Advice-Guided Reinforcement Learning in a non-Markovian Environment

Daniel Neider[1], Jean-Raphael Gaglione[2], Ivan Gavran[1], Ufuk Topcu[3], Bo Wu[3], Zhe Xu[4]
[1] Max Planck Institute for Software Systems, Kaiserslautern, Germany
[2] Ecole Polytechnique, France
[3] University of Texas at Austin, Texas, USA
[4] Arizona State University, Arizona, USA

Abstract

We study a class of reinforcement learning tasks in which the agent receives its reward for complex, temporally-extended behaviors sparsely. For such tasks, the problem is how to augment the state-space so as to make the reward function Markovian in an efficient way. While some existing solutions assume that the reward function is explicitly provided to the learning algorithm (e.g., in the form of a reward machine), the others learn the reward function from the interactions with the environment, assuming no prior knowledge provided by the user. In this paper, we generalize both approaches and enable the user to give *advice* to the agent, representing the user's best knowledge about the reward function, potentially fragmented, partial, or even incorrect. We formalize advice as a set of DFAs and present a reinforcement learning algorithm that takes advantage of such advice, with optimal convergence guarantee. The experiments show that using well-chosen advice can reduce the number of training steps needed for convergence to optimal policy, and can decrease the computation time to learn the reward function by up to two orders of magnitude.

1 Introduction

Reinforcement learning (RL) assumes the environment in which an intelligent agent operates to be modeled by a Markov Decision Process (MDP): the states of the MDP capture the relevant information about the environment, while state-action pairs are equipped with rewards that either reinforce desired or penalize undesired behaviors. In many RL tasks, however, the agent receives its reward sparsely for complex actions over a long period of time.

Learning an optimal policy in such settings is hard: they do not map naturally to MDPs as the reward does not depend on the immediate state of the environment and the chosen action but rather on the history of the actions that the agent has performed—in other words, the reward function is *non-Markovian*. A similar problem occurs under different guises: a non-Markovian dynamics of the MDP or partial observability that creates an illusion of a non-Markovian environment. For the moment, we focus on the non-Markovian reward formulation of the problem and discuss the other formulations with related work.

Clearly, a reward function is Markovian or non-Markovian only with respect to the underlying MDP, and one can augment the states of any MDP with (relevant parts of) the history to obtain an equivalent problem with a Markovian reward function. However, the exact augmentation is crucial: if done naively, the augmented state space becomes too large to be computationally tractable. To overcome this problem, finite-state machines or (equivalently) temporal logic formulas have been proposed to concisely capture the temporal nature of non-Markovian reward functions and make the RL task feasible (Bacchus, Boutilier, and Grove 1996; Jothimurugan, Alur, and Bastani 2019; Icarte et al. 2018b; Brafman, Giacomo, and Patrizi 2018; Camacho et al. 2019). In this paper we focus on a specific type of finite-state machines, called *reward machines*, which have, since proposed by Icarte et al. (2018b), been adopted as a way to encode non-Markovian reward functions.

The approaches above assume that the reward machine is fully known prior to the start of the RL process. However, this assumption is often unrealistic: in practice, reward functions (and, thus, reward machines) are notoriously difficult to find and often not fully known. Luckily, recent research has shown how to eliminate this requirement. For instance, Gaon and Brafman (2020) and Xu et al. (2020) combine automata learning with standard reinforcement learning techniques in order to infer the underlying reward machine through interactions with the environment. Although both Gaon and Brafman's and Xu et al.'s approaches guarantee convergence to the optimal policy and provide convincing experimental evaluations, methods that have to infer the reward machine necessarily converge slower than the ones that assume full knowledge of it.

While assuming full knowledge of the reward machine is rightly deemed overly optimistic (see above), we believe that assuming no knowledge is overly pessimistic. Indeed, a typical situation will be between the two extremes: a human engineer certainly has some high-level knowledge about the reward function/machine and can often provide some sort of "advice".

In this paper, we think of advice as suggesting which of the agent's observation sequences are *promising* (i.e., for which sequences the agent could get a reward, without determining the numerical value of the reward), and which are not. Formally, an observation (later called label sequence) is a word of finite length, and we represent advice as (a number of) deterministic finite automata (DFAs) that accept promising observations. The key argument in favor of choosing DFAs as a formalism for advice is that they are widely known among engineers. Furthermore, many declarative specification languages (such as regular expressions or linear temporal logic) can be translated into DFAs.

The main contribution of this paper is an RL algorithm, named AdvisoRL, that takes advice from the user about the reward machine and uses it to learn more efficiently. AdvisoRL generalizes the prior work: advice can express the full information about the reward machine as well as no information at all. Importantly, the advice is not assumed to be perfect: the algorithm can handle wrong advice.

More precisely, AdvisoRL takes a set of advice DFAs as its input. Interacting with the environment, the agent iteratively learns a reward machine. We develop a novel SAT-based method to learn a reward machine that is *compatible* with the advice and *consistent* with the observed rewards. This method is integrated with the standard Q-learning algorithm. The advice reduces the space of all possible reward machines consistent with the observations, thus speeding-up the convergence. If advice is given that is incompatible with the reward machine underlying the MDP, AdvisoRL discards that advice and recovers the learning process.

AdvisoRL guarantees to converge to an optimal policy for sufficiently long episodes (where the episode length depends on the reward function, advice and the underlying MDP). In Section 5, we show that AdvisoRL successfully uses the provided advice to reduce the number of steps needed for convergence to optimal policy and that it outperforms existing methods for incorporating high-level knowledge.

Related work The problem of incorporating high-level temporal knowledge into RL has been studied in hierarchical RL (Sutton, Precup, and Singh 1999; Dietterich 2000; Parr and Russell 1998). There, an RL problem is decomposed into a hierarchy of subtasks and the agent learns on two levels: a *meta-controller* decides on a subtask to pursue, and a *controller* decides on actions within the chosen subtask.

The idea of capturing temporal abstraction in a temporal logic or automaton-like form appeared already in the work of Bacchus, Boutilier, and Grove (1996), but an interest in it resurged with the introduction of task monitors (Jothimurugan, Alur, and Bastani 2019) and reward machines (Icarte et al. 2018b). They are assumed to be given to the RL agent.

A number of papers followed that suggested learning an automaton-like representation from experience instead of assuming that the user provides them (Icarte et al. 2019; Gaon and Brafman 2020; Xu et al. 2020; Furelos-Blanco et al. 2020; Hasanbeig et al. 2019). In this paper, we assume that the user is able to provide some knowledge, and we incorporate that knowledge into learning of reward machines.

The problem statements in aforementioned papers differ slightly, but in essence they are solving the same problem: learning a non-Markovian structure of the underlying MDP. This can either be a non-Markovian reward function, a non-Markovian environment dynamics, or partial observability that creates an impression of a non-Markovian dynamics. In our work we focus on a non-Markovian reward function, but a similar idea may be brought to bear for other problem formulations as well.

The notion of advice for RL has been explored in the work by Icarte et al. (2018a). There, an advice has the form of a temporal logic formula, which can be compiled into a DFA. They assume the availability of *background knowledge functions*, heuristic functions that help the agent follow the advice. No such assumption is necessary in our work. Focusing more on safety than on helping the learning process, the work on *shields* for RL (Alshiekh et al. 2018) corrects the agent behavior if it violates the temporal logic specification.

2 Preliminaries

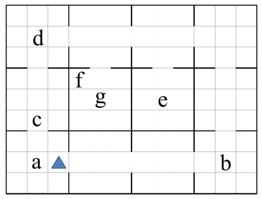
In this section we define our notion of advice and discuss the reasons for choosing the particular formalization. Furthermore, we introduce necessary background on reinforcement learning and reward machines. All the notions will be illustrated using the running example of office world (Figure 1a), adopted from the paper by Xu et al. (2020). The agent (its position denoted by a triangle symbol) has to first go to location b, then d, and finally to c, and receives a reward of 100 upon doing so. If it gets to c without going to b and d first, then it receives a reward of -50. Finally, each time the agent visits g, it gets a reward of -10.

Decision Processes We begin by defining Markov decision processes (MDPs), which model RL problems by incorporating sequential decision making with effects on received rewards and subsequent environment states. Our definition differs from the "usual" modeling used in reinforcement learning (e.g., (Sutton and Barto 2018)) in two ways: first, the reward function is defined over the whole history (i.e., the reward is *non-Markovian*, while the dynamics remains Markovian). Second, we include a set of propositions: they come from expert knowledge of what is relevant for successfully executing a task and are assumed to be available to the agent.

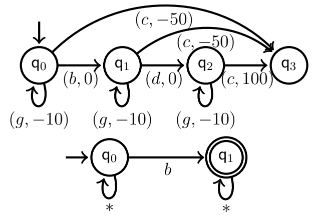
Definition 1. A *labeled Markov decision process (MDP)* is a tuple $\mathcal{M} = (S, s_I, A, p, R, \gamma, \mathcal{P}, L)$ consisting of a finite state space S, an agent's initial state $s_I \in S$, a finite set of actions A, and a probabilistic transition function $p\colon S \times A \times S \to [0, 1]$. A reward function $R : (S \times A)^+ \times S \to \mathbb{R}$ and a discount factor $\gamma \in [0, 1)$ together specify the overall payoff to the agent. Finally, a finite set $\mathcal{P}$ of propositional variables, and a labeling function $L : S \times A \times S \to 2^{\mathcal{P}}$ determine the set of relevant high-level events that the agent senses in the environment. We define the size of $\mathcal{M}$, denoted as $|\mathcal{M}|$, to be $|S|$ (i.e., the cardinality of the set S).

In our example, the state is the agent's position in the grid and the actions available to the agent are moving in the four cardinal directions. The transition function captures the small probability of slipping. The propositions are $\mathcal{P} = \{a, b, c, d, e, f, g\}$, and the labeling function L applied to the triple (s, a, s') returns a set of propositions at the state s'. Note, however, that the full state space is still necessary in order to capture the model's dynamics.

A *policy* is a function mapping states in S to a probability distribution over actions in A. At state $s \in S$,



(a) The office world



(b) Advice (bottom) compatible with the reward machine (top)

Figure 1: Running example

an agent using policy π picks an action a with probability $\pi(s, a)$, and the new state s' is chosen with probability $p(s, a, s')$. A policy π and the initial state s_I together determine a stochastic process. A *trajectory* is a realization of this stochastic process: a sequence of states and actions $s_0a_1s_1 \dots a_ks_k$, with $s_0 = s_I$. Its corresponding *label sequence* is $\ell_1\ell_2 \dots \ell_k$ where $L(s_i, a_{i+1}, s_{i+1}) = \ell_{i+1}$ for each $0 \le i < k$. Similarly, the corresponding *reward sequence* is $r_1r_2 \dots r_k$, where $r_i = R(s_0a_1s_1 \dots a_is_i)$, for each $i \le k$. The overall payoff of the agent is $\sum_i \gamma^i r_i$. We call the pair $(\lambda, \rho) := (\ell_1\ell_2 \dots \ell_k, r_1r_2 \dots r_k)$ a *trace*.

Reward machines Reward machines (Icarte et al. 2018b) are a way to encode a non-Markovian reward function. They are an automaton that upon reading a label, responds with a reward and moves to its next state. Technically, a reward machine is an instance of a Mealy machine (Shallit 2008), with a set of real numbers as its output alphabet and subsets of propositional variables (from the set $\mathcal{P}$ defined by the underlying MDP) as its input alphabet.

Definition 2. A *reward machine* $\mathsf{A} = (V, \mathsf{q}_I, 2^{\mathcal{P}}, M, \delta, \sigma)$ is defined by a finite, nonempty set V of states, an initial state $\mathsf{q}_I \in V$, an input alphabet $2^{\mathcal{P}}$, an output alphabet $M \subset \mathbb{R}$, a (deterministic) transition function $\delta\colon V \times 2^{\mathcal{P}} \to V$, and an output function $\sigma\colon V \times 2^{\mathcal{P}} \to M$. We define the size of A, denoted as $|\mathsf{A}|$, to be $|V|$ (i.e., the cardinality of the set V).

A run of a reward machine A on the sequence of labels $\ell_1\ell_2 \dots \ell_k \in (2^{\mathcal{P}})^*$ is a sequence $\mathsf{q}_0(\ell_1, r_1)\mathsf{q}_1(\ell_2, r_2) \dots \mathsf{q}_{k-1}(\ell_k, r_k)\mathsf{q}_k$ of states and label-reward pairs such that $\mathsf{q}_0 = \mathsf{q}_I$ and for all $i \in \{0, \dots, k-1\}$, we have $\delta(\mathsf{q}_i, \ell_{i+1}) = \mathsf{q}_{i+1}$ and $\sigma(\mathsf{q}_i, \ell_{i+1}) = r_{i+1}$. We write $\mathsf{A}(\ell_1\ell_2 \dots \ell_k) = r_1r_2 \dots r_k$ to connect the input label sequence to the sequence of rewards produced by the machine A. We say that a reward machine A *encodes* the reward function R of an MDP if for every trajectory $s_0a_1s_1 \dots a_ks_k$ and the corresponding label sequence $\ell_1\ell_2 \dots \ell_k$, the reward sequence that the agent receives equals $\mathsf{A}(\ell_1\ell_2 \dots \ell_k)$.

A reward machine for the motivating example is shown in the upper part of Figure 1b. We note here that there can be multiple reward machines that encode the reward function of an MDP—reward machines may differ on a label sequence that does not correspond to any trajectory of an underlying MDP.

3 Advice

We use *deterministic finite automata (DFAs)* to model advice from the user. Formally, a DFA is a tuple $\mathfrak{D} = (\mathsf{Q}, \mathsf{q}_\mathsf{I}, \Sigma, \delta, \mathsf{F})$ consisting of a nonempty, finite set Q of states, an initial state $\mathsf{q}_\mathsf{I} \in \mathsf{Q}$, an input alphabet Σ (here: $\Sigma = 2^{\mathcal{P}}$), a transition function $\delta\colon \mathsf{Q} \times \Sigma \to \mathsf{Q}$, and set $\mathsf{F} \subseteq \mathsf{Q}$ of final states. A run of $\mathfrak{D}$ on a sequence $u = a_1 \dots a_n \in \Sigma^*$ is a sequence $\mathsf{q}_0, \dots, \mathsf{q}_n$ of states such that $\mathsf{q}_0 = \mathsf{q}_\mathsf{I}$ and $\mathsf{q}_i = \delta(\mathsf{q}_{i-1}, a_i)$ for $i \in \{1, \dots, n\}$. A sequence u is accepted if the run of $\mathfrak{D}$ on u ends in a state in F, and $L(\mathfrak{D})$, the language of $\mathfrak{D}$, is the set of all accepted sequences.

The core idea of an advice DFA is to guide the inference of reward machines by providing information about which label sequences might result in a reward and which cannot result in a reward. More precisely, the meaning of an advice DFA $\mathfrak{D}$ is that all label sequence $\lambda \in L(\mathfrak{D})$ *can* (but not must) obtain a *positive reward*, whereas all label sequences $\lambda \notin L(\mathfrak{D})$ *must not* receive a positive reward. Hence, an advice DFA acts as a binary classifier indicating which explorations are promising and which are not.

In the RL algorithm we develop in this paper, advice DFAs are used to restrict the search space in which to search for the true reward machine (i.e., the one defining the reward function R of the MDP). We do so by only constructing candidate reward machines that are *compatible* with the advice DFAs given by the user, as defined below. This makes it possible to learn the correct machine with fewer examples and speeds up the overall convergence of our RL algorithm.

Definition 3. We call an advice DFA $\mathfrak{D}$ *compatible* with a reward machine A if for all nonempty label sequences $\ell_1\ell_2 \dots \ell_k \in (2^{\mathcal{P}})^+$ with $\mathsf{A}(\ell_1\ell_2 \dots \ell_k) = r_1r_2 \dots r_k$ it holds that $r_k > 0$ implies $\ell_1\ell_2 \dots \ell_k \in L(\mathfrak{D})$.

We have chosen DFAs as the means of providing advice mainly for two reasons. First, DFAs are a simple formalism, familiar to many engineers and data scientists, that admit effective translations from other common formalisms, such as regular expressions or Linear Temporal Logic. Second, DFAs have a simple, binary semantics, which does not permit to express different reward values for different behaviors. This reflects the observation that it is less demanding for humans to give advice suggesting "what must not be done" than providing the exact, quantitative value.

An example of an advice compatible with a reward machine is shown in the bottom part of Figure 1b. Finally, note that our definitions so far only permit advice for positive rewards. This is not a restriction since we can simply introduce a second "type" of advice DFA for negative rewards. For the sake of a simpler presentation, however, the remainder of this paper focuses on advice DFAs for positive rewards only.

4 Reinforcement Learning with Advice

Our advice-guided reinforcement learning algorithm builds on top of the *JIRP* algorithm by Xu et al. (2020) for the joint inference of reward machines and policies in reinforcement learning. At its heart lies a subroutine, called *QRM* (Icarte et al. 2018b), which takes a reward machine and a set of q-functions (one for each state of the reward machine) as input

Algorithm 1: The AdvisoRL algorithm

```
Input: A set $D = \{\mathfrak{D}_1, \ldots, \mathfrak{D}_\ell\}$ of advice DFAs
1  Initialize an empty sample $X \leftarrow \emptyset$
2  Initialize a reward machine $\mathsf{A}$ with state set $V$ that is
     compatible with $D$
3  Initialize a set of $q$-functions $Q = \{q^{\mathsf{q}} \mid \mathsf{q} \in V\}$
4  for episodes $i = 1, 2, \ldots$ do
5      $(\lambda, \rho, Q) \leftarrow$ QRM-episode($\mathsf{A}, Q$)
6      if $\mathsf{A}(\lambda) \neq \rho$ then
7          Add $(\lambda, \rho)$ to $X$
8      if $(\lambda, \rho)$ is not compatible with a DFA in $D$ then
9          Remove all non-compatible DFAs from $D$
10     if $X$ or $D$ have changed then
11         Infer a new minimal reward machine $\mathsf{A}$ that is
             consistent with $X$ and compatible with $D$
12         Re-initialize $Q$ (or transfer $q$-functions)
13 end
```

and performs q-learning on the product of the reward machine and the underlying MDP. The defining difference of JIRP to our algorithm, *AdvisoRL*, is that JIRP cannot handle advice from the user. However, we retain the desirable property that AdvisoRL—like JIRP—almost surely converges to the optimal policy in the limit.

AdvisoRL is shown as Algorithm 1. It maintains a hypothesis reward machine A and runs the QRM algorithm to learn an optimal policy (wrt. A). The episodes of QRM are used to collect traces and update q-functions. As long as the traces are consistent with the current hypothesis reward machine A and compatible with each advice DFAs in D, QRM interacts with the environment using A to guide the learning process.

If a trace (λ, ρ) is encountered that is inconsistent with the hypothesis reward machine (i.e., $\mathsf{A}(\lambda) \neq \rho$), our algorithm records it in a set X (Lines 6 and 7)—we call the trace (λ, ρ) a *counterexample* and the set X a *sample*. Similarly, if there exists an advice DFA $\mathfrak{D} \in D$ that is not compatible with the trace (λ, ρ), we remove $\mathfrak{D}$ from D (Lines 8 and 9). This is necessary because the (λ, ρ) is an actual trace experienced by the agent, showing that the advice was incorrect.

Every time the sample is updated or an advice is removed, AdvisoRL infers a new minimal reward machine A' (Line 11) that is (a) *consistent* with the sample in the sense that $\mathsf{A}'(\lambda) = \rho$ holds for all $(\lambda, \rho) \in X$ and (b) compatible with each advice DFA in D. Note that AdvisoRL infers a *minimal* consistent and compatible reward machine (i.e., one with the fewest number of states among all consistent and compatible reward machines). This additional requirement can be seen as an Occam's razor strategy (Löding, Madhusudan, and Neider 2016) and is crucial in that it enables AdvisoRL to converge to the optimal policy in the limit.

Also note that we re-use all of JIRP's algorithmic optimizations, such as transfer of q-functions from one reward machine to the next and batching of counterexamples, but we omit their descriptions here.

Inferring Reward Machines with Advice

In this section, we describe how AdvisoRL infers (minimal) reward machines from counterexamples while being guided by the advice DFAs. To this end, we assume that the inference algorithm maintains a finite sample $X \subset (2^{\mathcal{P}})^+ \times \mathbb{R}^+$ of traces as well as a set $D = \{\mathfrak{D}_1, \ldots \mathfrak{D}_\ell\}$ of advice DFAs. In the following, we assume that each advice DFA is compatible with the sample X in the sense that for each $(l_1 \ldots l_k, r_1 \ldots r_k) \in X$ with $r_k > 0$ we also have $l_1 \ldots l_k \in L(\mathfrak{D})$. If this is not satisfied, the corresponding advice DFA gets removed from D.

The task of the inference algorithm is to find a *minimal* reward machine A that is (a) consistent with X and (b) compatible with each $\mathfrak{D} \in D$. Note that the requirement to infer a minimal reward machine is crucial for the convergence of AdvisoRL to an optimal policy.

To solve this task, we follow a recent approach that uses SAT-based automata learning to verify parametric systems (Neider 2012; Neider and Jansen 2013; Neider 2014). The underlying idea is to reduce the inference task to a series of satisfiability checks of formulas in propositional logic. More precisely, we construct and solve a sequence of propositional formulas $\Phi_n^{X,D}$ for increasing values of $n > 0$ that have the following two properties:

- $\Phi_n^{X,D}$ is satisfiable if and only if there exists a reward machine with n states that is consistent with $\mathcal{X}$ and compatible with each $\mathfrak{D} \in D$; and
- if $\Phi_n^{X,D}$ is satisfiable, then a satisfying assignment contains sufficient information to construct a consistent and compatible reward machine with n states.

By starting with $n = 1$ and increasing n until $\Phi_n^{X,D}$ becomes satisfiable, we obtain an effective algorithm to compute a minimal reward machine that is consistent with X and compatible with D.

Before we show how to construct the formula $\Phi_n^{X,D}$ in the remainder, let us briefly introduce the necessary notation. We use the standard syntax for propositional logic: starting with a set of propositional variables $X = \{x, y, z, \ldots\}$, we construct propositional formulas Φ inductively using the Boolean connectives $\neg$, $\wedge$, $\vee$, and $\rightarrow$. An *interpretation* is a mapping $\mathcal{I} \colon X \rightarrow \{0, 1\}$ that assigns a Boolean value to each variable. Satisfaction of a propositional formula is defined in the usual way, and we write $\mathcal{I} \models \Phi$ to denote that the interpretation $\mathcal{I}$ *satisfies* the formula Φ.

Finally, given a sample X, let $R_X \subset \mathbb{R}$ denote the *finite* set of rewards that appear in a trace $\tau \in X$; we use this set as the output alphabet of our reward machine. Additionally, for a trace $\tau = (l_1 \ldots l_k, r_1 \ldots r_k) \in (2^{\mathcal{P}})^* \times \mathbb{R}^*$, we define the set of *prefixes of* τ by $\mathit{Pref}(\tau) = \{(l_1 \ldots l_i, r_1 \ldots r_i) \in (2^{\mathcal{P}})^* \times \mathbb{R}^* \mid 0 \leq i \leq k\}$ (note that $(\varepsilon, \varepsilon) \in \mathit{Pref}(\tau)$ always holds). We lift this notion to samples $X \subset (2^{\mathcal{P}})^* \times \mathbb{R}^*$ by $\mathit{Pref}(X) = \bigcup_{\tau \in X} \mathit{Pref}(\tau)$.

Encoding Reward Machines in Propositional Logic The encoding of reward machines in propositional logic exploits that once a set V of states and an initial state $\mathsf{q}_I \in V$ is fixed, every reward machine $\mathsf{A} = (V, \mathsf{q}_I, 2^{\mathcal{P}}, R_X, \delta, \sigma)$ is uniquely determined by its transition function δ and its output function σ. Hence, let us fix a set V of states with $|V| = n$ and an initial state $\mathsf{q}_I \in V$.

To encode the transition function and the set of final states, we introduce two propositional variables: $d_{\mathsf{p},l,\mathsf{q}}$ for $\mathsf{p}, \mathsf{q} \in V$ and $l \in 2^{\mathcal{P}}$; and $o_{\mathsf{p},l,r}$ for $\mathsf{p} \in V$, $l \in 2^{\mathcal{P}}$, and $r \in R_X$. Intuitively, the variable $d_{\mathsf{p},l,\mathsf{q}}$ is set to true if and only if the transition $\delta(\mathsf{p}, l) = \mathsf{q}$ exists in the prospective reward machine, while $o_{\mathsf{p},l,r}$ is set to true if and only if $\sigma(\mathsf{p}, l) = r$.

To ensure that the variables $d_{\mathsf{p},l,\mathsf{q}}$ and $o_{\mathsf{p},l,r}$ indeed encode deterministic functions, we add the following constraints:

$$\bigwedge_{\mathsf{p}\in V}\bigwedge_{l\in 2^{\mathcal{P}}}\Big[\big[\bigvee_{\mathsf{q}\in V} d_{\mathsf{p},l,\mathsf{q}}\big]\wedge\big[\bigwedge_{\mathsf{q}\neq\mathsf{q}'\in V}\neg d_{\mathsf{p},l,\mathsf{q}}\vee\neg d_{\mathsf{p},l,\mathsf{q}'}\big]\Big] \tag{1}$$

$$\bigwedge_{\mathsf{p}\in V}\bigwedge_{l\in 2^{\mathcal{P}}}\Big[\big[\bigvee_{r\in R_X} o_{\mathsf{p},l,r}\big]\wedge\big[\bigwedge_{r\neq r'\in R_X}\neg o_{\mathsf{p},l,r}\vee\neg o_{\mathsf{p},l,r'}\big]\Big] \tag{2}$$

Note that Formula (1) ensures that for every state $\mathsf{p} \in V$ and symbol $l \in 2^{\mathcal{P}}$ the variable $d_{\mathsf{p},l,\mathsf{q}}$ is set to true for exactly one $\mathsf{q} \in V$, whereas Formula (2) ensures that for every state $\mathsf{p} \in V$ and symbol $l \in 2^{\mathcal{P}}$ the variable $o_{\mathsf{p},l,r}$ is set to true for exactly one $r \in R_X$. For the remainder, let Φ^{RM} be the conjunction of Formulas (1) and (2).

Given a satisfying assignment $\mathcal{I} \models \Phi^{RM}_n$, we can derive a reward machine $\mathsf{A}_{\mathcal{I}} = (V, \mathsf{q_I}, 2^{\mathcal{P}}, R_X, \delta, \sigma)$ by $\delta(\mathsf{p}, l) = \mathsf{q}$ if and only $\mathcal{I}(d_{\mathsf{p},l,\mathsf{q}}) = 1$, and $\sigma(\mathsf{p}, l) = r$ if and only if $\mathcal{I}(o_{\mathsf{p},l,r}) = 1$. Note that the reward machine $\mathsf{A}_{\mathcal{I}}$ is well-defined due to the fact that $\mathcal{I} \models \Phi^{RM}_n$. However, $\mathsf{A}_{\mathcal{I}}$ is not (yet) related to the sample X or the set D of advice DFAs.

Consistency with Sample To encode consistency with a sample in propositional logic, we introduce new auxiliary variables $x_{\lambda,\mathsf{q}}$ for $(\lambda, \rho) \in \mathit{Pref}(X)$ and $\mathsf{q} \in V$. Intuitively, these variables capture the run of the prospective reward machine on (prefixes of) label sequences in X in the sense that $x_{\lambda,\mathsf{q}}$ is set to true if and only if the prospective reward machine reaches states q after reading λ. To obtain the desired meaning, we add the following constraints:

$$x_{\varepsilon,\mathsf{q_I}} \wedge \bigwedge_{\mathsf{p}\in V\setminus\{\mathsf{q_I}\}} \neg x_{\varepsilon,\mathsf{p}} \tag{3}$$

$$\bigwedge_{(\lambda l,\rho r)\in \mathit{Pref}(X)}\ \bigwedge_{\mathsf{p},\mathsf{q}\in V} (x_{\lambda,p} \wedge d_{\mathsf{p},l,\mathsf{q}}) \rightarrow x_{\lambda l,\mathsf{q}} \tag{4}$$

$$\bigwedge_{(\lambda l,\rho r)\in \mathit{Pref}(X)}\ \bigwedge_{\mathsf{p}\in V} x_{\lambda,p} \rightarrow o_{\mathsf{p},l,\mathsf{r}} \tag{5}$$

Intuitively, Formula (3) ensures that the reward machine starts in the initial state $\mathsf{q_I}$, while Formula (4) ensures that the variables $x_{\lambda,\mathsf{q}}$ encode valid runs of the prospective reward machine on prefixes of label sequences in X. Formula (5) enforces that the prospective reward machine outputs the correct rewards. For reference, we denote the conjunction of Formulas (3), (4), and (5) by Φ^X_n.

Consistency with Advice DFAs Let us now fix an advice DFA $\mathfrak{D} \in D$, say $\mathfrak{D} = (\mathsf{Q}_{\mathfrak{D}}, \mathsf{q}_{\mathsf{I},\mathfrak{D}}, 2^{\mathcal{P}}, \delta_{\mathfrak{D}}, \mathsf{F}_{\mathfrak{D}})$. We now show how to constrain the variables $d_{\mathsf{p},l,\mathsf{q}}$ and $o_{\mathsf{q},l,r}$ so that the prospective reward machine is compatible with $\mathfrak{D}$. We can then successively add similar constraints until the prospective reward machine is compatible with all advice DFAs from D.

The key idea of our encoding is to track the synchronized runs of the prospective reward machine $\mathsf{A}_{\mathcal{I}}$ and the advice DFA $\mathfrak{D}$. To this end, we introduce the following new auxiliary variables $y^{\mathfrak{D}}_{\mathsf{q},\mathsf{q}'}$ for $\mathsf{q} \in V$ and $\mathsf{q}' \in \mathsf{Q}_{\mathfrak{D}}$. Intuitively, $y^{\mathfrak{D}}_{\mathsf{q},\mathsf{q}'}$ is set to true if there exists a label sequence λ such that $\mathsf{A}_{\mathcal{I}} \colon \mathsf{q_I} \xrightarrow{\lambda} \mathsf{q}$ and $\mathfrak{D} \colon \mathsf{q}_{\mathsf{I},\mathfrak{D}} \xrightarrow{\lambda} \mathsf{q}'$.

To obtain the desired meaning, we add the following constraints:

$$y^{\mathfrak{D}}_{\mathsf{q_I},\mathsf{q}'_{\mathsf{I},\mathfrak{D}}} \tag{6}$$

$$\bigwedge_{\mathsf{p},\mathsf{q}\in V}\ \bigwedge_{l\in 2^{\mathcal{P}}}\ \bigwedge_{\delta_{\mathfrak{D}}(\mathsf{p}',l)=\mathsf{q}'} (y^{\mathfrak{D}}_{\mathsf{p},\mathsf{p}'} \wedge d_{\mathsf{p},l,\mathsf{q}}) \rightarrow y^{\mathfrak{D}}_{\mathsf{q},\mathsf{q}'} \tag{7}$$

$$\bigwedge_{\mathsf{p}\in V}\ \bigwedge_{\substack{\delta_{\mathfrak{D}}(\mathsf{p}',l)=\mathsf{q}'\\ \mathsf{q}'\notin\mathsf{F}_{\mathfrak{D}}}} y^{\mathfrak{D}}_{\mathsf{p},\mathsf{p}'} \rightarrow \neg \bigvee_{\substack{r\in R_X\\ r>0}} o_{\mathsf{p},l,r} \tag{8}$$

Note that Formula (6) ensures that the synchronized runs of the prospective reward machine $\mathsf{A}_{\mathcal{I}}$ and the advice DFA $\mathfrak{D}$ start in their initial states, while Formula (7) enforces that the variables $y^{\mathfrak{D}}_{\mathsf{q},\mathsf{q}'}$ correctly track the synchronized runs. Moreover, Formula (8) ensures that $\mathsf{A}_{\mathcal{I}}$ is compatible with $\mathfrak{D}$ by contraposition: if $\mathsf{A}_{\mathcal{I}}$ has moved to state p after reading a label sequence λ and is now processing a new label l but $\lambda l \notin L(\mathfrak{D})$ (indicated by $\mathfrak{D}$ reaching a non-final state $\mathsf{q}' \notin F_{\mathfrak{D}}$), then the output must not be positive. We denote the conjunction of Formulas (6), (7), and (8) by $\Phi^{\mathfrak{D}}_n$.

Finally, the formula $\Phi^{X,D}_n$ is given by $\Phi^{X,D}_n := \Phi^{RM}_n \wedge \Phi^X_n \wedge \bigwedge_{\mathfrak{D}\in D} \Phi^{\mathfrak{D}}_n$. With this, we obtain the following result.

Theorem 1. *Let $X \subset (2^{\mathcal{P}})^+ \times \mathbb{R}^+$ be sample and D a finite set of advice DFAs that are compatible with X. Moreover, let $\Phi^{X,D}_n$ be as defined above. Then, the following holds:*

1. *If $\mathcal{I} \models \Phi^{X,D}_n$, then the reward machine $\mathsf{A}_{\mathcal{I}}$ is consistent with X and compatible with each $\mathfrak{D} \in D$.*
2. *If there exists a reward machine with n states that is consistent with X and compatible with each $\mathfrak{D} \in D$, then $\Phi^{X,D}_n$ is satisfiable.*

Convergence to an Optimal Policy

We now show that AdvisoRL is guaranteed to converge to an optimal policy, given that it explores episode of sufficient length. However, two complications arise from the fact that some of the label sequences might not be *admissible* in the sense that the underlying MDP does not permit these label sequences (e.g., they are physically impossible). If this is the case, there can in general be multiple (even infinitely many) reward machines that are *equivalent* to the true underlying reward machine (i.e., that agree on the admissible traces but might differ on the traces that are not admissible). Moreover, the notion of compatibility needs to be restricted to the admissible label sequences. Nonetheless, AdvisoRL guarantees to learn an equivalent reward machine.

Lemma 1. *Let $\mathcal{M}$ be a labeled MDP, let A be the true reward machine encoding the rewards of $\mathcal{M}$, and* *let $D = \{\mathfrak{D}_1, \ldots, \mathfrak{D}_\ell\}$ be a set of advice DFAs. Moreover, let $n_{max} = \max_{1 \leq i \leq \ell} \{|\mathfrak{D}_i|\}$ and $m = \max\left\{2|\mathcal{M}| \cdot (|\mathsf{A}|+1) \cdot n_{max}, |\mathcal{M}|(|\mathsf{A}|+1)^2\right\}$. Then, AdvisoRL with eplen $\geq m$ almost surely learns a reward machine in the limit that is equivalent to* A.

The correctness of AdvisoRL now follows from Lemma 1 and the correctness of the QRM algorithm (Icarte et al. 2018b). Lemma 1 also provides us with an upper bound on the length of the episodes that AdvisoRL has to explore.

Theorem 2. *Let $\mathcal{M}$ be a labeled MDP, A the true reward machine encoding the rewards of $\mathcal{M}$, and $D = \{\mathfrak{D}_1, \ldots, \mathfrak{D}_\ell\}$ a set of advice DFAs. Moreover, let m be as in Lemma 1. Then, AdvisoRL with eplen $\geq m$ almost surely converges to an optimal policy in the limit.*

It is worth mentioning that if all label sequences are attainable, the upper bound on episode length reduces to $\max\left\{|\mathcal{M}|, 2(|\mathsf{A}|+1) \cdot n_{\max}\right\}$.

5 Case Studies

In this section, we implement the AdvisoRL approach in two case studies. We compare the three different methods:[1]

1. AdvisoRL: Our implementation uses the RC2 SAT solver (Morgado, Dodaro, and Marques-Silva 2014) from the PySAT library (Ignatiev, Morgado, and Marques-Silva 2018). A special case of AdvisoRL, when no advice is given, is the JIRP-SAT algorithm (Xu et al. 2020).
2. HRL (hierarchical reinforcement learning): We use a meta-controller for deciding the subtasks (represented by encountering each label) and use the low-level controllers expressed by neural networks (Kulkarni et al. 2016) for deciding the actions at each state for each subtask. We note that HRL may never be able to find an optimal policy (and this is the case even if it has access to the full true reward machine, as noted by Icarte et al. (2018b)).
3. DDQN (deep reinforcement learning with double q-learning): We adopt the double q-learning method of Hasselt, Guez, and Silver (2016). The DDQN can access the past 200 labels of the trajectory as well as the MDP state.

Figure 5 illustrates the advices we use with AdvisoRL in the case studies: DFAs expressing "eventually α_0, then eventually α_1..., then eventually α_{n-1}", noted $\{\alpha_0\alpha_1 \cdots \alpha_{n-1}\}$. (There is no restriction on using any other form of advice.)

Case Study I: Officeworld Domain

This case study relies on the officeworld scenario as introduced in Figure 1. We specify a more involved task than that in the motivating example: go to location b, then d, then back to b, and finally go to g. We run the AdvisoRL algorithm with different sets of advice DFAs D.

Impact of advices Figure 2a shows the attained reward of AdvisoRL for 8 sets of advice DFAs: $\emptyset$, $\{b\}$, $\{d\}$, $\{g\}$, $\{b,d,g\}$, $\{bd\}$, $\{bdb\}$, $\{bdbg\}$, and Figure 4a shows the optimal policy convergence training steps. We observe a clear impact of advices on the performance. It can be seen that the closer the advice is to the ground truth, the faster the maximal reward is reached.

[1]All experiments were conducted on a Vivobook laptop with 1.80-GHz Core i7 CPU and 32-GB RAM

advice	$\emptyset$	$\{b\}$	$\{bd\}$	$\{bdb\}$	$\{bdbg\}$
# steps [$\times 10^3$]	454.5	307.5	189.0	144.0	142.5
inference [s]	149.0	13.9	1.7	0.7	0.5
# inferences	6.1	5.0	3.2	2.3	2.0

Table 1: Performance of AdvisoRL, for different advice.

Table 1 shows the performance of AdvisoRL, for different advice DFAs. Measured over 30 independent simulation runs, the first row shows the median number of thousands of training steps to convergence; the second row shows the median time needed reward machines inference; and the third row shows the average number of inferences. With shorter individual inference time and fewer triggers of the reward machine inference, the cumulative inference time drops significantly when using advice, and becomes negligible compared the RL simulation time with $D \in \{\{bd\}, \{bdb\}, \{bdbg\}\}$.

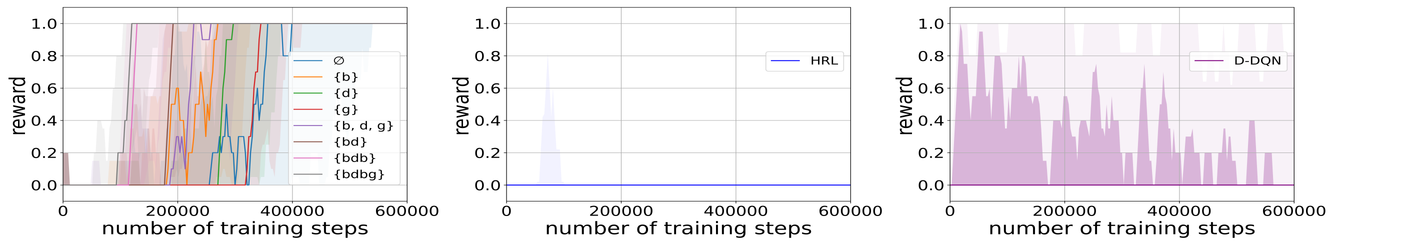
Figure 4b shows the cumulative inference time for $D \in \{\emptyset, \{b\}, \{d\}, \{g\}, \{b,d,g\}, \{bd\}, \{bdb\}, \{bdbg\}\}$. The inference time with $D = \{g\}$ increases probably because providing the end goal without intermediate steps may be counterproductive: we witness that many traces are added to the sample X. However, AdvisoRL still converges to an optimal policy with $D = \{g\}$ in fewer training steps than with $D = \emptyset$. Compared to AdvisoRL with $D = \emptyset$, AdvisoRL with $D = \{b\}$, $D = \{bd\}$ and $D = \{bdbg\}$ achieves optimal convergence in 68%, 42% and 31% of the training steps, and with 9.3%, 1.1% and 0.34% cumulative inference time, respectively.

We also assessed the performance of AdvisoRL in presence of incorrect advice. Our experiments showed that our algorithm is robust in that it quickly eliminated the incorrect advice and converged to an optimal policy.

Case Study II: Taxi Domain

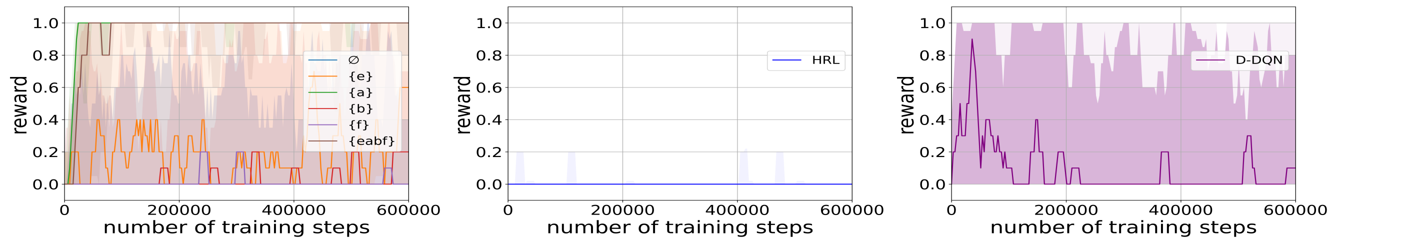
This experiment is inspired by the OpenAI Gym environment Taxi-v3 (https://gym.openai.com/envs/Taxi-v3/), introduced by Dietterich (1999). The agent, a taxi, navigates on a grid with walls and boarding locations.The agent starts on a random cell. A passenger starts on a random boarding location. The set of actions is $A = \{S, N, E, W, \mathit{Pickup}, \mathit{Dropoff}\}$. The action Pickup picks up the passenger present at the agent's current location and has no effect if no passenger is present. The action $\mathit{Dropoff}$ drops off the passenger from the taxi to the agent's current boarding location, and has no effect if the taxi is empty or not over any boarding location. The actions S, N, E, W correspond to moving in the four cardinal directions.

We make some assumptions to simplify the problem: the passenger always starts on location A, and the agent starts on a random cell other than a boarding location. We specify the task as carrying the passenger to location B. We define eight labels: *a*, *b*, *c*, *d* for standing on an empty location (A,



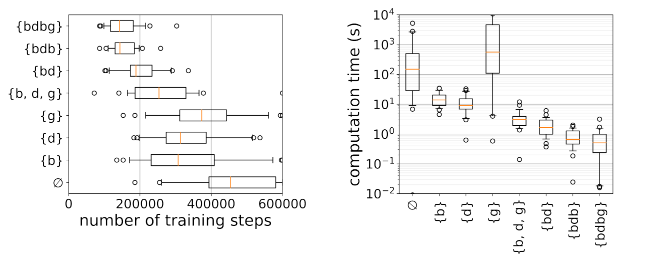
(a) AdvisoRL with advices (b) HRL (c) DDQN

Figure 2: Attained rewards of 30 independent simulation runs averaged for every 10 training steps each for case study I.



(a) AdvisoRL with advices (b) HRL (c) DDQN

Figure 3: Attained rewards of 30 independent simulation runs averaged for every 10 training steps each for case study II.



(a) Training steps for optimal policy convergence.

(b) Cumulative inference time, on a logarithmic scale.

Figure 4: Distribution of 30 independent simulation runs on each set of advice DFAs with AdvisoRL on case study I.

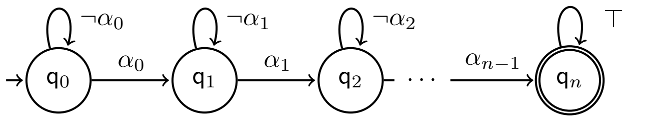
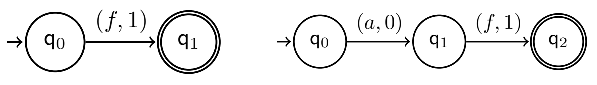


Figure 5: Advice DFA format used in our case study. The short notation for such DFAs would be $\alpha_0\alpha_1\cdots\alpha_{n-1}$.

B, C, D respectively), and e, f, g, h for standing on a location with the passenger on it (A, B, C, D respectively).

Results Figure 3a presents performances by using 6 different set of advice DFAs: $\emptyset$, $\{e\}$, $\{a\}$, $\{b\}$, $\{f\}$, $\{eabf\}$. Optimal reward is reached only with advices $\{a\}$ and $\{eabf\}$.

With $D = \emptyset$, only one reward machine is inferred, preventing counterexamples to be registered and new inference to be triggered. This is because the label f (having the pas-



(a) $D = \emptyset$ (b) $D = \{a\}$

Figure 6: The inferred reward machine in all runs of the taxi domain. Omitted transitions are self-looping transitions.

senger on B) is in the sequence if and only if the agent succeeds in the task. Hence the reward machine inferred without advice (Figure 6a) is the smallest possible that will correctly classify every attainable label sequence.

The label a, being triggered when the passenger is picked up from A, is helpful to AdvisoRL because it helps the agent track the passenger's location. The advice a lets the reward machine include this label (Figure 6b).

This case study showcases the limitations of the algorithm without advice (which corresponds to the JIRP algorithm from (Xu et al. 2020)) when the reward function is Markovian while the MDP dynamics is non-Markovian. The minimal reward machine that it infers is of no help to improve the RL performance. On the other hand, well-chosen advices can help the solver infer a helpful reward machine.

6 Conclusion

We have presented AdvisoRL, a novel RL algorithm that is guided by high-level human advice. In our evaluation, we have demonstrated that AdvisoRL outperforms the existing techniques and is robust to incorrect advice.

For future work, we will extend AdvisoRL to different kinds of advice (e.g., mandating a positive reward to be given for certain label sequences).

Acknowledgements

This work was partially supported by the grants ARL W911NF2020132, ONR N00014-20-1-2115, and ARO W911NF-20-1-0140.

References

Alshiekh, M.; Bloem, R.; Ehlers, R.; Könighofer, B.; Niekum, S.; and Topcu, U. 2018. Safe Reinforcement Learning via Shielding. In *AAAI*, 2669–2678. AAAI Press.

Bacchus, F.; Boutilier, C.; and Grove, A. J. 1996. Rewarding Behaviors. In *AAAI/IAAI, Vol. 2*, 1160–1167. AAAI Press / The MIT Press.

Brafman, R. I.; Giacomo, G. D.; and Patrizi, F. 2018. LTLf/LDLf Non-Markovian Rewards. In *AAAI*, 1771–1778. AAAI Press.

Camacho, A.; Icarte, R. T.; Klassen, T. Q.; Valenzano, R. A.; and McIlraith, S. A. 2019. LTL and Beyond: Formal Languages for Reward Function Specification in Reinforcement Learning. In *IJCAI*, 6065–6073. ijcai.org.

Dietterich, T. G. 1999. Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition. *CoRR* cs.LG/9905014. URL https://arxiv.org/abs/cs/9905014.

Dietterich, T. G. 2000. Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition. *J. Artif. Intell. Res.* 13: 227–303.

Furelos-Blanco, D.; Law, M.; Russo, A.; Broda, K.; and Jonsson, A. 2020. Induction of Subgoal Automata for Reinforcement Learning. In *AAAI*, 3890–3897. AAAI Press.

Gaon, M.; and Brafman, R. I. 2020. Reinforcement Learning with Non-Markovian Rewards. In *AAAI*, 3980–3987. AAAI Press.

Hasanbeig, M.; Jeppu, N. Y.; Abate, A.; Melham, T.; and Kroening, D. 2019. DeepSynth: Program Synthesis for Automatic Task Segmentation in Deep Reinforcement Learning. *CoRR* abs/1911.10244.

Hasselt, H. v.; Guez, A.; and Silver, D. 2016. Deep Reinforcement Learning with Double Q-Learning. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, 2094–2100. AAAI Press. URL http://dl.acm.org/citation.cfm?id=3016100.3016191.

Icarte, R. T.; Klassen, T. Q.; Valenzano, R. A.; and McIlraith, S. A. 2018a. Advice-Based Exploration in Model-Based Reinforcement Learning. In *Canadian Conference on AI*, volume 10832 of *Lecture Notes in Computer Science*, 72–83. Springer.

Icarte, R. T.; Klassen, T. Q.; Valenzano, R. A.; and McIlraith, S. A. 2018b. Using Reward Machines for High-Level Task Specification and Decomposition in Reinforcement Learning. In *ICML*, volume 80 of *Proceedings of Machine Learning Research*, 2112–2121. PMLR.

Icarte, R. T.; Waldie, E.; Klassen, T. Q.; Valenzano, R. A.; Castro, M. P.; and McIlraith, S. A. 2019. Learning Reward Machines for Partially Observable Reinforcement Learning. In *NeurIPS*, 15497–15508.

Ignatiev, A.; Morgado, A.; and Marques-Silva, J. 2018. PySAT: A Python Toolkit for Prototyping with SAT Oracles. In *SAT*, 428–437. doi:10.1007/978-3-319-94144-8_26. URL https://doi.org/10.1007/978-3-319-94144-8_26.

Jothimurugan, K.; Alur, R.; and Bastani, O. 2019. A Composable Specification Language for Reinforcement Learning Tasks. In *NeurIPS*, 13021–13030.

Kulkarni, T. D.; Narasimhan, K.; Saeedi, A.; and Tenenbaum, J. 2016. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In *NeurIPS'2016*, 3675–3683.

Löding, C.; Madhusudan, P.; and Neider, D. 2016. Abstract Learning Frameworks for Synthesis. In *TACAS*, volume 9636 of *Lecture Notes in Computer Science*, 167–185. Springer.

Morgado, A.; Dodaro, C.; and Marques-Silva, J. 2014. Core-Guided MaxSAT with Soft Cardinality Constraints. In O'Sullivan, B., ed., *Principles and Practice of Constraint Programming - 20th International Conference, CP 2014, Lyon, France, September 8-12, 2014. Proceedings*, volume 8656 of *Lecture Notes in Computer Science*, 564–573. Springer. doi:10.1007/978-3-319-10428-7_41. URL https://doi.org/10.1007/978-3-319-10428-7_41.

Neider, D. 2012. Computing Minimal Separating DFAs and Regular Invariants Using SAT and SMT Solvers. In *ATVA*, volume 7561 of *Lecture Notes in Computer Science*, 354–369. Springer.

Neider, D. 2014. *Applications of automata learning in verification and synthesis*. Ph.D. thesis, RWTH Aachen University. URL http://darwin.bth.rwth-aachen.de/opus3/volltexte/2014/5169.

Neider, D.; and Jansen, N. 2013. Regular Model Checking Using Solver Technologies and Automata Learning. In *NASA Formal Methods*, volume 7871 of *Lecture Notes in Computer Science*, 16–31. Springer.

Parr, R.; and Russell, S. J. 1998. Reinforcement learning with hierarchies of machines. In *Advances in neural information processing systems*, 1043–1049.

Shallit, J. O. 2008. *A Second Course in Formal Languages and Automata Theory*. Cambridge University Press. ISBN 978-0-521-86572-2. URL http://www.cambridge.org/gb/knowledge/isbn/item1173872/?site_locale=en_GB.

Sutton, R. S.; and Barto, A. G. 2018. *Reinforcement learning: An introduction*. MIT press.

Sutton, R. S.; Precup, D.; and Singh, S. P. 1999. Between MDPs and Semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning. *Artif. Intell.* 112(1-2): 181–211.

Xu, Z.; Gavran, I.; Ahmad, Y.; Majumdar, R.; Neider, D.; Topcu, U.; and Wu, B. 2020. Joint Inference of Reward Machines and Policies for Reinforcement Learning. In *ICAPS*, 590–598. AAAI Press. URL https://aaai.org/ojs/index.php/ICAPS/article/view/6756.