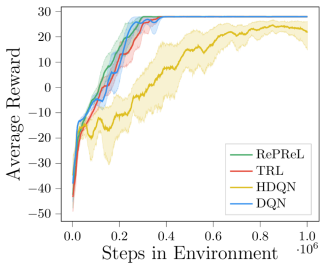
The DQN agent learn a single end-to-end policy for achieving the task; while others decompose the task into subtask and learn policy for achieving subtasks. Further, the HDQN agent learns a meta-controller to generate a high-level plan; while the TRL and the RePReL agent leverage a planner to provide the high-level plan. And finally, the TRL agent uses same state representation to learn each policy; while the RePReL uses D-FOCI statements to obtain the task-specific representation to learn subtask policies. All RL algorithms were implemented on a fork of RL-kit[1]. Table 1 in the Appendix provides details on the hyperparameters used for training all the agents. All the results presented below are aggregated over 10 runs.

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 plans. We set a budget of $1e6$ on the number of steps that can be taken in training environment for learning in each task. Figures 3 (a–e) compares the learning curves of the four agents on all five tasks. While the RePReL, TRL and DQN agent achieves the optimal reward after 40K steps in Taxi *task* 1; their performance significantly differs in Taxi *task* 2 and Taxi *task* 3 where the number of passenger increases. In two tasks of Office World also we see that RePReL achieves the optimal reward faster than others. Hence, we answer **Q1** affirmatively in that RePReL abstractions statistically significantly outperforms the state-of-the-art hybrid planner-RL architecture, Taskable RL, as well as approaches that only learn from data, HDQN and DQN.

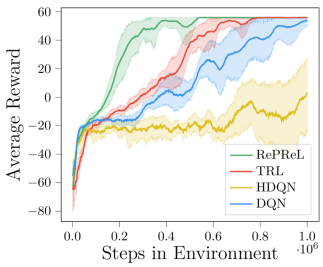
Transfer: We next evaluate these agents on the ability to transfer to a related task in the same domain. To this effect, we transferred all the agents learned for performing *task* 1 and *task* 2 in Office World to perform the *task* 2 and *task* 1, respectively. We swap the task of the agents after they are trained for $1e6$ steps. Performance of the transferred agents are presented in Figures 3 (e–f). All agents start with a higher average reward (than learning from scratch as seen in Fig. 3 (d–e)), but RePReL has the steepest learning curve and achieves optimal reward very fast. This allows us to answer **Q2** affirmatively.

Generalization: For evaluating generalization, we transfer all the agents trained on Taxi *task* 1 to *task* 2 and train them on *task* 2. Subsequently, we transfer agents from *task* 2 to *task* 3. Figures 3 (h–i) presents the learning curve of these transferred agents. While the transferred TRL, HDQN and DQN agents have steeper learning curves than the respective agent learning from scratch (in Fig. 3 (b–c)), the transferred RePReL agent shows zero-shot generalization ability on both tasks. These transfer results in allows us to answer **Q3** affirmatively in that RePReL allows for successful generalization across varying number of objects and is best suited for relational domains.

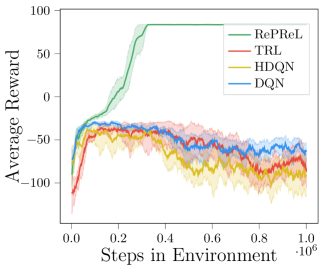
[1] https://github.com/rail-berkeley/rlkit



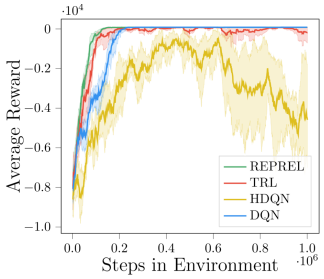
(a) Taxi: Task 1



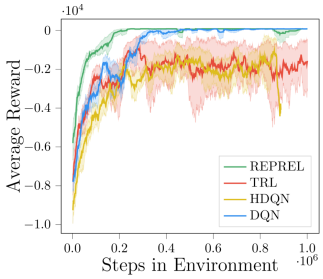
(b) Taxi: Task 2



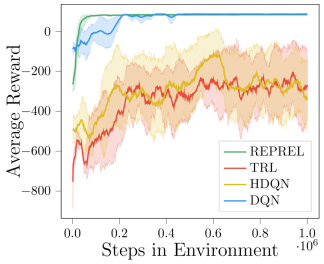
(c) Taxi: Task 3



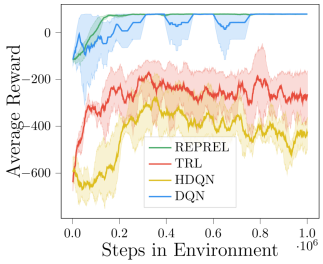
(d) Office: Task 1



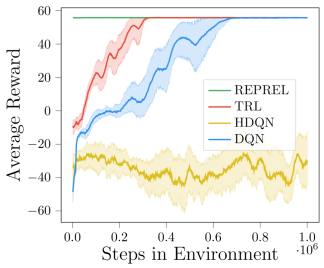
(e) Office: Task 2



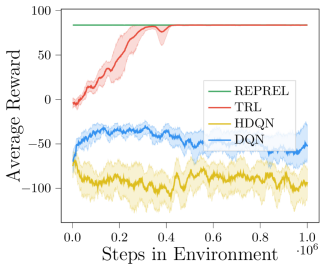
(f) Office: Transfer from Task 2 to Task 1



(g) Office: Transfer from Task 1 to Task 2



(h) Taxi: Generalization from one passenger to two



(i) Taxi: Generalization from two passengers to three

Figure 3: Comparison of RePReL with DQN, HDQN and TRL for sample efficiency in (a–e), for transfer to a different task in (f–g), and for generalization across number of objects in (h–i). In Taxi Domain, Task 1 is to transport one passenger, Task 2 and 3 are to transport 2 and 3 passengers, respectively. In Office World, Task 1 is to deliver coffee and Task 2 is to deliver mail.

4 Discussion and Related Work

State abstractions has been shown to be useful for transfer learning in the literature [35, 33, 1]. Our approach on task-specific state abstraction is inspired from Dietterich [5] and the bisimulation conditions [29, 13] to define abstractions in relational settings using first-order probabilistic models [28]. Relational Reinforcement Learning (RRL) is another set of works that is related to ours. RRL aims to learn a policy in a relational world in the presence of multiple objects. RRL literature [34, 27, 30, 4, 15] tried learning a value function or a policy on a relational representation of the state by using relational induction, symbolic dynamic programming and linear programming alternative but these methods have failed to adapt to mildly complex environments. With the advent of Neuro-Symbolic Theorem provers and Neural Logic Reasoner [8, 6, 31, 3] there is a renewed interest in the community to attempt the relational domains.

While our initial evaluation of RePReL are quite encouraging, there are several avenues that need more evaluation. First is using the Graph Neural Network (GNN) [2] based architecture to learn the policy. Few recent work using Graph-based neural network have shown tremendous improvement over the traditional Neural Network based architecture for relational domains. Specifically, Zambaldi et al. [37] introduce the relational inductive biases in the deep learning using GNNs, claiming that such architectures can perform reasoning, and show zero-shot transfer to complex domains. Inspired by this, Li et al. [21] use GNN architecture for multi-object manipulation and show that a curriculum learning approach can be order of magnitude more sample efficient than the traditional architecture. Hence it is imperative we compare our approach with a GNN based architecture. While this is our current research direction, we emphasize that our proposed RePReL framework is generalizable to any network architecture used for the policy, thus noting the wide applicability of this combination.

References

[1] David Abel, Dilip Arumugam, Lucas Lehnert, and Michael Littman. 2018. State abstractions for lifelong reinforcement learning. In *ICML*, Vol. 80. 10–19.

[2] Peter W Battaglia, Jessica B Hamrick, et al. 2018. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01203* (2018).

[3] William W. Cohen, Fan Yang, and Kathryn Mazaitis. 2020. TensorLog: A Probabilistic Database Implemented Using Deep-Learning Infrastructure. *J. Artif. Intell. Res.* 67 (2020), 285–325.

[4] Srijita Das, Sriraam Natarajan, Kaushik Roy, Ronald Parr, and Kristian Kersting. 2020. Fitted Q-Learning for Relational Domains. *CoRR* abs/2006.05595 (2020).

[5] Thomas G Dietterich. 1998. The MAXQ Method for Hierarchical Reinforcement Learning.. In *ICML*. 118–126.

[6] Honghua Dong, Jiayuan Mao, Tian Lin, Chong Wang, Lihong Li, and Denny Zhou. 2019. Neural Logic Machines. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*.

[7] Manfred Eppe, Phuong D. H. Nguyen, and Stefan Wermter. 2019. From Semantics to Execution: Integrating Action Planning With Reinforcement Learning for Robotic Causal Problem-Solving. *Frontiers in Robotics and AI* 6 (2019), 123.

[8] Richard Evans and Edward Grefenstette. 2018. Learning Explanatory Rules from Noisy Data. *J. Artif. Intell. Res.* 61 (2018), 1–64.

[9] Alan Fern, Sungwook Yoon, and Robert Givan. 2006. Approximate policy iteration with a policy language bias: Solving relational Markov decision processes. *JAIR* 25 (2006), 75–118.

[10] Clement Gehring, Masataro Asai, Rohan Chitnis, Tom Silver, Leslie Pack Kaelbling, Shirin Sohrabi, and Michael Katz. 2021. Reinforcement Learning for Classical Planning: Viewing Heuristics as Dense Reward Generators. *Planning and Reinforcement Learning PRL Workshop at ICAPS* (2021).

[11] Lise Getoor and Benjamin Taskar. 2007. *Introduction to Statistical Relational Learning*. The MIT Press.

[12] Malik Ghallab, Dana Nau, and Paolo Traverso. 2004. *Automated Planning: theory and practice*. Elsevier.

[13] Robert Givan, Thomas Dean, and Matthew Greig. 2003. Equivalence notions and model minimization in Markov decision processes. *Artificial Intelligence* 147, 1-2 (2003), 163–223.

[14] Matthew Grounds and Daniel Kudenko. 2005. Combining reinforcement learning with symbolic planning. In *AAMAS III*, Vol. 4865. 75–86.

[15] Carlos Guestrin, Daphne Koller, Chris Gearhart, and Neal Kanodia. 2003. Generalizing plans to new environments in relational MDPs. In *IJCAI*. 1003–1010.

[16] León Illanes, Xi Yan, Rodrigo Toro Icarte, and Sheila A McIlraith. 2020. Symbolic Plans as High-Level Instructions for Reinforcement Learning. *ICAPS* (2020), 540–550.

[17] Harsha Kokel, Arjun Manoharan, Sriraam Natarajan, Ravindran Balaraman, and Prasad Tadepalli. 2021. RePReL: Integrating Relational Planning and Reinforcement Learning for Effective Abstraction. *ICAPS* 31, 1 (May 2021), 533–541.

[18] Tejas D Kulkarni, Karthik Narasimhan, Ardavan Saeedi, and Josh Tenenbaum. 2016. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In *NeurIPS*. 3675–3683.

[19] Junkyu Lee, Michael Katz, Don Joven Agravante, Miao Liu, Tim Klinger, Murray Campbell, Shirin Sohrabi, and Gerald Tesauro. 2021. AI Planning Annotation in Reinforcement Learning: Options and Beyond. *Planning and Reinforcement Learning PRL Workshop at ICAPS* (2021).

[20] Lihong Li, Thomas J Walsh, and Michael L Littman. 2006. Towards a Unified Theory of State Abstraction for MDPs.. In *ISAIM*, Vol. 4. 5.

[21] Richard Li, Allan Jabri, Trevor Darrell, and Pulkit Agrawal. 2020. Towards Practical Multi-object Manipulation using Relational Reinforcement Learning. In *ICRA*. IEEE, 4051–4058.

[22] Daoming Lyu, Fangkai Yang, Bo Liu, and Steven Gustafson. 2019. A Human-Centered Data-Driven Planner-Actor-Critic Architecture via Logic Programming. In *ICLP*.

[23] Daoming Lyu, Fangkai Yang, Bo Liu, and Steven Gustafson. 2019. SDRL: Interpretable and data-efficient deep reinforcement learning leveraging symbolic planning. In *AAAI*. 2970–2977.

[24] Volodymyr Mnih, Koray Kavukcuoglu, et al. 2015. Human-level control through deep reinforcement learning. *Nature* (2015).

[25] Sriraam Natarajan, Prasad Tadepalli, Thomas G Dietterich, and Alan Fern. 2008. Learning first-order probabilistic models with combining rules. *Ann. Math. Artif. Intell.* 54, 1-3 (2008), 223–256.

[26] Dana Nau, Yue Cao, Amnon Lotem, and Hector Munoz-Avila. 1999. SHOP: Simple hierarchical ordered planner. In *IJCAI*. 968–975.

[27] Bob Price and Craig Boutilier. 2001. Imitation and reinforcement learning in agents with heterogeneous actions. In *Conference of the Canadian Society for Computational Studies of Intelligence*, Vol. 2056. 111–120.

[28] Luc De Raedt, Kristian Kersting, Sriraam Natarajan, and David Poole. 2016. Statistical relational artificial intelligence: Logic, probability, and computation. *Synthesis Lectures on Artificial Intelligence and Machine Learning* 10, 2 (2016), 1–189.

[29] Balaraman Ravindran and Andrew G Barto. 2003. SMDP Homomorphisms: An Algebraic Approach to Abstraction in Semi Markov Decision Processes.. In *IJCAI*. 1011–1018.

[30] Scott Sanner and Craig Boutilier. 2009. Practical solution techniques for first-order MDPs. *Artificial Intelligence* 173, 5-6 (2009), 748–788.

[31] Luciano Serafini, Ivan Donadello, and Artur S. d'Avila Garcez. 2017. Learning and reasoning in logic tensor networks: theory and application to semantic image interpretation. In *Proceedings of the Symposium on Applied Computing, SAC 2017, Marrakech, Morocco, April 3-7, 2017*. ACM, 125–130.

[32] David Silver, Julian Schrittwieser, et al. 2017. Mastering the game of Go without human knowledge. *Nat.* 550, 7676 (2017), 354–359.

[33] Jonathan Sorg and Satinder Singh. 2009. Transfer via soft homomorphisms. In *AAMAS*. 741–748.

[34] Prasad Tadepalli, Robert , and Kurt Driessens. 2004. Relational reinforcement learning: An overview. In *ICML workshop on relational reinforcement learning*. 1–9.

[35] Thomas J Walsh, Lihong Li, and Michael L Littman. 2006. Transferring state abstractions between MDPs. In *ICML Workshop on Structural Knowledge Transfer for Machine Learning*.

[36] Fangkai Yang, Daoming Lyu, Bo Liu, and Steven Gustafson. 2018. PEORL: Integrating symbolic planning and hierarchical reinforcement learning for robust decision-making. *IJCAI* (2018), 4860–4866.

[37] Vinicius Zambaldi, David Raposo, et al. 2019. Deep reinforcement learning with relational inductive biases. In *ICLR*.