Regular Reinforcement Learning

Taylor Dohmen(✉), Mateo Perez, Fabio Somenzi, and Ashutosh Trivedi

University of Colorado, Boulder, CO 80309, USA
{taylor.dohmen,mateo.perez,fabio,
ashutosh.trivedi}@colorado.edu

Abstract. In reinforcement learning, an agent incrementally refines a behavioral policy through a series of episodic interactions with its environment. This process can be characterized as *explicit* reinforcement learning, as it deals with explicit states and concrete transitions. Building upon the concept of symbolic model checking, we propose a *symbolic* variant of reinforcement learning, in which sets of states are represented through predicates and transitions are represented by predicate transformers. Drawing inspiration from regular model checking, we choose *regular languages* over the states as our predicates, and *rational transductions* as predicate transformations. We refer to this framework as *regular reinforcement learning*, and study its utility as a symbolic approach to reinforcement learning. Theoretically, we establish results around decidability, approximability, and efficient learnability in the context of regular reinforcement learning. Towards practical applications, we develop a deep regular reinforcement learning algorithm, enabled by the use of graph neural networks. We showcase the applicability and effectiveness of (deep) regular reinforcement learning through empirical evaluation on a diverse set of case studies.

Keywords: Reinforcement Learning · Regular Model Checking · Graph Neural Networks · Symbolic Techniques for Verification and Synthesis

1 Introduction

Reinforcement learning [51] (RL) is a sampling-based approach to synthesis, capable of producing solutions with superhuman efficiency [10,44,49]. An RL agent interacts with its environment through episodic interactions while receiving scalar rewards as feedback for its performance. Following the explicit/symbolic dichotomy of model checking approaches, the interactions in classic RL can be characterized as *explicit*: each episode consists of a sequence of experiences in which the agent chooses an action from a concrete state, observes the next concrete state, and receives an associated reward for this explicit interaction.

We envision a *symbolic* approach to RL, where each experience may deal with a set of states represented by a *predicate*, and the evolution of the system

© The Author(s) 2024
A. Gurfinkel and V. Ganesh (Eds.): CAV 2024, LNCS 14683, pp. 184–208, 2024.
https://doi.org/10.1007/978-3-031-65633-0_9

is described by *predicate transformers*. When the state space is large, symbolic representations may lead to greater efficiency and better generalization. Moreover, there are systems with naturally succinct representations—such as factored MDPs [27,32], succinct MDPs [23], and Petri nets [9]—that can benefit from symbolic manipulation of states.

The concept of symbolic interactions with an environment differs significantly from typical approximation methods used in RL, such as linear approximations [51] or deep neural networks [31]. In the context of such techniques, a learning agent attempts to generalize observations based on perceived similarities between them. In the symbolic setting, however, the generalization of a given interaction is explicitly provided by the environment itself. As a result, symbolic interactions facilitate a more direct form of generalization, where the environment ensures that similar interactions lead to similar outcomes.

This paper presents *regular reinforcement learning* (RRL), a symbolic approach to RL that employs regular languages and rational transductions, respectively, as models of predicates and their transformations. While natural languages can be used to encode symbolic interactions, we use regular languages [50] for the following reasons: (1) Regular languages enable unambiguous representation of predicates and predicate transformers. (2) Regular languages possess elegant theoretical properties including the existence of minimal canonical automata, determinizability, closure under many operations, and decidable emptiness and containment. (3) Regular languages hold a special position in machine learning, enjoying numerous efficient learnability results and active learning algorithms.

Regular languages also form the basis of a class of powerful symbolic model checking algorithm for infinite-state systems known as *regular model checking* (RMC) [2,5,13,18]. The following example introduces the concepts of RRL though a variation on the canonical token passing protocol used in the RMC literature [5,18].

Example 1 (Token Passing). The token passing protocol involves an arbitrary number of processes arranged in a linear topology and indexed by consecutive natural numbers. At any point in time, each process can be in one of two states: t if it has a token or n if it does not. The states of the system are then strings over the alphabet $\{t, n\}$. The initial state, in which only the leftmost process has a token, is the regular language tn^*.

At each time step, an agent chooses an action from the set $\{a, b, c\}$, and each of these actions corresponds to one of the following outcomes.

(a) Each even-indexed process with a token passes it to the right. Each odd-indexed process with a token passes a copy of it to the right.
(b) Each odd-indexed process with a token passes it to the right. Each even-indexed process with a token passes a copy of it to the right.
(c) The outcome of a (resp. b) occurs with probability p (resp. $(1 - p)$).

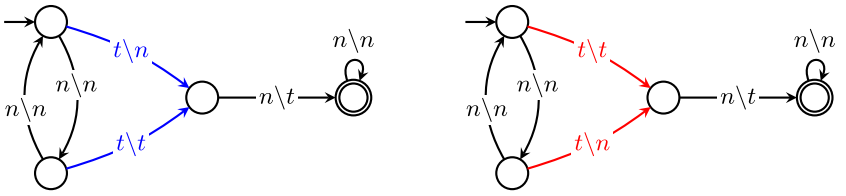


Fig. 1. An edge from q_0 to q_2 labelled by $t\backslash n$ denotes that if the transducer reads the symbol t from state q_0, then it outputs the symbol n and moves to q_2. Double-circled states are accepting. Such a machine is understood to only produces outputs for inputs that, once completely processed, leave the transducer in an accepting state.

Figure 1 depicts finite state transducers[1] corresponding to actions a and b. The property to be verified is that exactly one process possesses a token at any given time. The essence of this property may be captured by a reward function[2] $R : 2^{\{t,n\}^*} \to \mathbb{R}$ defined such that

$$R(L) = \begin{cases} 0 & \text{if } L \subseteq n^*tn^*, \\ -1 & \text{otherwise.} \end{cases}$$

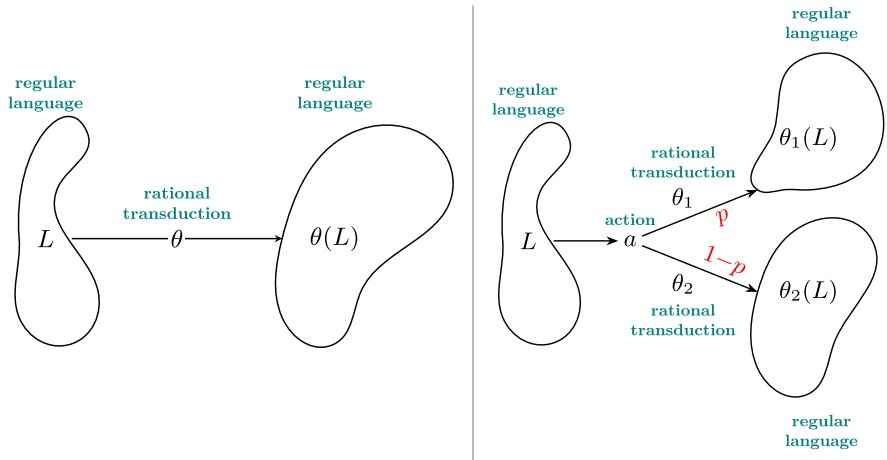
From the initial configuration tn^*, action a moves the system to state ntn^* and incurs a reward of 0, while action b transitions the system to the configuration ttn^* and incurs a reward of -1. The optimal policy selects action a when the token is with a process with an even index, and chooses action b otherwise.

In RRL, the agent initially chooses an action that it deems appropriate for the state tn^*. The environment then returns a language obtained by applying either transducer T_a or transducer T_b to transform tn^*, depending on the agent's choice. From tn^*, the two possible languages are ntn^* and ttn^*. The environment also assigns a reward to the agent. Repeated interactions of this type result in a sequence of states (regular languages) and rewards. The goal of the agent is to learn a policy (a function from regular languages to actions) that maximizes the cumulative reward.

Since there are infinitely many regular languages, the system described in Example 1 gives rise to an infinite-state decision process. As there is no known convergent RL algorithm for infinite-state environments in general, this prohibits the direct use of tabular RL algorithms for RRL. In regular model checking, techniques exist to address the difficulties of dealing with infinite state spaces, such as widening and acceleration. In regular reinforcement learning, we will leverage advances in graph neural networks to tackle this challenge.

[1] Note that the transducers in Fig. 1 are designed under the implicit assumption that their input strings will contain at most one t symbol. This is because there is no way of removing tokens from the system once they have been introduced, and, as a result, no learning can occur after a second token is introduced.

[2] Since containment is decidable for regular languages, R is computable.



(2.1) An RMC transition. (2.2) An RMDP transition.

Fig. 2. Illustration of the difference between transitions in RMC and RMDPs.

Contributions. As in RMC, the primary application of language-theoretic modeling in RRL is the symbolic representation of states and transitions in the underlying system. We formalize RL environments that are constructed according to this principle under the name *regular Markov decision processes* (RMDPs). These environments generalize the systems modeled in RMC by incorporating controllable dynamics (through the agent selecting actions) and stochastic transition dynamics. Figure 2 provides a visual depiction of the similarities and differences between system transitions in RMC vs. RRL.

We provide a theoretical analysis of various aspects of RMDPs, focussing on issues related to decidability, finiteness, and approximability of optimal policies. This clarifies the basic limits of RRL and helps in determining when standard RL methods can or cannot be adapted to this setting. In particular, we establish the following results in Sect. 4.

- The optimal expected payoff, known as the value, of a given RMDP under an arbitrary payoff function is not computable.
- For any RMDP with computable rewards and transition probabilities, the value under a discounted payoff is approximately computable.
- For any RMDP with computable rewards and transition probabilities, the value under a discounted payoff is PAC-learnable.
- We identify several conditions under which an RMDP remains finite and present a Q-learning algorithm for such situations.

After this, we turn our attention toward practical applications of RRL. In Sect. 5 we propose a formulation of deep RRL. By representing regular languages as finite-state automata and viewing automata as labeled directed graphs, we are able to exploit graph neural networks for approximating optimal values and policies. Graph neural networks [56] are neural network architectures that process graphs as input, typically by performing repeated message passing of vectors over the graph's structure. We demonstrate through a collection of experimental case studies that deep RRL is both natural and effective.

2 Related Work

Regular Model Checking. RMC [3,5,18,55] is a verification framework based on symbolically encoding system states and transitions as regular languages and rational transductions, respectively. Despite the relative simplicity of rational transductions, allowing their arbitrary iteration produces a Turing-complete model of computation. Consequently, significant effort has been put into methods to approximate the transitive closures of rational transductions, and to compute them exactly in special cases [17,38,52].

In particular, incorporating automata learning techniques into the RMC toolbox [33,43,45] has shown promise. There is also significant work on improving the framework's expressive capabilities by extending RMC to enable the use of ω-regular languages [13,40,41], regular tree languages [4,6,16,19], and more powerful types of transductions [26]. RMC and its various extensions have been successfully applied to verification safety and liveness properties in a variety of settings related to mutual exclusion protocols, programs operating on unbounded data structures [14,15], lossy channel systems [8], and additive dynamical systems over numeric domains [11,12]. To the best of our knowledge, this paper is the first to combine deep reinforcement learning with regular model checking.

Regular Languages and Reinforcement Learning. The use of regular languages in RL has become increasingly popular to meet the increasing demand for structured, principled representations in neuro-symbolic artificial intelligence. The work closest to our own employs regular languages as a mechanism for modeling aspects of environments with certain kinds of non-Markovian, or history-dependent, dynamics.

Regular Decision Processes [20,21] are the topic of a recent line of research at the intersection of language-theoretic regularity and sequential optimization. A regular decision process is a finite state probabilistic transition system—much like a traditional Markov decision process (MDP)—except that transition probabilities and rewards are dependent on some regular property of the history. Note that while regular decision processes provide a succinct modeling framework for a subclass of non-Markovian optimization problems, they can be converted to larger, but semantically equivalent, finite-state MDPs. In contrast, the RMDPs introduced in this paper are not generally equivalent to finite MDPs. Considerable work has been done to develop the theory and practice around regular deci-

sion processes, including design and analysis of inference algorithms [1], learning efficiency analysis [47], and empirical evaluations of specific modeling tasks [42].

Reward Machines [34–36] are finite state machines used in modeling reward signals in decision processes with non-Markovian, but regular, reward dynamics Attention to the topic has resulted in inference algorithms for learning reward machines in partially observable MDPs [37], methods for jointly learning reward machines and corresponding optimal policies [57], adaptations of active grammatical inference algorithms like $L^\star$ for reward machine inference [58], generalization to probabilistic machines modeling stochastic reward signals [24,29], applications to robotics [22], and more [25,59].

3 Preliminaries

Let $\mathbb{N}$ and $\mathbb{R}$ denote, respectively, the natural numbers and the real numbers. For a set X, we write 2^X to denote its powerset and $|X|$ to denote its cardinality.

3.1 Regular Languages

An alphabet Σ is a finite set of symbols, and a word w over Σ is a finite string of its symbols. The length $|w|$ of a word w is the number of its constituent symbols. The empty word, of length 0, is denoted by ε. We write Σ^n for the set of all words of length n. Further, let $\Sigma^{\leq n} = \bigcup_{k=0}^{n} \Sigma^k$ be the set of all strings of length at most n and let $\Sigma^* = \bigcup_{n=0}^{\infty} X^n$ be the set of all words over Σ. A subset $L \subseteq \Sigma^*$ is a called a language.

Definition 1 (FSA). *A (nondeterministic) finite-state automaton (FSA) $\mathcal{A}$ is given by a tuple $\langle \Sigma, Q, q_0, F, \delta \rangle$, where Σ is an alphabet, Q is a finite set of states, $q_0 \in Q$ is a distinguished initial state, $F \subseteq Q$ is a set of final states, and $\delta : Q \times \Sigma \to 2^Q$ is a transition function.*

The transition function δ may be extended to $\delta^* : Q \times \Sigma^* \to 2^Q$ such that $\delta^*(q, \varepsilon) = \{q\}$ and $\delta^*(q, \sigma w) = \bigcup_{q' \in \delta(q,\sigma)} \delta^*(q', w)$. The semantics of an FA $\mathcal{A}$ are given by a language

$$L_{\mathcal{A}} = \{w \in \Sigma^* : \delta^*(q_0, w) \cap F \neq \emptyset\},$$

and we say that $\mathcal{A}$ recognizes *$L_{\mathcal{A}}$.*

A language is *regular* if it is recognized by an FSA.

3.2 Rational Transductions

Let Σ and Γ be alphabets. A mapping $\theta : \Sigma^* \to 2^{\Gamma^*}$, or equivalently a relation over $\Sigma^* \times \Gamma^*$, is called a *transduction*. For a language L and a transduction $\theta : \Sigma^* \to 2^{\Gamma^*}$, let $\theta(L) = \bigcup_{x \in L} \theta(x)$. The domain of $\theta : \Sigma^* \to 2^{\Gamma^*}$ is given as $\mathrm{dom}(\theta) = \{x \in \Sigma^* : \theta(x) \neq \emptyset\}$ and its image is defined as $\mathrm{im}(\theta) = \bigcup_{x \in \Sigma^*} \theta(x)$.

Given a finite set of transductions Θ with type $\Sigma^* \to 2^{\Sigma^*}$, each finite word $\theta_1 \dots \theta_n \in \Theta^*$ corresponds to the transduction $\theta_n \circ \dots \circ \theta_1$ where ε represents the identity mapping, i.e. $\varepsilon(x) = x$ for every $x \in \Sigma^*$. For convenience, we identify the word $\theta_1 \dots \theta_n$ with the transduction $\theta_n \circ \dots \circ \theta_1$ so that $\theta_1 \dots \theta_n(x) = \theta_n \circ \dots \circ \theta_1(x)$ holds for every $x \in \Sigma^*$. The set of languages reachable from a given language L via elements of Θ^* is called the *orbit* of Θ on L and is written as

$$\mathrm{Orb}_\Theta(L) = \{\tau(L) : \tau \in \Theta^*\}.$$

Definition 2 (FST). *A (nondeterministic) finite-state transducer (FST) T is given by a tuple $\langle \Sigma, \Gamma, Q, q_0, F, \delta \rangle$, where*

- *Σ and Γ are input and output alphabets, respectively,*
- *Q is a finite set of states,*
- *$q_0 \in Q$ is a distinguished initial state,*
- *$F \subseteq Q$ is a set of final states, and*
- *$\delta : Q \times \Sigma \to 2^{Q \times \Gamma^*}$ is a transition function that maps each state-input pair to a set of state-output pairs.*

The transition function δ may be extended to $\delta^* : Q \times \Sigma^* \to 2^{Q \times \Gamma^*}$ such that $\delta(q, \varepsilon) = \{\langle q, \varepsilon \rangle\}$ and $\delta^*(q, \sigma x) = \{\langle q_2, yz \rangle : \langle q_1, y \rangle \in \delta(q, \sigma) \wedge \langle q_2, z \rangle \in \delta^*(q_1, x)\}$. The semantics of T are the transduction $[\![T]\!] : \Sigma^* \to 2^{\Gamma^*}$ defined by

$$[\![T]\!](x) = \{y \in \Gamma^* : \exists q \in F.\ \langle q, y \rangle \in \delta^*(q_0, x)\}.$$

A *rational transduction* θ is one for which there exists an FST T such that $\theta = [\![T]\!]$. A *rational function* θ is a rational transduction such that $|\theta(x)| \leq 1$ for all $x \in \Sigma^*$. When discussing rational functions, we write the type as $\Sigma^* \to \Gamma^*$.

Remark 1. While the title of *regular language* has become standard terminology, there is no universally adopted vocabulary for their relational counterparts. The transductions we qualify here as *rational* are sometimes qualified alternatively in related work with terms such as *regular*, *FST-definable*, *GSM-definable*, etc.

3.3 Markov Decision Processes

Let $\mathrm{Dist}(X)$ be the family of all probability distributions over a set X.

Definition 3 (MDP). *A Markov decision process (MDP) M is presented by a tuple $\langle S, \hat{s}, A, p, r \rangle$, where*

- *S is a set of states,*
- *$\hat{s} \in S$ is a distinguished initial state,*
- *A is a set of actions,*
- *$p : S \times A \to \mathrm{Dist}(S)$ is a probabilistic transition function, and*
- *$r : S \times A \to \mathbb{R}$ is a reward function.*

For any states $s, t \in S$ and action $a \in A$, we write $p(t \mid s, a)$ as a shorthand for $p(s, a)(t)$. We call an MDP *finite* if both S and A are finite sets.

A *policy* over an MDP $M = \langle S, \hat{s}, A, p, r \rangle$ is a history-dependent function that determines how the next action is stochastically chosen. More formally, a policy is defined as a mapping $\pi : S\,(AS)^* \to \mathrm{Dist}\,(A)$ from the domain of interaction histories to probability distributions over the action space. Let $\mathit{\Pi}_M$ be the set of all policies over the MDP M. Fixing a policy π on M induces a family of probability distributions $\{\mathbb{P}^n_\pi : n \in \mathbb{N}\}$ on histories $h = s_1 a_1 \ldots s_n a_n$ with $s_1 = \hat{s}$, where $\mathbb{P}^n_\pi(h) = \prod_{k=1}^{n-1} p(s_{k+1} \mid s_k, a_k)\pi(a_k \mid s_1 a_1 \ldots a_{k-1} s_k)$. There exists a unique extension $\mathbb{P}_\pi \in \mathrm{Dist}\,((SA)^\omega)$ of the family $\{\mathbb{P}^n_\pi : n \in \mathbb{N}\}$ and a corresponding expectation $\mathbb{E}_\pi$.

An *objective* over an MDP M with states S and actions A is a real-valued function J over the domain of infinite real sequences. Whenever the function $\mathsf{J} \circ r$ is $\mathbb{P}_\pi$-measurable, the expectation $\mathbb{E}_\pi(\mathsf{J}) = \int \mathsf{J} \circ r \; \mathrm{d}\mathbb{P}_\pi$ is well-defined and can be used to evaluate the quality of the policy π with respect to the environment M. The J-*value* of M is defined as $\mathrm{Val}_\mathsf{J}(M) = \sup_{\pi \in \mathit{\Pi}_M} \mathbb{E}_\pi(\mathsf{J})$.

Let J be a fixed objective function.

- The J-*value problem* asks, given as input (i) an MDP M and (ii) a lower bound b, to decide whether the inequality $b \leq \mathrm{Val}_\mathsf{J}(M)$ holds.
- The J-value is *computable* if, and only if, there is an algorithm that, given an MDP M as input, returns $\mathrm{Val}_\mathsf{J}(M)$.
- The J-value is *approximable* if, and only if, there exists an algorithm that, given as input (i) an MDP M and (ii) a tolerance $\epsilon > 0$, returns a value V such that $|\mathrm{Val}_\mathsf{J}(M) - V| \leq \epsilon$.

4 Regular Markov Decision Processes

Regular Markov decision processes (RMDPs) are MDPs where states have been provided with a specific structure expressed through a regular language over some alphabet Σ. An execution of an RMDP starts with an initial regular language $L_0 = I$. At each step $i \geq 0$, a decision maker or learning agent selects an action a_i from the current state L_i. The environment resolves the action by selecting a transduction θ_i from the probabilistic distribution over Θ corresponding to the action and returning the next state as $L_{i+1} = \theta_i(L_i)$ and returning the reward $\boldsymbol{r}(L_i)$. The goal of the agent is to learn a policy for selecting actions in a manner that optimizes the value of a given objective J in expectation.

Definition 4 (RMDP). *A* regular Markov decision process *(RMDP)* $\boldsymbol{R}$ *is given by a tuple* $\langle \Sigma, I, \Theta, A, \boldsymbol{p}, \boldsymbol{r} \rangle$*, where*

- Σ *is an alphabet,*
- $I \subseteq \Sigma^*$ *is an initial regular language,*
- Θ *is a finite set of rational transductions with type* $\Sigma^* \to 2^{\Sigma^*}$*,*
- A *is a finite set of actions,*

– $\boldsymbol{p} : 2^{\Sigma^*} \times A \to \mathrm{Dist}\,(\Theta)$ *is a mapping from language-action pairs to distributions over* Θ*, and*
– $\boldsymbol{r} : 2^{\Sigma^*} \to \mathbb{R}$ *is a bounded reward function.*

Semantically, $\boldsymbol{R}$ *is interpreted as a countable MDP* $[\![\boldsymbol{R}]\!] = \langle S, \hat{s}, A, p, r \rangle$*. The state set is defined as* $S = \mathrm{Orb}_\Theta\,(I)$ *with initial state* $\hat{s} = I$*, and the transition and reward functions are such that the equations*

$$p(\theta(L) \mid L, a) = \boldsymbol{p}(\theta \mid L, a) \qquad \text{and} \qquad r(L, a) = \boldsymbol{r}(L),$$

hold for all languages $L \in S$*, actions* $a \in A$*, and transductions* $\theta \in \Theta$*. The value of an objective* J *over a RMDP* $\boldsymbol{R}$ *is defined as* $\mathrm{Val}_\mathsf{J}\,(\boldsymbol{R}) = \mathrm{Val}_\mathsf{J}\,([\![\boldsymbol{R}]\!])$*.*

An RMDP $\boldsymbol{R}$ is called *finite* if the orbit $\mathrm{Orb}_\Theta\,(I)$, is finite. An RMDP is said to be *computable* if the transition probability map $\boldsymbol{p}$ and the reward function $\boldsymbol{r}$ are computable.

4.1 Undecidability of Values

Our first theoretical result establishes that value problems for RMDPs are generally undecidable.

Theorem 1. *Determining whether an arbitrary RMDP satisfies any fixed non-trivial property is undecidable.*

Proof. We construct, as depicted in Fig. 3, a deterministic FST that can simulate the transition relation of an arbitrary Turing machine (TM). Configurations of the TM, i.e. combinations of internal state and tape contents, are encoded as words in the regular language $(0+1)\,(0+1)^*\,Z\,(0+1)^*$, where Z is the finite set of internal states. The index i of the single element of Z occurring in each such word represents that the tape head of the TM is at position $i-1$. Assume that the TM in question includes an arbitrary transition instruction according to the following pair of rules.

– If 0 is read in state z, then write b_0, go to state z_0, and move the tape head.
– If 1 is read in state z, then write b_1, go to state z_1, and move the tape head.

We leave the direction of the tape head shift undetermined and show all possibilities in Fig. 3. The red edges show the construction for the above TM transition when the tape head shifts left. The blue edges show the construction when the tape head shifts right.

In combination with Rice's theorem [46]—which states that no non-trivial[3] property is decidable for the class of Turing machines— this construction implies the desired result. □

It follows from Theorem 1 that optimal values of RMDPs are not computable in general.

Corollary 1. *Under any objective, the RMDP value problem is undecidable.*

[3] Using Rice's terminology, a property is trivial with respect to class of models if it holds for all models in the class or if it holds for no models in the class.

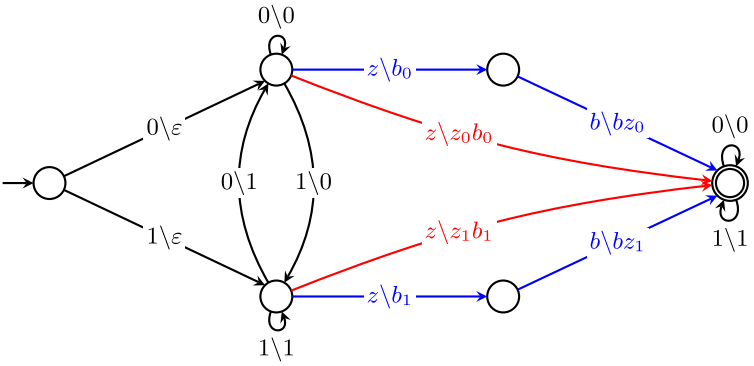


Fig. 3. The FST from the proof of Theorem 1, simulating the transition function of a Turing machine over a binary alphabet.

4.2 Discounted Optimization

We now consider RMDPs under discounted objectives. Let $x = x_1, x_2, \ldots$ be a bounded infinite sequence of real numbers. Given a discount factor $\lambda \in [0, 1)$, the λ-*discounted objective* D_λ is defined as

$$\mathsf{D}_\lambda(x) = \sum_{n=1}^{\infty} x_n \lambda^{n-1}.$$

Over computable RMDPs, it is possible to approximate the D_λ-value to an arbitrary tolerance. This result is facilitated by properties of the discounted objective, and therefore holds even when the RMDP in question is not finite. The proof uses the standard technique of finding, given a tolerance ϵ and a discount factor λ, the least n such that

$$\left| \mathsf{D}_\lambda(x) - \sum_{k=1}^{n} \lambda^{k-1} x_k \right| \leq \epsilon.$$

Theorem 2. *If $\boldsymbol{R}$ is a computable RMDP, then the D_λ-value is ϵ-approximable, for any $\lambda \in [0, 1)$ and any $\epsilon > 0$.*

Proof. For a given RMDP $\boldsymbol{R}$, the D_λ-value can be characterized by the following Bellman optimality equation:

$$D(L) = \max_{a \in A} \left\{ \boldsymbol{r}(L) + \lambda \sum_{\theta \in \Theta} \boldsymbol{p}(\theta \mid L, a) D(\theta(L)) \right\}.$$

It follows from the more general result on MDPs with countable state space, finite action space, and bounded reward [30]. Let b be an upper bound on the absolute value of the rewards. For a given $\epsilon > 0$, let n be such that

$$\frac{\lambda^{n+1} b}{1 - \lambda} \leq \epsilon.$$

Then a solution to the following recurrence characterizes an ε-optimal value and corresponding memoryful policy for the RMDP:

$$D_n(L) = \begin{cases} \max_{a \in A} \left\{ \boldsymbol{r}(L) + \lambda \sum_{\theta \in \Theta} \boldsymbol{p}(\theta \mid L, a) D_{n-1}(\theta(L)) \right\} & \text{if } n > 0 \\ 0 & \text{otherwise.} \end{cases}$$

The proof is now complete. □

An RL algorithm is *probably approximately correct* (PAC) [53], with respect to parameters $\epsilon > 0$ and $\delta > 0$, if after polynomially many samples of the environment, it produces an ϵ-optimal policy with probability $1 - \delta$. Objective functions under which PAC algorithms exist are called PAC-learnable.

Theorem 3. *For every RMDP, the* D_λ*-value is PAC-learnable.*

Proof. Our approach for calculating ϵ-optimal policies for the discounted objective involves computing a policy that is optimal for a fixed number of steps, denoted by n. Given $\epsilon > 0$, we choose n such that

$$\frac{\lambda^{n+1} b}{1 - \lambda} \leq \epsilon.$$

This policy can be computed on a finite-state MDP obtained by unfolding the given RMDP n times. We can then apply existing PAC-MDP algorithms [7] to compute an $\frac{\epsilon}{2}$-optimal policy, which is also an ϵ-optimal policy for the RMDP. □

4.3 Finiteness Conditions

In this section, we provide three sufficient conditions to guarantee finiteness of the RMDP. Fix an arbitrary RMDP $\boldsymbol{R} = (\Sigma, I, \Theta, A, \boldsymbol{p}, \boldsymbol{r})$ with semantics $[\![\boldsymbol{R}]\!] = (S, \hat{s}, A, p, r)$.

Word-Based Condition. A transduction $\theta : \Sigma^* \to 2^{\Gamma^*}$ is (i) *length-preserving* if $|\theta(x)| = |x|$, (ii) *decreasing* if $|\theta(x)| < |x|$, (iii) *non-increasing* if $|\theta(x)| \leq |x|$, (iv) *non-decreasing* if $|\theta(x)| \geq |x|$, (v) *increasing* if $|\theta(x)| > |x|$, for all $x \in \Sigma^*$.

Proposition 1. *If* I *is a finite language,* Θ *is non-increasing,* $|\Sigma| = n$*, and* $\max_{x \in I} |x| = m$*, then* $\mathrm{Orb}_\Theta(I) \in 2^{O(n^m)}$.

Proof. The statement can be derived from the observation that the longest string possibly appearing in the image $\theta(I)$ of a finite language I under a non-increasing transformation θ is of length $m = \max_{w \in I} |w|$. There are n^m strings of length m over an alphabet Σ of size n, so it follows that $|\theta(I)| \leq 1 + \sum_{k=1}^{m} n^k$. More succinctly, this says that $|\theta(I)| = O(n^m)$. Since $\mathrm{Orb}_\Theta(I)$ must comprise some collection of subsets of $\Sigma^{\leq m}$, we conclude that $|\mathrm{Orb}_\Theta(I)| = 2^{O(n^m)}$. □

Language-Based Condition. A transduction $\theta : \Sigma^* \to 2^{\Sigma^*}$ is (i) *specializing* if $\theta(L) \subseteq L$, (ii) *non-specializing* if $\theta(L) \not\subseteq L$, (iii) *generalizing* if $L \subseteq \theta(L)$, (iv) *non-generalizing* if $L \not\subseteq \theta(L)$, for all $L \subseteq \Sigma^*$.

Proposition 2. *If* $|I| = n$ *and* Θ *is specializing, then* $\mathrm{Orb}_\Theta(I) \leq 2^n$.

Proof. This result can be deduced from the observation that when beginning with a finite initial language, specializing transformations can only generate languages with a cardinality that is either equal to or smaller than that of I. □

Reward-Based Condition. Let $\sim_{\boldsymbol{R}} \subseteq 2^{\Sigma^*} \times 2^{\Sigma^*}$ be an equivalence relation over languages such that $L_1 \sim_{\boldsymbol{R}} L_2$ if, and only if,

$$\boldsymbol{r}(L_1) = \boldsymbol{r}(L_2) \quad \text{and} \quad \forall \theta \in \Theta. \quad \theta(L_1) \sim_{\boldsymbol{R}} \theta(L_2).$$

This relation is often useful as a means of partitioning the state space of an RMDP into a finite set of equivalence classes that respects the structure of its dynamics. For instance, it is straightforward to deduce the following proposition.

Proposition 3. *If there exists* $n \in \mathbb{N}$ *such that* $\boldsymbol{r}(L) = \boldsymbol{r}\left(L \cap \Sigma^{\leq n}\right)$ *holds for every* $L \subseteq \Sigma^*$, *then* $\mathrm{Orb}_\Theta(I)$ *has finitely many* $\sim_{\boldsymbol{R}}$*-equivalence classes.*

4.4 Q-Learning in RMDPs

We have discussed some conditions that ensure the finiteness of $[\![M]\!]$. When any such condition is satisfied, it becomes feasible to employ off-the-shelf RL algorithms for discounted optimization. Equation (1)—in which $[L]_\sim$ denotes the equivalence class of $\sim_{\boldsymbol{R}}$ to which the language L belongs—provides an iteration scheme for a variation on the Q-learning [54] algorithm tailored for RMDPs. If learning rates $(\alpha_n)_{n\in\mathbb{N}}$ are such that $\sum_{n=1}^{\infty} \alpha_n = \infty$ and $\sum_{n=1}^{\infty} \alpha_n^2 < \infty$, and the trajectory $([L_n]_\sim, a_n, r_n)_{n\in\mathbb{N}}$ includes each pair $[L]_\sim, a$ infinitely often, then iterating Eq. (1) converges almost surely to an optimal policy.

$$Q_{n+1}([L_n]_\sim, a_n) \coloneqq (1-\alpha_n)\, Q_n\left([L_n]_\sim, a_n\right) + \alpha_n \left(r_n + \max_{a\in A} Q_n\left([L_{n+1}]_\sim, a\right)\right) \quad (1)$$

5 Deep Regular Reinforcement Learning

Generally speaking, RMDPs may have infinite state spaces, so we cannot guarantee convergence of Q-learning. In light of this fact and the uncomputability of exact discounted values—established by Theorem 1—it makes sense to consider approximate learning methods. Accordingly, we propose a deep learning approach based on using *graph neural networks* (GNNs). Our key insight in this context is the observation that we can use automaton representations of the states of an RMDP directly as inputs to GNNs. Much like standard deep RL uses

feature vectors as inputs for neural networks, this technique uses automata—which are essentially finite labeled graphs—as inputs for GNNs. We term this approach *deep regular reinforcement learning*.

Before presenting experimental results, we describe the overall architecture of our learning scheme in the next subsection.

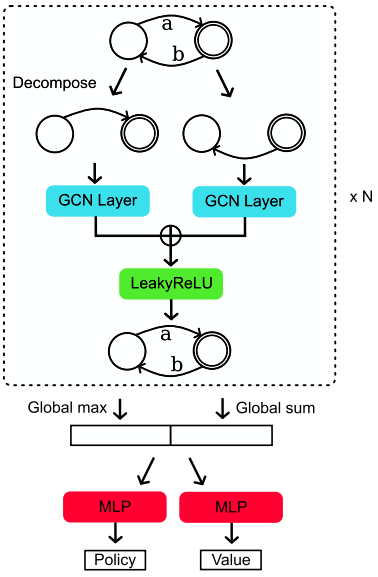


Fig. 4. Deep regular reinforcement learning architecture.

Our graph neural network architecture, is based on the graph convolution operator proposed by Kipf & Welling [39]. We perform an independent graph convolution for each letter in the input automaton— only allowing the convolution to operate over the graph connectivity for that letter—and take the mean of the resulting vectors for each node, followed by a nonlinearity. We repeat this for N layers and then concatenate the sum of all node vectors with the element-wise maximum of all node vectors. Separate multi-layer perceptrons produce the policy and state value predictions from this representation. We use proximal policy optimization (PPO) [48] for training. Figure 4 outlines this architecture. The initial embedding for each node in the automaton is a binary vector of length two, which encodes whether a node is the initial state and if a node is accepting. For all experiments, the graph neural network had 3 graph convolution layers with hidden dimension 256, and the multi-layer perceptron heads had 2 layers each with hidden dimension 256. We used the LeakyReLU nonlinearity.

In the remainder of this section, we present specific examples of regular RL problems and provide experimental results to illustrate the effectiveness of deep regular reinforcement learning.

5.1 Token Passing

We first consider the token passing scenario of Example 1 (cf. Fig. 1). Note that this example admits a partitioning of the environment via the equivalence relation defined in Sect. 4.3: there are two equivalence classes, corresponding to whether there are an even or an odd number of n symbols before the first t symbol. We compare using the GNN on the original representation (GNN) and on the representation formed by the two equivalence classes (GNN+EC). Figure 5 shows the FSA representations used for the two equivalence classes.

(5.1) even automaton (5.2) odd automaton

Fig. 5. Automata used to represent even and odd equivalence classes.

The hyperparameters we used for PPO in this case study were 1024 steps per update, a 512 batch size, 4 optimization epochs, a clip range of $\epsilon = 0.2$, and a discount factor of $\lambda = 0.99$. The learning curves[4] are shown in Fig. 6.

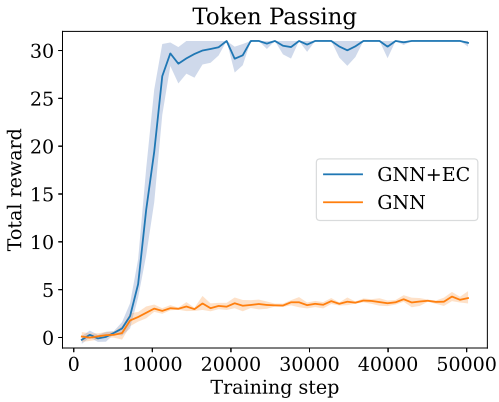


Fig. 6. Reward curves for the token passing case study.

Note that under the selected network architecture, determining whether the number of n symbols occurring before the first t symbol is even or odd is largely determined by the multi-layer perceptron components. Roughly, the number of

[4] Here, the dark lines are means and the shaded regions are the 10^{th} to 90^{th} percentiles over 5 training runs. All subsequent reward curves should be read this way as well.

n symbols before the first t symbol is encoded in unary in the global sum component of the graph representation. The multi-layer perceptrons must then detect parity on this unary representation, which is challenging. To encourage learning, we use a denser reward of 1 on every successful step and -1 on failure, up to a time limit of 30. Although alternative network architectures have the potential to perform better, the simple two-state equivalence class representation is still expected to result in faster learning than the unmodified representation. The learning run is shortened, and the maximum episode length is set to 30 to highlight the difference between these two setups. We see that forming equivalence classes leads to an increase in learning speed.

5.2 Duplicating Pebbles

Consider a grid world with multiple pebbles on it. The agent can select two adjacent directions, e.g., "up" and "right", and every pebble will be duplicated and moved in each of these directions. The goal of the agent is to have at least one pebble reach the goal state, while all pebbles do not accumulate a cost greater than a threshold $t = 2$. If a pebble goes over a trap cell, it incurs a cost of 1 for that pebble.

Although the number of pebbles grows exponentially, doubling after each action, the set of paths the pebbles take has an FSA representation. Namely, one can represent the growing paths by adding a state to the FSA with two transitions to the state corresponding to the two directions selected. This added state is marked as the only accepting state. The language of this FSA corresponds to all paths that pebbles have taken. A reward of -1 is given on failure and a reward of 1 is given on success. The grid layout is shown in Fig. 7, where the initial pebble begins in the top left. The agent learns the optimal policy "down, right", "down, right", "up, left", "up, left" in about 10k training steps. Figure 7 shows the execution of this optimal policy, from left to right, top to bottom. Traps are denoted by red cells and the goal is denoted by a green cell. The number in a cell counts the number of pebbles it contains.

The hyperparameters we used for PPO in this case study were 512 steps per update, a 128 batch size, 4 optimization epochs, a clip range of $\epsilon = 0.2$, and a discount factor of $\lambda = 0.99$. The resulting reward curve is shown in Fig. 8.

Since the representation of the state is an FSA of the possible trajectories, a linear program is solved at each step to find the highest cost path needed for computing the reward. Note that when "up, left" is first performed, some pebbles wrap around to the opposite side of the grid. If "down, right" was performed 3 times, instead of twice, then the agent would fail the objective since the 2 pebbles at $(1, 2)$ on the grid would duplicate and visit the trap state again after having already visited it once.

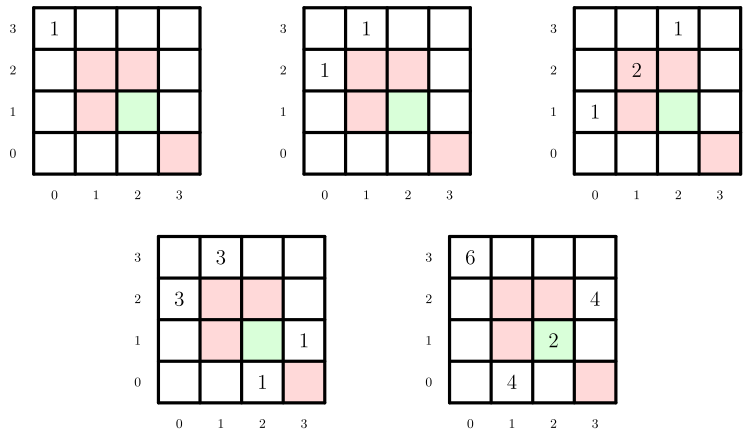


Fig. 7. Execution of the optimal policy for the duplicating pebbles case study.

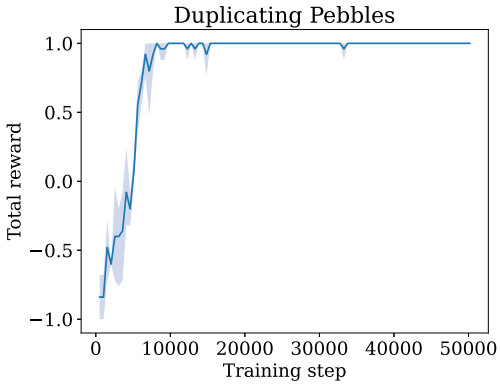


Fig. 8. Reward curve for the duplicating pebbles case study.

5.3 Shunting Yard Algorithm

To showcase representation of unbounded data structures like stacks and queues as a strength of RRL, we consider learning the shunting yard algorithm [28] which transforms an expression in infix notation to postfix notation.

We represent the input as a regular language consisting of a single string containing the concatenation of the infix notation input, the stack, and the output, each separated by a special symbol "#". The agent has three actions:

- moving the first character of the input to the output,
- pushing the first character of the input to the stack, and
- popping the top character on the stack to the output.

We generate random infix notation expressions and give a reward of −1 if the output is an invalid postfix expression, a reward of 0.5 if the output is a valid

State	Action
3*3+7+5##	Move
*3+7+5##3	Push
3+7+5#*#3	Move
+7+5#*#33	Pop
+7+5##33*	Push
7+5#+#33*	Move
+5#+#33*7	Pop
+5##33*7+	Push
5#+#33*7+	Move
#+#33*7+5	Pop
##33*7+5+	

State	Action
1+7*8+0##	Move
+7*8+0##1	Push
7*8+0#+#1	Move
*8+0#+#17	Push
8+0#+*#17	Move
+0#+*#178	Pop
+0#+#178*	Pop
+0##178*+	Push
0#+#178*+	Move
#+#178*+0	Pop
##178*+0+	

State	Action
7-1*3##	Move
-1*3##7	Push
1*3#-#7	Move
*3#-#71	Push
3#-*#71	Move
#-*#713	Pop
#-#713*	Pop
##713*-	

Fig. 9. Runs produced by the learned policy for the shunting yard algorithm.

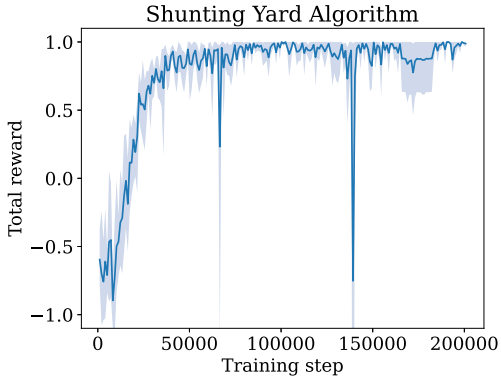


Fig. 10. Reward curve for the shunting yard algorithm case study.

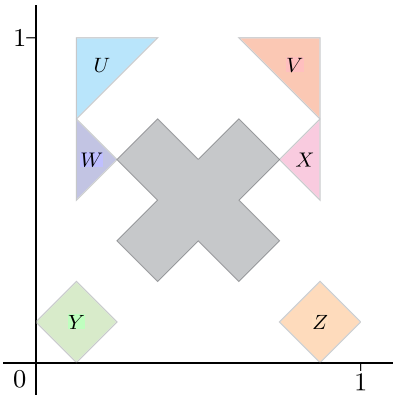
postfix expression that evaluates to the wrong value, and a reward 1 if the output evaluates to the correct value. The agent is able to learn an effective strategy in about 100k time steps.

Figure 9 shows example runs produced by the learned policy. The representation used during learning is an FSA accepting a single string, but we print the string for clarity. Actions are the actions selected upon observing that state. The last state is the final state at termination.

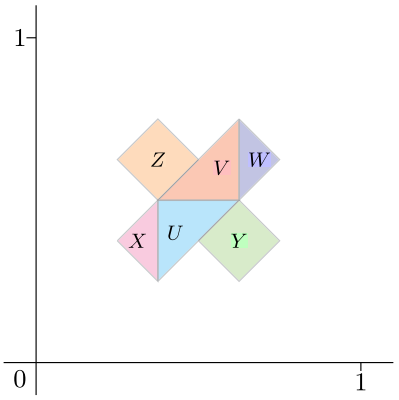
The hyperparameters we used for PPO in this case study were 1024 steps per update, a 128 batch size of, 10 optimization epochs, a clip range of $\epsilon = 0.2$, and a discount factor of $\lambda = 0.99$. The resulting reward curve is shown in Fig. 10.

5.4 Modified Tangrams

This case study examines the application of deep RRL to variations of geometric puzzles known as tangrams, which involve arranging a finite set of polygonal



(11.1) Tiles in their initial positions.



(11.2) Tiles arranged in a solution.

Fig. 11. A modified tangram. The goal is to cover the gray shape at the center resembling the symbol "×" by rearranging the colored tiles $\{U, V, W, X, Y, Z\}$. (Color figure online)

tiles on a flat surface to create a picture. The picture is typically a silhouette in the shape of a common object such as a building or a tree, and the puzzle is completed once the tiles have been arranged into a configuration that covers the silhouette exactly. A standard tangram set includes a collection of target pictures, 5 right triangles (2 large, 1 medium, and 2 small), a square, and a parallelogram. We consider modified tangrams, which we qualify as such because the pieces do not coincide with the standard tile set. An example is displayed in Fig. 11.

In order to cast these sorts of puzzles into the RRL framework, we apply standard notions used in positional numeration systems to connect geometric shapes and regular languages. More precisely, tiles are considered as sets of points in the unit square $[0,1] \times [0,1]$ of the Euclidean plane. Then, sets of points are encoded by regular languages consisting of digital expansions of these points.

For a numeration base $b \in \mathbb{N}$, define a map $\langle\!\langle \cdot \rangle\!\rangle_b : \{0, \ldots, b-1\}^* \to (0,1)$ as

$$\langle\!\langle w \rangle\!\rangle_b = \sum_{k=1}^{|w|} \frac{w_k}{b^k}$$

to interpret each string of digits w as a base-b digital expansion (where the leftmost symbol is the most significant bit) of a number $\langle\!\langle w \rangle\!\rangle_b$ in the unit interval. Such interpretations extend to languages so that

$$\langle\!\langle L \rangle\!\rangle_b = \{\langle\!\langle w \rangle\!\rangle_b : w \in L\}.$$

We fix the base as $b = 2$, and consider automata over the alphabet the two-dimensional boolean alphabet $\mathbb{B}^2$ to encode points in the plane. We design automata capturing languages that represent the sets of points included in particular shapes, as illustrated in Fig. 12.

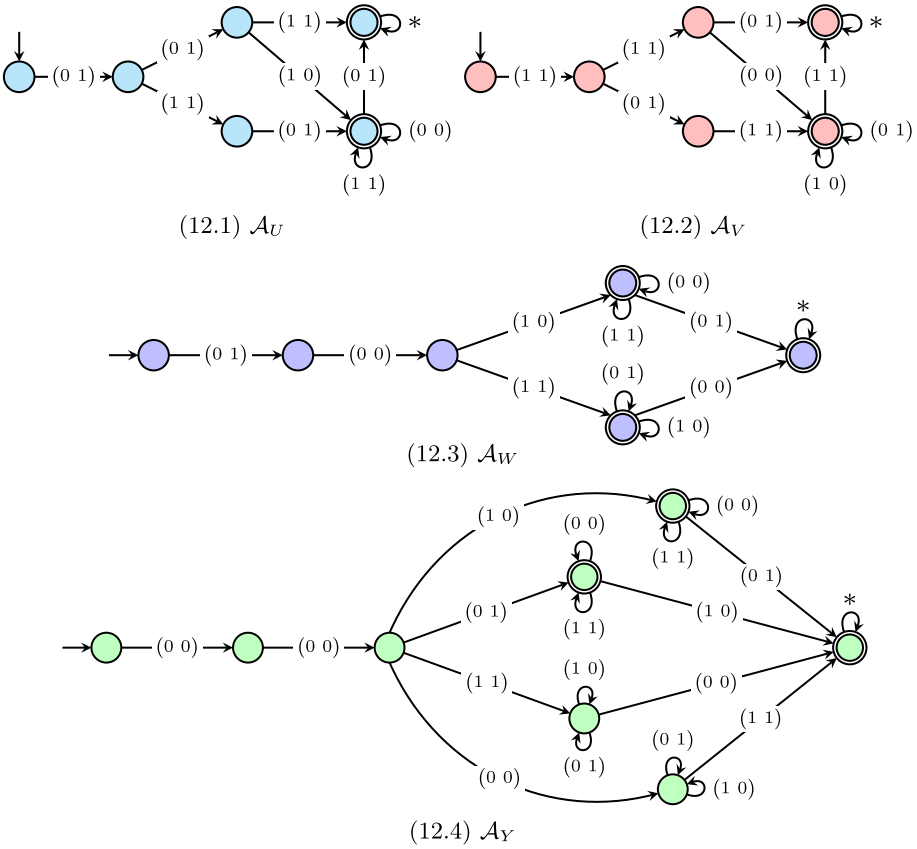
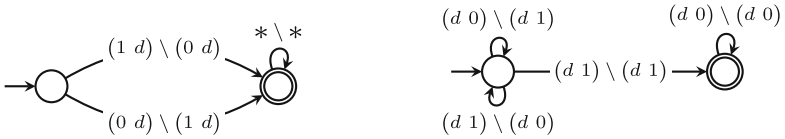


Fig. 12. Automata corresponding to some of the starting tiles shown in Fig. 11.1.

Remark 2. Automata $\mathcal{A}_X$ and $\mathcal{A}_Z$ may be obtained from the automata $\mathcal{A}_W$ (Fig. 12.3) and $\mathcal{A}_Y$ (Fig. 12.4), respectively. This can be done by taking the logical complement of the x-coordinate on every non-looping transition and exchanging pairs of self loops on a common state labeled by $\big(0\ 0\big)$ and $\big(1\ 1\big)$ to ones labeled by $\big(0\ 1\big)$ and $\big(1\ 0\big)$.

We also design finite-state transducers, as illustrated in Fig. 13, for basic geometric operations such as translation by 1/2, translation by 1/4, and reflection across $x = 1/2$ and $y = 1/2$.

The agent's goal is to apply these basic transformations to move each shape from its initial position into the goal region. To reduce the number of actions, the agent selects transformations for one of the shapes at a time and uses a special "submit" action to move to the next shape. We treat the collection of automata as a single nondeterministic FSA, and specially mark the alphabet of the active automaton in the collection. Rewards are proportional to the overlap with the remaining exposed target shape when the submit action is used. On all other steps, a reward of -0.01 is given to encourage promptness.



(13.1) translate right by 1/2 (13.2) reflect about $y = 1/2$

Fig. 13. FSTs implementing some rigid transformations on the unit square. Arbitrary digits are represented by d, while $*$ represents arbitrary pairs of digits.

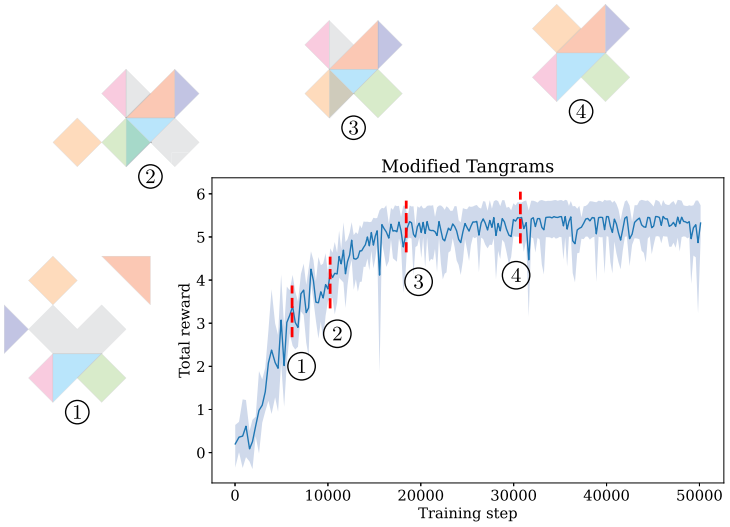


Fig. 14. Annotated reward curve for the modified tangram example.

The hyperparameters used for PPO in this case study were 256 steps per update, a 64 batch size, 10 optimization epochs, a clip range of $\epsilon = 0.1$, and a discount factor of $\lambda = 0.99$. The resulting reward curve—which we annotate at various points to show the agent's progress—is shown in Fig. 14.

6 Conclusion

This paper introduced a framework for symbolic reinforcement learning, dubbed *regular reinforcement learning*, where system states are modeled as regular languages and transition relations are modeled as rational transductions. We established theoretical results about the limitations and capabilities of this framework, proving that optimal values and policies are approximable and efficiently learnable under discounted payoffs. Furthermore, we developed an approach to deep regular reinforcement learning that combines aspects of deep learning and symbolic representation via the use of graph neural networks. Through a variety of case studies, we illustrated the effectiveness of deep regular reinforcement learning.

Acknowledgements. This work was supported in part by the NSF through grant CCF-2009022 and the NSF CAREER award CCF-2146563.

References

1. Abadi, E., Brafman, R.I.: Learning and solving regular decision processes. In: International Joint Conference on Artificial Intelligence, IJCAI, pp. 1948–1954. ijcai.org (2020). https://doi.org/10.24963/ijcai.2020/270
2. Abdulla, P.A.: Regular model checking. Int. J. Softw. Tools Technol. Transfer **14**(2), 109–118 (2012). https://doi.org/10.1007/S10009-011-0216-8
3. Abdulla, P.A.: Regular model checking: evolution and perspectives. In: Olderog, E.-R., Steffen, B., Yi, W. (eds.) Model Checking, Synthesis, and Learning. LNCS, vol. 13030, pp. 78–96. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-91384-7_5
4. Abdulla, P.A., Jonsson, B., Mahata, P., d'Orso, J.: Regular tree model checking. In: Brinksma, E., Larsen, K.G. (eds.) CAV 2002. LNCS, vol. 2404, pp. 555–568. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45657-0_47
5. Abdulla, P.A., Jonsson, B., Nilsson, M., Saksena, M.: A survey of regular model checking. In: Gardner, P., Yoshida, N. (eds.) CONCUR 2004. LNCS, vol. 3170, pp. 35–48. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-28644-8_3
6. Abdulla, P.A., Legay, A., d'Orso, J., Rezine, A.: Tree regular model checking: a simulation-based approach. J. Logic Algebraic Program. **69**(1–2), 93–121 (2006). https://doi.org/10.1016/j.jlap.2006.02.001
7. Agarwal, A., Jiang, N., Kakade, S.M., Sun, W.: Reinforcement learning: Theory and algorithms. CS Department, UW Seattle, Seattle, WA, USA, Technical Report **32**, 96 (2019)
8. Baier, C., Bertrand, N., Schnoebelen, P.: On computing fixpoints in well-structured regular model checking, with applications to lossy channel systems. In: Hermann, M., Voronkov, A. (eds.) LPAR 2006. LNCS (LNAI), vol. 4246, pp. 347–361. Springer, Heidelberg (2006). https://doi.org/10.1007/11916277_24
9. Bause, F., Kritzinger, P.S.: Stochastic Petri Nets - an Introduction to the Theory, 2nd edn. Vieweg (2002)
10. Berner, C., Brockman, G., Chan, B., Cheung, V., Dębiak, P., Dennison, C., Farhi, D., Fischer, Q., Hashme, S., Hesse, C., et al.: Dota 2 with large scale deep reinforcement learning. arXiv preprint arXiv:1912.06680 (2019)
11. Boigelot, B.: On iterating linear transformations over recognizable sets of integers. Theoret. Comput. Sci. **309**(1–3), 413–468 (2003). https://doi.org/10.1016/S0304-3975(03)00314-1
12. Boigelot, B., Legay, A., Wolper, P.: Iterating transducers in the large. In: Hunt, W.A., Somenzi, F. (eds.) CAV 2003. LNCS, vol. 2725, pp. 223–235. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45069-6_24
13. Boigelot, B., Legay, A., Wolper, P.: Omega-regular model checking. In: Jensen, K., Podelski, A. (eds.) TACAS 2004. LNCS, vol. 2988, pp. 561–575. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24730-2_41
14. Bouajjani, A., Habermehl, P., Moro, P., Vojnar, T.: Verifying programs with dynamic 1-selector-linked structures in regular model checking. In: Halbwachs, N., Zuck, L.D. (eds.) TACAS 2005. LNCS, vol. 3440, pp. 13–29. Springer, Heidelberg (2005). https://doi.org/10.1007/978-3-540-31980-1_2
15. Bouajjani, A., Habermehl, P., Rogalewicz, A., Vojnar, T.: Abstract regular tree model checking of complex dynamic data structures. In: Yi, K. (ed.) SAS 2006. LNCS, vol. 4134, pp. 52–70. Springer, Heidelberg (2006). https://doi.org/10.1007/11823230_5

16. Bouajjani, A., Habermehl, P., Rogalewicz, A., Vojnar, T.: Abstract regular (tree) model checking. Int. J. Softw. Tools Technol. Transfer **14**(2), 167–191 (2012). https://doi.org/10.1007/s10009-011-0205-y
17. Bouajjani, A., Habermehl, P., Vojnar, T.: Abstract regular model checking. In: Alur, R., Peled, D.A. (eds.) CAV 2004. LNCS, vol. 3114, pp. 372–386. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-27813-9_29
18. Bouajjani, A., Jonsson, B., Nilsson, M., Touili, T.: Regular model checking. In: Emerson, E.A., Sistla, A.P. (eds.) CAV 2000. LNCS, vol. 1855, pp. 403–418. Springer, Heidelberg (2000). https://doi.org/10.1007/10722167_31
19. Bouajjani, A., Touili, T.: Widening techniques for regular tree model checking. Int. J. Softw. Tools Technol. Transfer **14**(2), 145–165 (2012). https://doi.org/10.1007/s10009-011-0208-8
20. Brafman, R.I., Giacomo, G.D.: Regular decision processes: A model for non-markovian domains. In: International Joint Conference on Artificial Intelligence, IJCAI, pp. 5516–5522. ijcai.org (2019). https://doi.org/10.24963/ijcai.2019/766
21. Brafman, R.I., Giacomo, G.D.: Regular decision processes: modelling dynamic systems without using hidden variables. In: International Conference on Autonomous Agents and MultiAgent Systems, AAMAS, pp. 1844–1846. International Foundation for Autonomous Agents and Multiagent Systems (2019). http://dl.acm.org/citation.cfm?id=3331938
22. Camacho, A., Varley, J., Zeng, A., Jain, D., Iscen, A., Kalashnikov, D.: Reward machines for vision-based robotic manipulation. In: International Conference on Robotics and Automation, ICRA, pp. 14284–14290. IEEE (2021). https://doi.org/10.1109/ICRA48506.2021.9561927
23. Chatterjee, K., Fu, H., Goharshady, A.K., Okati, N.: Computational approaches for stochastic shortest path on succinct MDPs. In: International Joint Conference on Artificial Intelligence, IJCAI, pp. 4700–4707. ijcai.org (2018). https://doi.org/10.24963/ijcai.2018/653
24. Corazza, J., Gavran, I., Neider, D.: Reinforcement learning with stochastic reward machines. In: Conference on Artificial Intelligence, AAAI vol. 36, no. 6, pp. 6429–6436 (2022). https://doi.org/10.1609/aaai.v36i6.20594
25. Dann, M., Yao, Y., Alechina, N., Logan, B., Thangarajah, J.: Multi-agent intention progression with reward machines. In: International Joint Conference on Artificial Intelligence, IJCAI, pp. 215–222. ijcai.org (2022). https://doi.org/10.24963/ijcai.2022/31
26. Dave, V., Dohmen, T., Krishna, S.N., Trivedi, A.: Regular model checking with regular relations. In: Bampis, E., Pagourtzis, A. (eds.) FCT 2021. LNCS, vol. 12867, pp. 190–203. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-86593-1_13
27. Delgado, K.V., Sanner, S., De Barros, L.N.: Efficient solutions to factored MDPs with imprecise transition probabilities. Artif. Intell. **175**(9–10), 1498–1527 (2011). https://doi.org/10.1016/j.artint.2011.01.001
28. Dijkstra, E.W.: Algol 60 translation: An algol 60 translator for the x1 and making a translator for algol 60. Stichting Mathematisch Centrum. Rekenafdeling (MR 34/61) (1961)
29. Dohmen, T., Topper, N., Atia, G.K., Beckus, A., Trivedi, A., Velasquez, A.: Inferring probabilistic reward machines from non-Markovian reward signals for reinforcement learning. In: International Conference on Automated Planning and Scheduling, ICAPS, pp. 574–582. AAAI Press (2022). https://doi.org/10.1609/icaps.v32i1.19844

30. Feinberg, E.A.: Total expected discounted reward MDPs: existence of optimal policies (2011). https://onlinelibrary.wiley.com/doi/abs/10.1002/9780470400531.eorms0906
31. Goodfellow, I.J., Bengio, Y., Courville, A.C.: Deep Learning. Adaptive Computation and Machine Learning, MIT Press (2016). http://www.deeplearningbook.org/
32. Guestrin, C., Koller, D., Parr, R., Venkataraman, S.: Efficient solution algorithms for factored MDPs. J. Artif. Intell. Res. **19**, 399–468 (2003). https://doi.org/10.1613/jair.1000
33. Habermehl, P., Vojnar, T.: Regular model checking using inference of regular languages. In: International Workshop on Verification of Infinite-State Systems, INFINITY, pp. 21–36. Electronic Notes in Theoretical Computer Science, Elsevier (2004). https://doi.org/10.1016/j.entcs.2005.01.044
34. Icarte, R.T.: Reward Machines. Ph.D. thesis, University of Toronto, Canada (2022). http://hdl.handle.net/1807/110754
35. Icarte, R.T., Klassen, T.Q., Valenzano, R.A., McIlraith, S.A.: Using reward machines for high-level task specification and decomposition in reinforcement learning. In: International Conference on Machine Learning, ICML. Proceedings of Machine Learning Research, vol. 80, pp. 2112–2121. PMLR (2018). http://proceedings.mlr.press/v80/icarte18a.html
36. Icarte, R.T., Klassen, T.Q., Valenzano, R.A., McIlraith, S.A.: Reward machines: exploiting reward function structure in reinforcement learning. J. Artif. Intell. Res. **73**, 173–208 (2022). https://doi.org/10.1613/jair.1.12440
37. Icarte, R.T., Waldie, E., Klassen, T.Q., Valenzano, R.A., Castro, M.P., McIlraith, S.A.: Learning reward machines for partially observable reinforcement learning. In: Conference on Neural Information Processing Systems, NeurIPS, pp. 15497–15508 (2019). https://proceedings.neurips.cc/paper/2019/hash/532435c44bec236b471a47a88d63513d-Abstract.html
38. Jonsson, B., Saksena, M.: Systematic acceleration in regular model checking. In: Damm, W., Hermanns, H. (eds.) CAV 2007. LNCS, vol. 4590, pp. 131–144. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-73368-3_16
39. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. In: International Conference on Learning Representations, ICLR. OpenReview.net (2017). https://openreview.net/forum?id=SJU4ayYgl
40. Legay, A.: Extrapolating (omega-)regular model checking. Int. J. Softw. Tools Technol. Transfer **14**(2), 119–143 (2012). https://doi.org/10.1007/s10009-011-0209-7
41. Legay, A., Wolper, P.: On (omega-)regular model checking. ACM Trans. Computat. Logic **12**(1), 2:1–2:46 (2010). https://doi.org/10.1145/1838552.1838554
42. Lenaers, N., van Otterlo, M.: Regular decision processes for grid worlds. In: Leiva, L.A., Pruski, C., Markovich, R., Najjar, A., Schommer, C. (eds.) Benelux Conference on Artificial Intelligence, BNAIC/Benelearn. Communications in Computer and Information Science, vol. 1530, pp. 218–238. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-93842-0_13
43. Lin, A.W., Rümmer, P.: Regular model checking revisited. In: Olderog, E.-R., Steffen, B., Yi, W. (eds.) Model Checking, Synthesis, and Learning. LNCS, vol. 13030, pp. 97–114. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-91384-7_6
44. Mnih, V., et al.: Human-level control through reinforcement learning. Nature **518**, 529–533 (2015)

45. Neider, D., Jansen, N.: Regular model checking using solver technologies and automata learning. In: Brat, G., Rungta, N., Venet, A. (eds.) NFM 2013. LNCS, vol. 7871, pp. 16–31. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38088-4_2
46. Rice, H.G.: Classes of recursively enumerable sets and their decision problems. Trans. Am. Math. Soc. **74**(2), 358–366 (1953)
47. Ronca, A., Giacomo, G.D.: Efficient PAC reinforcement learning in regular decision processes. In: International Joint Conference on Artificial Intelligence, IJCAI, pp. 2026–2032. ijcai.org (2021). https://doi.org/10.24963/ijcai.2021/279
48. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms. CoRR abs/1707.06347 arxiv:1707.06347 (2017)
49. Silver, D., et al.: Mastering the game of Go with deep neural networks and tree search. Nature **529**, 484–489 (2016)
50. Sipser, M.: Introduction to the Theory of Computation, chap. 1. PWS Publishing Company (1997)
51. Sutton, R.S., Barto, A.G.: Reinforcement learning - an Introduction. Adaptive Computation and Machine Learning, MIT Press (1998). https://www.worldcat.org/oclc/37293240
52. Touili, T.: Regular model checking using widening techniques. In: Verification of Parameterized Systems, VEPAS 2001, Satellite Workshop of ICALP, pp. 342–356. Electronic Notes in Theoretical Computer Science, Elsevier (2001). https://doi.org/10.1016/S1571-0661(04)00187-2
53. Valiant, L.G.: A theory of the learnable. In: Symposium on Theory of Computing, STOC, pp. 436–445. ACM (1984). https://doi.org/10.1145/800057.808710
54. Watkins, C.J.C.H., Dayan, P.: Technical note q-learning. Mach. Learn. **8**, 279–292 (1992). https://doi.org/10.1007/BF00992698
55. Wolper, P., Boigelot, B.: Verifying systems with infinite but regular state spaces. In: Hu, A.J., Vardi, M.Y. (eds.) CAV 1998. LNCS, vol. 1427, pp. 88–97. Springer, Heidelberg (1998). https://doi.org/10.1007/BFb0028736
56. Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., Philip, S.Y.: A comprehensive survey on graph neural networks. IEEE Trans. Neural Netw. Learn. Syst. **32**(1), 4–24 (2021). https://doi.org/10.1109/TNNLS.2020.2978386
57. Xu, Z., Gavran, I., Ahmad, Y., Majumdar, R., Neider, D., Topcu, U., Wu, B.: Joint inference of reward machines and policies for reinforcement learning. In: International Conference on Automated Planning and Scheduling, Nancy, France, October 26-30, 2020, pp. 590–598. AAAI Press (2020). https://doi.org/10.1609/icaps.v30i1.6756
58. Xu, Z., Wu, B., Ojha, A., Neider, D., Topcu, U.: Active finite reward automaton inference and reinforcement learning using queries and counterexamples. In: Holzinger, A., Kieseberg, P., Tjoa, A.M., Weippl, E. (eds.) CD-MAKE 2021. LNCS, vol. 12844, pp. 115–135. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-84060-0_8
59. Zhou, W., Li, W.: A hierarchical Bayesian approach to inverse reinforcement learning with symbolic reward machines. In: International Conference on Machine Learning, ICML. Proceedings of Machine Learning Research, vol. 162, pp. 27159–27178. PMLR (2022). https://proceedings.mlr.press/v162/zhou22b.html

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.