AUTOMATON DISTILLATION: A NEURO-SYMBOLIC TRANSFER LEARNING APPROACH FOR DEEP RL

Anonymous authors
Paper under double-blind review

ABSTRACT

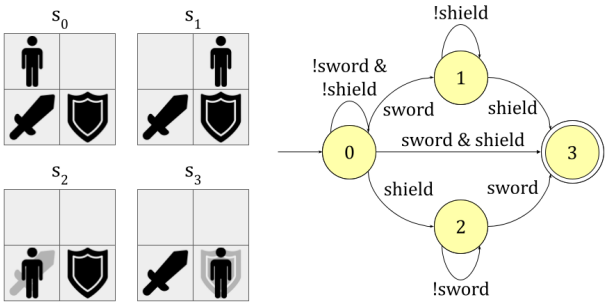
Reinforcement learning is a powerful tool for finding optimal policies in sequential decision processes. However, deep learning methods suffer from two weaknesses: collecting the amount of agent experience required for practical RL problems is prohibitively expensive, and the learned policies exhibit poor generalization on tasks outside the training distribution. To mitigate these issues, we introduce automaton distillation, a form of neuro-symbolic transfer learning in which Q-value estimates from a teacher are distilled into a low-dimensional representation in the form of an automaton. We then propose two methods for generating Q-value estimates: static transfer, which reasons over an abstract MDP constructed based on prior knowledge, and dynamic transfer, where symbolic information is extracted from a DQN teacher. The resulting Q-value estimates from either method are used to bootstrap learning in the target environment via a modified DQN loss function. We list several failure modes of existing automaton-based transfer methods and demonstrate that both static and dynamic automaton distillation decrease the time required to find optimal policies for various decision tasks.

INTRODUCTION

Sequential decision tasks, in which an agent seeks to learn a policy to maximize long-term reward through trial and error, are often solved using reinforcement learning approaches. These approaches must balance exploration - the acquisition of novel experiences - with exploitation - taking the predicted best action based on knowledge gained from exploration. Performing sufficient exploration to find the optimal policy requires collecting a large amount of experience, which can be expensive. Indeed, it has been observed that deep learning requires far more data and training time than human learners to achieve comparable effectiveness at a given task. Thus, it is desirable to improve the sample efficiency of RL algorithms.

One explanation for the high sample efficiency observed in human learning relative to neural networks is the ability to apply high-level concepts learned through prior experience to environments not encountered during training. Although traditional deep learning methods do not contain an explicit notion of abstraction, neuro-symbolic computing has shown promise as a way to integrate high-level symbolic reasoning into neural approaches. Symbolic logic provides a formal mechanism for injecting and extracting knowledge from neural networks to guide learning and provide explainability, respectively (Tran and Garcez, 2016). Additionally, reinforcement learning over policies expressed as symbolic programs has been shown to improve performance and generalization on previously unseen tasks (Verma et al., 2018; Anderson et al., 2020).

We adopt a different approach which uses a symbolic representation of RL objectives to facilitate knowledge transfer between an expert in a related source domain (the 'teacher') and an agent learning the target task (the 'student'). Although it is common to use reward signals to convey an objective, many decision tasks can be more naturally expressed as a high-level description of the intermediate steps required to achieve the objective in the form of natural language. These natural language descriptions can be translated into a corresponding specification in a formal language such as linear temporal logic (LTL) (Brunello, Montanari, and Reynolds, 2019), which can be converted



(a) (b)

Figure 1: (a) A simple NMRDP $\mathcal{M} = \langle S, s_0, A, T, R \rangle$ with four states $S = \{s_0, s_1, s_2, s_3\}$ and starting state s_0. The agent can take one of four actions $A = \{\text{LEFT}, \text{DOWN}, \text{RIGHT}, \text{UP}\}$, and the transition function T is defined accordingly. A sequence of actions satisfies the objective iff the agent eventually reaches both the sword tile and the shield tile. The reward function R is defined such that the agent receives +1 reward for reaching the sword tile and +1 reward for reaching the shield tile. The reward signal can be decomposed using the atomic propositions $AP = \{\text{sword}, \text{shield}\}$, with a corresponding labeling function L such that $L(s_0) = \{\}, L(s_1) = \{\}, L(s_2) = \{\text{sword}\}, L(s_3) = \{\text{shield}\}$. Rollouts which achieve the objective also satisfy the LTL_f specification $\phi = \mathbf{F}(\text{sword}) \wedge \mathbf{F}(\text{shield})$. (b) An automaton $\mathcal{A} = \langle \Sigma, \Omega, \omega_0, F, \delta \rangle$ defined over the alphabet $\Sigma = \{\{\}, \{\text{sword}\}, \{\text{shield}\}, \{\text{sword, shield}\}\}$. The automaton accepts the subset of strings in Σ^* that satisfy the LTL_f formula.

into an equivalent automaton representation (Wolper, Vardi, and Sistla, 1983). The resulting automaton acts as a common language between the teacher and student tasks; states and actions in the source and target domains can be mapped to nodes and transitions in the automaton, respectively. Furthermore, by assigning value estimates to automaton transitions, the automaton representation can be transformed into a compact model of the environment to facilitate learning.

In this paper, we develop two variants of transfer learning which leverage the automaton representation of an objective to convey information about the reward signal from the teacher to the student. The first, static transfer, generates estimates of the Q-value of automaton transitions by performing value iteration over the abstract MDP defined by the automaton. The second, dynamic transfer, distills knowledge from a pre-trained DQN into the automaton by mapping teacher Q-values of state-action pairs in the experience replay buffer to their corresponding transition in the automaton. Q-value estimates generated by either method can subsequently be used to bootstrap the learning process for the student. We argue that other automaton-based transfer methods may induce *negative transfer* in some cases, and demonstrate that our proposed method can reduce training time even in cases where existing methods harm performance.

PRELIMINARIES

We assume the teacher and student decision processes are non-Markovian Reward Decision Processes (NMRDP), as defined below.

Definition 1 (Non-Markovian Reward Decision Process (NMRDP))**.** *An NMRDP is a decision process defined by the tuple $\mathcal{M} = \langle S, s_0, A, T, R \rangle$, where S is the set of valid states, $s_0 \in S$ is the initial state, A is the set of valid actions, $T : S \times A \times S \to \mathbb{R}$ is a transition function defining transition probabilities for each state-action pair to every state in S, and $R : (S \times A)^* \to \mathbb{R}$ defines the reward signal observed at each time step based on the sequence of previously visited states and actions.*

NMRDPs differ from Markov Decision Processes (MDPs) in that the reward signal R may depend on the entire history of observations, rather than only the current state. However, the reward signal is often a function of a set of abstract properties of the current state, which is of much smaller dimension than the original state space. Thus, it can be beneficial to represent the reward signal in terms of a simpler vocabulary defined over features extracted from the state. We assume the

existence of a set of atomic propositions AP for each environment which capture the dynamics of the reward function as well as a labeling function $L : S \to 2^{AP}$ that translates experiences into truth assignments for each proposition $p \in AP$. Furthermore, we assume that the objectives in the teacher and student environments are identical and represented by the automaton $\mathcal{A} = (2^{AP}, \Omega, \omega_0, F, \delta)$ as defined below.

Definition 2 (Deterministic Finite-State Automaton (DFA))**.** *A DFA is an automaton defined by the tuple $\mathcal{A} = \langle \Sigma, \Omega, \omega_0, F, \delta \rangle$, where Σ is the alphabet of the input language, Ω is the set of states with starting state ω_0, $F \subseteq \Omega$ is the set of accepting states, and $\delta : \Omega \times \Sigma \mapsto \Omega$ defines a state transition function.*

The atomic propositions AP comprise a vocabulary of abstract properties of the state space which directly correspond to the reward structure. Using the labeling function L, states in an NMRDP can be mapped to an element in the alphabet $\Sigma = 2^{AP}$. Then the set of rollouts which satisfy the objective constitute a regular language over Σ. The parameters $\Omega, \omega_0, F, \delta$ are chosen such that the set of strings accepted by the objective automaton $\mathcal{A}$ is equivalent to the aforementioned regular language; we illustrate this with a simple example in Figure 1. An additional consequence of developing such a vocabulary is that RL objectives can be expressed as a formal language and subsequently converted into a DFA. Then, reward functions can be defined over automaton transitions rather than the original state-action space.

Finally, we assume access to an oracle in the form of a DQN trained in the teacher environment. We would like to transfer information from a teacher to a student by learning parameters over the states and/or transitions of the automaton in a surrogate environment and using those parameters to train a DQN in the target environment. That is, we distill knowledge from a teacher automaton to a student DQN. This contrasts with traditional policy distillation, which distills knowledge directly from the teacher DQN to the student DQN.

Related Work

Deep RL has made remarkable progress in many practical problems, such as recommendation systems, robotics, and autonomous driving. However, despite all successes, insufficient data and poor generalization remain open problems in deep RL. In most real-world problems, it is difficult to obtain a sufficient amount of training data, so RL agents often learn with simulated data. However, RL agents trained with simulated data usually have poor performance when it is transferred to unknown environment dynamics in real-world data. To address these two challenges, Transfer Learning techniques (Zhu, Lin, and Zhou, 2020) have been adopted to solve two RL tasks: 1) state representation transfer and 2) policy transfer.

Among all transfer learning techniques, Domain Adaptation (DA) is the most well-studied in deep RL. Early attempts at DA construct a mapping from states and actions in the source domain onto the target domain by hand (Taylor and Stone, 2005). Subsequent works aim to learn a set of general latent environment representations that can be transferred across different but similar domains. For example, a multi-stage agent, DisentAngled Representation Learning Agent (DARLA) (Higgins et al., 2017), proposed to learn a general representation by adding an internal layer of pre-trained Denosising AutoEncoder (DAE). With learned general representation in the source domain, the agent quickly learns a robust policy that produces decent performance on similar target domains without further tuning. However, DARLA cannot clearly define and separate domain-specific and domain-general features, and it causes performance degradation on some target tasks. Moreover, Contrastive Unsupervised Representations for Reinforcement Learning (CURL) (Srinivas, Laskin, and Abbeel, 2020) aimed to address this issue by integrating contrastive loss. Furthermore, Latent Unified State Representation (LUSR) (Xing et al., 2021) proposed a two-stage agent that can fully separate domain-general and domain-specific features by embedding both the forward loss and the reverse loss in Cycle-Consistent AutoEncoder (Jha et al., 2018).

Previous work has explored the use of high-level symbolic domain descriptions to construct a low-dimensional abstraction of the original state space (Kokel et al., 2022), which can be used to model the dynamics of the original system. (Icarte et al., 2022) further construct an automaton which realizes the abstract decision process and use the automaton to convey information about the reward signal. One benefit of the automaton representation is the ability to express non-Markovian reward

signals in terms of the automaton state. During an episode, the state of the automaton can be computed in parallel with observations from the learning environment. Given the current automaton state ω and a new observation s', the new automaton state can be computed as $\omega' = \delta(\omega, L(s'))$. We assume that the atomic propositions capture the non-Markovian behavior of the reward signal; thus, the system dynamics are Markovian in the cross-product of the observation and automaton state spaces. Formally, we represent the cross-product of an NMRDP and its corresponding objective automaton as a Markov Decision Process.

Definition 3 (Cross-Product Markov Decision Process)**.** *The cross-product of an NMRDP $\mathcal{M} = \langle S, s_0, A, T, R\rangle$ and a DFA $\mathcal{A} = \langle \Omega, \omega_0, \Sigma, \delta, F\rangle$ which captures the non-Markovian behavior of the reward signal is a Markov Decision Process (MDP) $\mathcal{M}_{prod} = \langle S \times \Omega, (s_0, \omega_0), A \times \Sigma, T \times \delta, R'\rangle$ where $R' : \Omega \times \Sigma \to \mathbb{R}$ is a Markovian reward signal (i.e. can be expressed as a function of only the current state and action).*

Note that transforming an NMRDP into a cross-product MDP permits the use of traditional RL algorithms, which rely upon the Markovian assumption, for decision tasks with non-Markovian reward signals.

Implicit in the previous discussion is that the objective is translated into an automaton prior to learning. The infeasibility of explicitly constructing an automaton to represent a desired objective has motivated the development of automated methods for converting a reward specification into an automaton representation. Such methods observe a correspondence between various formal logics (such as regular expressions, LTL, and its variants) and finite-state automata (Wolper, Vardi, and Sistla, 1983). In particular, finite-trace Linear Temporal Logic (LTL_f) has been used to represent reinforcement learning objectives (Camacho et al., 2019; Velasquez et al., 2021).

Definition 4 (Finite-Trace Linear Temporal Logic (LTL_f))**.** *A formula in LTL_f consists of a set of atomic propositions AP which are combined by the standard propositional operators and the following temporal operators: the next operator $\boldsymbol{X}\phi$ (ϕ will be true in the next time step), the eventually operator $\boldsymbol{F}\phi$ (ϕ will be true in some future time step), the always operator $\boldsymbol{G}\phi$ (ϕ will be true in all future time steps), the until operator ϕ_1 $\boldsymbol{U}$ ϕ_2 (ϕ_2 will be true in some future time step, and until then ϕ_1 must be true), and the release operator ϕ_1 $\boldsymbol{R}$ ϕ_2 (ϕ_2 must be true always or until ϕ_1 first becomes true).*

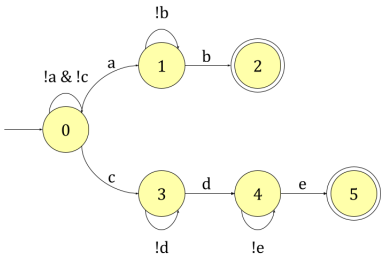
It has been shown that specifications in LTL_f can be transformed into an equivalent deterministic Büchi automaton (De Giacomo and Vardi, 2015), and tools for compiling automata are readily available (Zhu et al., 2017). Moreover, it is possible, in principle, to convert descriptions using a predefined subset of natural language into linear temporal logic (Brunello, Montanari, and Reynolds, 2019). Thus, it is feasible to translate a specification provided by a domain expert into an automaton representation using automated methods.

Static transfer learning methods which utilize the automaton representation in conjunction with the reward function were developed in (Camacho et al., 2018; Icarte et al., 2022). By treating the nodes and edges in the automaton as states and actions respectively, the automaton can be transformed into a low-dimensional abstract MDP which can be solved using Q-learning or value iteration approaches. The solution to the abstract MDP can be used to speed up learning in the original environment by using a potential-based reward shaping function (Camacho et al., 2019) or by introducing counterfactual experiences during training (Icarte et al., 2022).

However, static transfer methods perform poorly when the abstract MDP fails to capture the behavior of the underlying process. Consider applying the static transfer approach proposed in (Icarte et al., 2022) for the objective defined by the LTL_f formula $\phi = \mathbf{F}(\text{b} \vee \text{e}) \wedge (\neg\mathbf{F}(\text{a}) \vee \neg\mathbf{F}(\text{c})) \wedge (\text{a}\ \mathbf{R}\ \neg\text{b}) \wedge (\text{c}\ \mathbf{R}\ \neg\text{d}) \wedge (\text{d}\ \mathbf{R}\ \neg\text{e})$, whose automaton is given below:

Assume that the reward function grants a reward of 1 for transitions leading to either terminal state and a reward of 0 for all other transitions. (Icarte et al., 2022) perform value iteration over the abstract MDP:

$$V(\omega) := \max_{\omega' = \delta(\omega,\sigma)} R(\omega, \sigma) + \gamma V(\omega') \tag{1}$$



In the previous automaton, there are two traces which satisfy the objective: one of length 2 and one of length 3. Due to the discount factor $\gamma < 1$, value iteration will favor taking transition a, which has a shorter accepting path, over transition c in the starting state. However, it may be the case that observing b after observing a takes many steps in the original environment, and thus the longer trace $c \rightarrow d \rightarrow e$ takes less steps to reach an accepting state.

(Camacho et al., 2018) suggests another method for static transfer using automata which, instead of using value iteration to generate a reward shaping function, defines the potential for each node as the inverse of the distance between each node and the closest accepting state in the automaton. This method suffers from the same failure case as (Icarte et al., 2022); shorter traces in the automaton do not necessarily require fewer steps in the original process.

In contrast to static transfer methods, which use prior knowledge of the reward function to model the behavior of the target process, dynamic transfer leverages experience acquired by interaction in a related domain to empirically estimate the target value function. Dynamic transfer has the advantage of implicitly factoring in knowledge of the teacher environment dynamics; in the previous example, if the shorter trace $a \rightarrow b$ takes more steps in the teacher environment than the longer trace $c \rightarrow d \rightarrow e$, this will be reflected in the discounted value estimates learned by the teacher.

We focus on a dynamic transfer learning algorithm which distills value estimates from an agent trained using Deep Q-Learning into the objective automaton via a contractive mapping from state-action pairs in the teacher NMRDP to the abstract MDP defined by the automaton. An analogous expansive mapping from the abstract MDP to the student NMRDP provides an initial estimate of the Q-value of state-action pairs in the target domain, which can be used to bootstrap the learning of a student Deep Q-Network. Using the automaton as an intermediary between the teacher and student environments allows experiences to be shared between domains without requiring handcrafted state-space maps (Taylor and Stone, 2005) or learning maps in an unsupervised manner (Ammar et al., 2015).

AUTOMATON DISTILLATION

In automaton distillation, a teacher agent is trained using standard Deep Q-Learning. Then, teacher Q-values are distilled into the objective automaton such that the value of each transition in the automaton represents an estimate of the Q-value of the corresponding state-action pair in the teacher environment. Finally, a student is trained in the target environment using a modified version of the DQN loss function which incorporates the automaton Q-values. The standard DQN loss function is given in Equation (2), where ER is an experience replay buffer with samples collected over the course of training, U denotes the uniform distribution, and $Q(s, a; \theta)$ is the student DQN parameterized by θ which accepts the state s as input and outputs the predicted value for taking action a in state s. This standard DQN loss function is defined in terms of an underlying Markov Decision Process (MDP). It is worth noting that θ^{target} is often taken from an old copy of the DQN in order to stabilize learning.

$$Loss(\theta) = \mathbb{E}_{(s,a,r,s') \sim U(ER)}[(r + \gamma \max_{a'} Q(s', a'; \theta^{\text{target}}) - Q(s, a; \theta))^2] \tag{2}$$

The teacher DQN is trained using only standard reinforcement learning methods (Wang et al., 2016; Van Hasselt, Guez, and Silver, 2016; Schaul et al., 2015). However, in order to track the current node in the automaton as well as the current state, we store samples of the form $((s, \omega), a, r, (s', \omega'))$ in

the experience replay buffer ER. We define $\eta_{\text{teacher}} : \Omega \times \Sigma \to \mathbb{N}$ to be the number of times each automaton node ω and a set of atomic propositions $\sigma \in 2^{AP}$ appears in the experience replay ER of the teacher (note that ω and σ define a transition in the automaton objective as given by $\delta(\omega, \sigma) = \omega'$):

$$\eta_{\text{teacher}}(\omega, \sigma) = |\{((s, \omega), a, r, (s', \omega')) \in ER | L(s') = \sigma\}| \tag{3}$$

Similarly, we define $Q^{\text{avg}}_{\text{teacher}} : \Omega \times \Sigma \to \mathbb{R}$ to be the average Q-value corresponding to the automaton transition given by ω and $\sigma \in 2^{AP}$, according to the teacher DQN.

$$Q^{\text{avg}}_{\text{teacher}}(\omega, \sigma) = \frac{\sum_{\{((s,\omega),a,r,(s',\omega')) \in ER | L(s') = \sigma\}} Q_{\text{teacher}}(s, a)}{\eta_{\text{teacher}}(\omega, \sigma)} \tag{4}$$

We leverage the preceding equations to create a new loss function (shown in Equation 5) during training of the student network, whereby the average Q-values of the teacher corresponding to transitions in the automaton objective are pre-computed and used to bootstrap the learning process. Intuitively, using teacher Q-values to estimate the optimal value function during the early phases of training resolves the issue of poor initial value estimates, which causes slow convergence in standard reinforcement learning algorithms. As we show in the experiments, this is an effective means of performing non-Markovian knowledge transfer since the automaton objective is a much lower-dimensional representation of the environment dynamics and encodes the non-Markovian reward signal.

$$\begin{aligned} Loss(\theta) = \mathbb{E}_{((s,\omega),a,r,(s',\omega')) \sim U(ER)} [\beta(\omega, L(s')) Q^{\text{avg}}_{\text{teacher}}(\omega, L(s')) \\ + (1 - \beta(\omega, L(s')))(r + \gamma \max_{a'} Q(s', a'; \theta^{\text{target}})) - Q(s, a; \theta)]^2 \end{aligned} \tag{5}$$

In the preceding equation, $\beta : \Omega \times \Sigma \to [0, 1]$ is an annealing function that controls the importance of the initial Q-value estimate given by the automaton relative to the standard Q-learning update. We use $\beta(\omega, \sigma) = \rho^{\eta_{student}(\omega,\sigma)}$ where $\rho = 0.999$ and $\eta_{student}(\omega, \sigma)$ represents how many times the automaton transition defined by ω and σ has been previously sampled from the experience replay buffer while training the student.

The asymptotic behavior of the automaton Q-learning algorithm depends on the choice of β; when $\beta = 0$, automaton Q-learning reduces to standard Q-learning. We provide the following proof of convergence for tabular automaton Q-learning.

Theorem 1 The automaton Q-learning algorithm given by

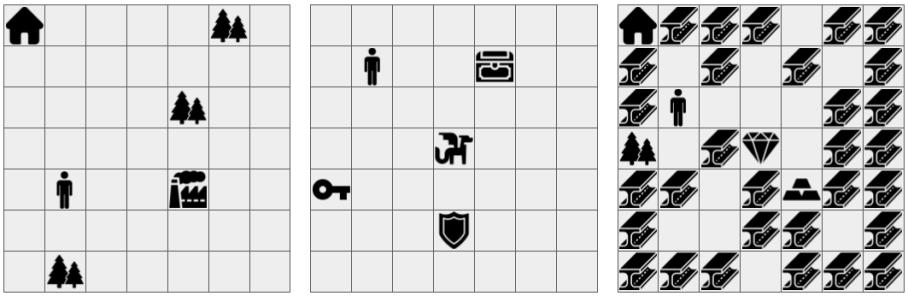
$$\begin{aligned} Q_{t+1}(s_t, a_t) =& (1 - \alpha_t) Q_t(s_t, a_t) + \alpha_t (1 - \beta_t)(R(s_t, a_t) + \gamma V_t(s_{t+1})) \\ &+ \alpha_t \beta_t Q^{\text{avg}}_{\text{teacher}}(\omega, L(s_{t+1})) \end{aligned} \tag{6}$$

converges to the optimal $Q^*(s, a)$ values if

1. The state and action spaces are finite.
2. $\alpha_t \in [0, 1)$, $\sum_t \alpha_t = \infty$ and $\sum_t \alpha_t^2 < \infty$.
3. $\beta_t \geq 0$, $\lim_{t \to \infty} \beta_t = 0$, and $\sum_t \alpha_t (1 - \beta_t) = \infty$.
4. $\text{Var}(R(s, a))$ is bounded.
5. $\gamma = 1$ and all policies lead to a cost-free terminal state; otherwise, $\gamma \in [0, 1)$.

Proof: We decompose automaton Q-learning into two parallel processes q and r given by

$$\begin{aligned} q_{t+1}(s_t, a_t) =& (1 - \alpha_t) q_t(s_t, a_t) + \alpha_t (1 - \beta_t)(R(s_t, a_t) + \gamma V_t(s_{t+1})) \\ r_{t+1}(s_t, a_t) =& (1 - \alpha_t) r_t(s_t, a_t) + \alpha_t \beta_t Q^{\text{avg}}_{\text{teacher}}(\omega, L(s_{t+1})) \end{aligned} \tag{7}$$



(a) (b) (c)

Figure 2: Example 7×7 environment configurations for the *Blind Craftsman* (a), *Dungeon Quest* (b), and *Diamond Mine* (c) environments.

such that $Q_t(s, a) = q_t(s, a) + r_t(s, a)$. q corresponds to a modified version of Q-learning; using the same annealing function as in Equation 5, q converges to the optimal Q-table Q^* w.p.1 as shown in (Jaakkola, Jordan, and Singh, 1993). To see that r converges to 0 w.p.1, we observe that $Q^{\text{avg}}_{\text{teacher}}(\omega, L_{s_{t+1}})$ is constant when training the student and so r_t is a contraction whose fixed point occurs at

$$r_t(s_t, a_t) = \beta_t Q^{\text{avg}}_{\text{teacher}}(\omega, L(s_{t+1})) \tag{8}$$

Since $\lim_{t\to\infty} \beta = 0$, the fixed point of r_t approaches 0 as $t \to \infty$. Thus, since q and r converge to Q^* and 0 respectively w.p.1, their sum Q converges to Q^* w.p.1.

As an alternative to dynamic automaton distillation, static value estimates can be effective in cases where the abstract MDP defined by the automaton accurately captures the environment dynamics. Such estimates can be computed via tabular Q-learning over the abstract MDP:

$$Q(\omega, \sigma) := Q(\omega, \sigma) + \alpha(R(\omega, \sigma) + \gamma \max_{\sigma'} Q(\omega', \sigma') - Q(\omega, \sigma)) \tag{9}$$

The resulting Q-values can be used in the place of $Q^{\text{avg}}_{\text{teacher}}$ in Equation 5. This method has the benefit of stabilizing training in the early stages without requiring any additional information beyond the reward structure.

Experimental Results

We evaluate both variants of automaton distillation on various grid-world environments. We use a 7×7 version of each grid-world environment as the source domain and a corresponding 10×10 version as the target domain. Agents are trained in a parallel fashion on 8 environment instances. The layout of the grid-world is randomly generated once for each environment instance and subsequently kept fixed across interaction episodes. In each grid-world environment, the objective is expressed as an LTL_f formula over atomic propositions corresponding to the agent's inventory and the tile the agent is currently standing on. The state space consists of the current location of the agent and all special tiles as well as the agent's inventory. As an action, the agent may either move one square in any of the cardinal directions or interact with the tile the agent is currently standing on. The environments we use are described below (sample 7×7 configurations for each environment can be seen in Figure 2).

Blind Craftsman: This environment consists of wood tiles, factory tiles, and a home tile. The agent can acquire wood by standing on a wood tile and taking the interact action (the agent may carry a maximum of two wood at a time). Once the agent has collected wood, it can craft a tool by standing on a factory tile and taking the interact action (one wood must be consumed to craft each tool). The objective is satisfied when the agent has crafted three tools and arrived at the home space. Since the agent can only carry two wood at a time, the agent must alternate between collecting wood and

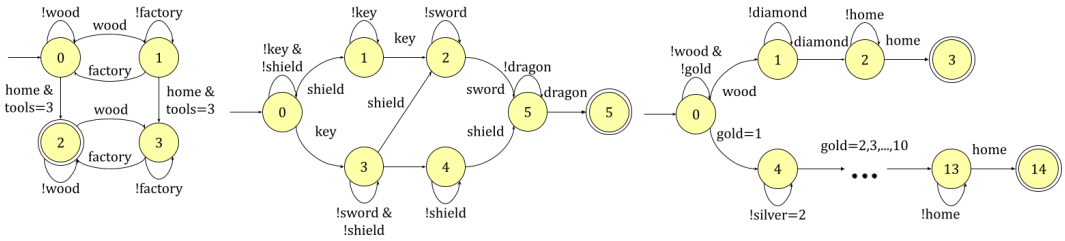


Figure 3: Objective automata for the *Blind Craftsman* (a), *Dungeon Quest* (b), and *Diamond Mine* environments.

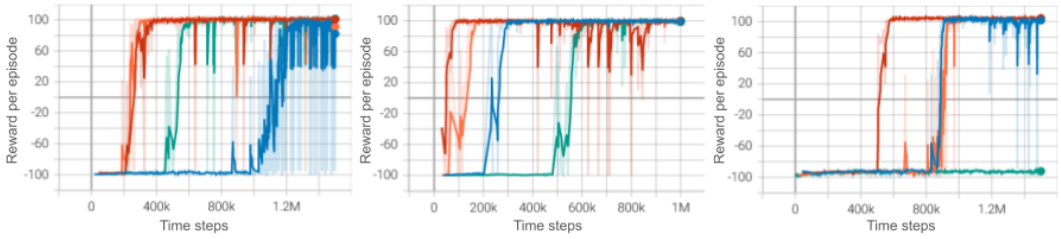
crafting tools. The objective is defined over the atomic propositions $AP = \{\text{wood}, \text{factory}, \text{tools} \geq 3, \text{home}\}$ and given by the LTL$_f$ formula $\phi = \mathbf{G}(\text{wood} \implies \mathbf{F}\,\text{factory}) \wedge \mathbf{F}(\text{tools} \geq 3 \wedge \text{home})$ with a corresponding automaton of 4 nodes and 12 transitions. The agent receives a reward of +1 for each wood collected and tool crafted, a reward of +100 for returning home with at least three tools, and a reward of -0.1 per time step.

Dungeon Quest: This environment consists of a key tile, a chest tile, a shield tile, and a dragon tile. The agent can acquire a key and a shield by taking the interact action while standing on the key or shield tile, respectively. Additionally, the agent can obtain a sword by standing on the chest tile and taking the interact action if it has already obtained the key. The agent may choose to acquire these objects in one of several orders: either it may obtain the shield before obtaining the key, after obtaining the key but before obtaining the sword, or after obtaining the key and the sword. Once the agent has both the sword and the shield, it may traverse to the dragon tile to defeat it and complete the objective. The objective is defined over the atomic propositions $AP = \{\text{key}, \text{shield}, \text{sword}, \text{dragon}\}$ and given by the LTL$_f$ formula $\phi = \mathbf{F}(\text{dragon}) \wedge (\text{key}\ \mathbf{R}\ \neg\text{sword}) \wedge (\text{sword}\ \mathbf{R}\ \neg\text{dragon}) \wedge (\text{shield}\ \mathbf{R}\ \neg\text{dragon})$ with a corresponding automaton of 7 nodes and 17 transitions. The agent receives a reward of +1 for collecting each item, a reward of +100 for defeating the dragon, and a reward of -0.1 per time step.

Diamond Mine: This environment consists of a wood tile, a diamond tile, gold tiles, and iron tiles. To achieve the objective, the agent has two options: it may collect either 1 diamond or 10 gold. The agent may acquire wood, iron, or gold by taking the interact action while standing on a wood, iron, or gold tile, respectively. Once the agent has collected wood and 30 iron, it automatically crafts a pickaxe. The agent may then obtain diamond by standing on the diamond tile while holding a pickaxe. Once the agent has acquired either 1 diamond or 10 gold, it may return to the home tile to complete the objective. To simplify the resulting automaton and limit unnecessary reward, once the agent has collected gold, it cannot obtain the diamond, and vice versa. The objective is defined over the atomic propositions $AP = \{\text{wood}, \text{diamond}, \text{gold = 1}, \text{gold = 2}, ..., \text{gold = 10}, \text{home}\}$ and given by the LTL$_f$ formula $\phi = \mathbf{F}(\text{home}) \wedge (\neg\mathbf{F}(\text{gold} = 1) \vee \neg\mathbf{F}(\text{wood})) \wedge (\text{wood}\ \mathbf{R}\ \neg\text{diamond}) \wedge (\text{gold = 1}\ \mathbf{R}\ \neg\text{gold = 2}) \wedge ... \wedge (\text{gold = 9}\ \mathbf{R}\ \neg\text{gold = 10}) \wedge ((\text{diamond} \vee \text{gold = 10})\ \mathbf{R}\ \neg\text{home})$ with a corresponding automaton of 15 nodes and 29 transitions. The agent receives a reward of +1 for collecting gold, +10 for collecting diamond, a reward of +100 for returning home, and a reward of -0.1 per time step.

Agents are trained for 1.5 million play steps, or until convergence. Each agent is represented by a Dueling DQN (Wang et al., 2016), which consists of a convolutional feature extractor and separate value and advantage heads. The feature extractor is a residual network with 3 residual blocks, each using a 3×3 convolutional kernel with 32 filters and Leaky ReLU activation. The resulting feature map is flattened and split into equal halves, which are fed separately to the value and advantage heads. The value and advantage heads each contain a single fully connected layer with 1 and $\#\ actions$ nodes, respectively.

The network takes as input a stack of 2D grids, where each layer represents a single entity type (either the agent, a tile type, or an inventory item type). In each layer, the value 1 at a given position indicates the presence of the corresponding entity at that position and the value 0 indicates otherwise. Each inventory item is represented by a constant-valued input plane. Additionally, as proposed in (Icarte et al., 2022), we incorporate the automaton state into the input, training on elements of



(a) (b) (c)

Figure 4: Reward per episode achieved over the course of training using dynamic automaton distillation (red) vs. static automaton distillation (orange), CRM (Icarte et al., 2022) (blue), and vanilla Q-learning (green) on the *Blind Craftsman* (a), *Dungeon Quest* (b), and *Diamond Mine* (c) environments.

the cross-product MDP state space $(s, \omega) \in S \times \Omega$. The objective automaton is generated using the SPOT synthesis tool (Duret-Lutz and Poitrenaud, 2004) based on the LTL_f behavioral specification; automata for each environment are shown in Figure 3. The automaton state is then tracked throughout each training episode and stored alongside each experience in the replay buffer. The automaton state is converted to a one-hot vector representation and concatenated to each half of the convolutional feature extractor output.

For each environment, we evaluate the performance of automaton distillation (static and dynamic variants) against the state-of-the-art static transfer learning algorithm proposed in (Icarte et al., 2022) and vanilla Q-learning. Figure 4 shows the number of training steps required for each algorithm to converge to an optimal policy. Automaton distillation outperforms existing methods and vanilla Q-learning in all environments. A primary use case for dynamic transfer learning occurs in the *Diamond Mine* environment, where the assumption that short automaton traces correlate with shorter trajectories in the original environment is misleading. Dynamic automaton distillation instead utilizes an empirical estimate of trajectory length over the teacher decision process, circumventing the inaccuracies in the abstract MDP. Thus, dynamic automaton distillation is effective when the optimal policies in the teacher and student environments follow similar automaton traces.

Some behavioral specifications can lead to objective automata with cycles, as evidenced by the *Blind Craftsman* environment. Cycles in the automaton do not necessarily lead to infinite reward loops - it is often the case that, in the original environment, the cycle may be taken only a finite number of times. While it is possible to construct an automaton without cycles by expanding the state space of the automaton to include the number of cycles taken, the maximum number of cycle traversals must be known *a priori* and incorporated into the objective specification. Additionally, in transfer learning, environments which share an objective may admit different numbers of cycle traversals; thus, cycles offer a compact representation which permits knowledge transfer between environments. However, the presence of cycles can aggravate the differences between the abstract MDP and original decision process, resulting in negative knowledge transfer. In such cases, state-of-the-art transfer methods (Icarte et al., 2022) may actually *increase* training time relative to a naïve learning algorithm.

CONCLUSION

In this paper we propose automaton distillation, which leverages symbolic knowledge of the objective and reward structure in the form of formal language, to stabilize and expedite training of reinforcement learning agents. Value estimates for transitions in the automaton are either generated using static (i.e. *a priori*) methods such as value iteration over an abstract representation of the target domain or dynamically estimated by mapping experiences collected in a related source domain to automaton transitions. The resulting value estimates are used by the student as initial learning targets to bootstrap the learning process. We illustrate several failure cases of existing automaton-based transfer methods, which exclusively reason over *a priori* knowledge, and argue instead for the use of dynamic transfer. We perform experiments showing that static and dynamic variants of automaton distillation reduce training costs and outperform state-of-the-art knowledge transfer techniques.

REFERENCES

Ammar, H. B.; Eaton, E.; Luna, J. M.; and Ruvolo, P. 2015. Autonomous cross-domain knowledge transfer in lifelong policy gradient reinforcement learning. In *Twenty-fourth international joint conference on artificial intelligence*.

Anderson, G.; Verma, A.; Dillig, I.; and Chaudhuri, S. 2020. Neurosymbolic reinforcement learning with formally verified exploration. *Advances in neural information processing systems* 33:6172–6183.

Brunello, A.; Montanari, A.; and Reynolds, M. 2019. Synthesis of ltl formulas from natural language texts: State of the art and research directions. In *26th International Symposium on Temporal Representation and Reasoning (TIME 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

Camacho, A.; Chen, O.; Sanner, S.; and McIlraith, S. A. 2018. Non-markovian rewards expressed in ltl: Guiding search via reward shaping (extended version). In *GoalsRL, a workshop collocated with ICML/IJCAI/AAMAS*.

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*, volume 19, 6065–6073.

De Giacomo, G., and Vardi, M. 2015. Synthesis for ltl and ldl on finite traces. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*.

Duret-Lutz, A., and Poitrenaud, D. 2004. Spot: an extensible model checking library using transition-based generalized bu/spl uml/chi automata. In *The IEEE Computer Society's 12th Annual International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems, 2004.(MASCOTS 2004). Proceedings.*, 76–83. IEEE.

Higgins, I.; Pal, A.; Rusu, A.; Matthey, L.; Burgess, C.; Pritzel, A.; Botvinick, M.; Blundell, C.; and Lerchner, A. 2017. Darla: Improving zero-shot transfer in reinforcement learning. In *International Conference on Machine Learning*, 1480–1490. PMLR.

Icarte, R. T.; Klassen, T. Q.; Valenzano, R.; and McIlraith, S. A. 2022. Reward machines: Exploiting reward function structure in reinforcement learning. *Journal of Artificial Intelligence Research* 73:173–208.

Jaakkola, T.; Jordan, M.; and Singh, S. 1993. Convergence of stochastic iterative dynamic programming algorithms. *Advances in neural information processing systems* 6.

Jha, A. H.; Anand, S.; Singh, M.; and Veeravasarapu, V. R. 2018. Disentangling factors of variation with cycle-consistent variational auto-encoders. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 805–820.

Kokel, H.; Prabhakar, N.; Ravindran, B.; Blasch, E.; Tadepalli, P.; and Natarajan, S. 2022. Hybrid deep reprel: Integrating relational planning and reinforcement learning for information fusion. In *2022 25th International Conference on Information Fusion (FUSION)*, 1–8. IEEE.

Schaul, T.; Quan, J.; Antonoglou, I.; and Silver, D. 2015. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*.

Srinivas, A.; Laskin, M.; and Abbeel, P. 2020. Curl: Contrastive unsupervised representations for reinforcement learning. *arXiv preprint arXiv:2004.04136*.

Taylor, M. E., and Stone, P. 2005. Behavior transfer for value-function-based reinforcement learning. In *Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, 53–59.

Tran, S. N., and Garcez, A. S. d. 2016. Deep logic networks: Inserting and extracting knowledge from deep belief networks. *IEEE transactions on neural networks and learning systems* 29(2):246–258.

Van Hasselt, H.; Guez, A.; and Silver, D. 2016. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30.

Velasquez, A.; Bissey, B.; Barak, L.; Beckus, A.; Alkhouri, I.; Melcer, D.; and Atia, G. 2021. Dynamic automaton-guided reward shaping for monte carlo tree search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, 12015–12023.

Verma, A.; Murali, V.; Singh, R.; Kohli, P.; and Chaudhuri, S. 2018. Programmatically interpretable reinforcement learning. In *International Conference on Machine Learning*, 5045–5054. PMLR.

Wang, Z.; Schaul, T.; Hessel, M.; Hasselt, H.; Lanctot, M.; and Freitas, N. 2016. Dueling network architectures for deep reinforcement learning. In *International conference on machine learning*, 1995–2003. PMLR.

Wolper, P.; Vardi, M. Y.; and Sistla, A. P. 1983. Reasoning about infinite computation paths. In *24th Annual Symposium on Foundations of Computer Science (sfcs 1983)*, 185–194. IEEE.

Xing, J.; Nagata, T.; Chen, K.; Zou, X.; Neftci, E.; and Krichmar, J. L. 2021. Domain adaptation in reinforcement learning via latent unified state representation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, 10452–10459.

Zhu, S.; Tabajara, L. M.; Li, J.; Pu, G.; and Vardi, M. Y. 2017. Symbolic ltlf synthesis. *arXiv preprint arXiv:1705.08426*.

Zhu, Z.; Lin, K.; and Zhou, J. 2020. Transfer learning in deep reinforcement learning: A survey. *arXiv preprint arXiv:2009.07888*.