• RESEARCH PAPER •

A Formal Language for Performance Evaluation based on Reinforcement Learning

Fujun Wang[1], Zining Cao[1,2,3,4*], Yan Ma[5*]

[1]*College of Computer science and technology, Nanjing University of Aeronautics and Astronautics, Nanjing* 211106*, China;*
[2]*Key Laboratory of Safety-Critical Software, Nanjing University of Aeronautics and Astronautics, Nanjing* 211106*, China;*
[3]*Collaborative Innovation Center of Novel Software Technology and Industrialization, Nanjing University of Aeronautics and Astronautics, Nanjing* 211106*, China;*
[4]*Science and Technology on Electro-optic Control Laboratory, Luoyang* 471023*, China;*
[5]*College of Accounts, Nanjing University of Finance and Economics, Nanjing* 210023*, China*

Abstract During the past decades, performance evaluation has received great research attention, and many formal specification languages have been proposed to overcome the shortcomings of temporal logics that can only produce true or false responses, as they are still logics. With the popularity of reinforcement learning, as far as we know, there is still no formal language that can describe the goal of reinforcement learning till now. In this work we aim to propose a novel formal language to fill the gap between formal languages and the objective functions of RL. In this regard our work is the first to use a formal language to describe the goal function of reinforcement learning. First, we propose two formal languages, one is intuitive, closer to natural language, and can describe not only the minimum/maximum value of the path that satisfying the specification, but also the average value of all the paths that satisfying the specification, the other is succinct, more expressive and can describe almost the former. Then, we prove the monotonicity of the operator of the later language, thus the existence of fixed points. Finally, we use the proposed language to describe performance-related queries and use reinforcement learning techniques to calculate the fixed points of the corresponding formulas to get the target values. The effectiveness of the new language is demonstrated by a case study.

Keywords formal language, performance evaluation, reinforcement learning, fixed point, complete lattice

Citation Wang F, Cao Z, Zong H, et al. A Formal Language for Performance Evaluation based on Reinforcement Learning. Sci China Inf Sci, for review

1 Introduction

Performance Evaluation (PE) means to describe, to analyse, and to optimise the dynamic, time-dependent behaviour of systems [1]. It includes the measurement and modeling of real system behavior, the definition and determination of characteristic performance measures, and the development of design rules which guarantee an adequate quality of service [2,3]. Performance evaluation has been a discipline of computer science for more than fifty years. During five decades of research, performance evaluation has achieved a rich body of knowledge. Most research groups focused on areas like performance evaluation techniques and tools, communication networks, computer architecture, computer resource management, as well as the performance of software systems [4]. As time evolves, the scope of performance evaluation has been broadened to include parallel and distributed architectures, processor architecture, operating systems, database systems, real-time systems and more. However, most of the research focuses on the technical and application fields of performance evaluation, and scant attention is paid to the use of a formal specification language to describe performance-related queries, like what is the minimum/maximum/average cost of the system from the current state to the target state.

Reinforcement learning (RL) refers to the problem of an agent interacting with an unknown environment, which is usually modeled as the Markov Decision Process (MDP), to maximize its cumulative

* Corresponding author (email: caozn@nuaa.edu.cn, mayan1616@163.com)

reward in the long run. Although RL has been widely used in real life [5–7], the goal of RL still cannot be described by existing formal specification languages or temporal logics. In other words, there is still no language or temporal logic that can describe the goal of RL, that's maximizing the cumulative reward in the long run, let alone use RL techniques to get the result.

Motivation. The main motivation of the paper is to propose a novel formal language that can be used to describe the goal of RL. Using RL to solve a problem can be viewed as using RL techniques to compute the fixed point of the corresponding problem. Although it is well known that the goal of RL is to maximize the cumulative reward in the long run, it is still necessary to use a formal specification language to describe the goal of RL. Formal language is a language designed for use in situations in which natural language is unsuitable, for example in mathematics, logic, or computer programming. The symbols and formulas of such languages precisely stand for specified syntax and semantics relations to one another, i.e., the properties described by formal language are unambiguous and succinct, which are two of the significant advantages that natural language does not have. In this work, to fill the gap between formal languages and the objective functions of RL, we aim to propose a novel formal language for performance evaluation, which can describe not only the objective functions of RL but also other performance related properties, and in reinforcement learning realm, we can also use RL techniques to compute the results of some properties described by the proposed language.

Contribution. The main contributions of the paper are as follows:

• First, we propose two formal languages, one is intuitive, closer to natural language, similar to the temporal logic form and can describe the goal of reinforcement learning, the other is succinct, more expressive and can describe almost the former language;

• Second, we prove the monotonicity of language L_2, and thus the existence of fixed points. That is to say, the results of the properties described by language L_2 can be obtained by calculating the fixed points;

• Third, the relevant algorithms for the main part of the language are given and the corresponding results are obtained by using reinforcement learning;

• Fourth, our method is demonstrated by a case study.

Organization. Section 2 reviews some related studies in the literature. After preliminaries in Section 3, we give the syntax and semantics of the formal specification languages, prove the existence of the fixed points and the method of how to calculate the performance properties described by L_1 in Section 4. The algorithms for language L_1 are presented in Section 5. Then, in Section 6, we present a case study together with the experiment results. Finally, in Section 7, conclusions together with future perspectives are stated.

2 Related Work

A remarkable feature of the objective function of reinforcement learning is the discount factor, which guarantees the convergence of the objective function. In formal language and automata realm, discounted sum was first investigated in [8] on weighted automata, which was applied to infinite words, but the author only investigate the behavior of the system, not give a formal language to describe performance related property. In [9, 10], Krishnendu *et al.* also use the discounted sum of the transition weights to represent the real value of an infinite run, but only study the decision problems of automata theory together with their computational complexity. A representative achievement of introducing discount into temporal logic is the Discounted Computation Tree Logic (DCTL) [11, 12], which achieves robustness with respect to model perturbations by giving a quantitative interpretation to predicates and logical operators, and by discounting the importance of events according to how late they occur. However, in DCTL, the discounted sum was mutiplied by a normalizing factor to ensure that the value for all paths is within an interval (i.e., [0, 1]), and it represents the long-run average value of a quantitative observation. In addition, the value depends of the length of the path. Therefore, DCTL is not suitable for describing RL objective function. In addition, the author also introduces discounting into the μ-calculus, proposed discounted μ-calculus [13], quantitative μ-calculus [14], and developed a system theory with discounting. Inspired by DCTL, Wojciech proposed Markov Temporal Logic (MTL) for Markovian models [15], specifically, MTL_0 for Markov Chains, MTL_1 for Markov Decision Processes. In the same year, he also proposed MTL_2 for Stochastic Multi-Agent Systems in [16]. The temporal logics of the MTL series treat the discount factor in almost the same way as the DCTL, they still cannot suitable for describing the objective function of

RL. Other temporal logics that include discounting, like $\text{LTL}^{\text{disc}}[\text{D}]$ [17], still have the same problem.

In addition to these, as performance evaluation has been and will continue to be of great practical significance in all walks of life, most importantly, it has achieved a rich body of knowledge, therefore, it has attracted the attention of a growing number of researchers. They try to break through the limitation of temporal logics and propose a formal language to describe the properties related to performance evaluation. Over the past decade, two representative formal languages, Computation Tree Measurement Language (CTML) [18] and Hybrid Automata Stochastic Language (HASL) [19], emerged. HASL, which provides a unified framework both for performance and dependability evaluation and for model checking. By using HASL, which can express both the probability computation of a complex set of paths as well as elaborate performability measures, it can produce non-necessarily boolean quantities. In [18], the author proposed a real-valued formal language, CTML, towards the unification of performance evaluation and model checking. This language expands the results of quantitative analysis from $\{0, 1\}$ to $[0, \infty)$. In [20], a language called Continuous Time Real-valued Measurement Language (CTRML) was put forward, which is a further extension of CTML that further generalize the results to real numbers, it also combines the advantage of both CTML and HASL. In [21], the author put forth a language, called Performance Evaluation Language based on the combination of CTRML and MMTD (Measure of Medium Truth Degree), which can quantify the fuzzy information of the probabilistic property. Nevertheless, none of the above-mentioned languages involve discounting factor, not to mention describing the objective function of reinforcement learning.

3 Preliminaries

In order to make this paper self-contained, this section recalls some concepts, theorems, and gives the definitions that used in the paper. In this paper, R denotes real number, N denotes integer number, D denotes a bounded real number, π denotes a policy, and ρ represents a path with ρ_i (or $\rho[i]$) represents the $(i+1)$th state in the path. In this paper, we aim to propose a novel formal specification language for performance evaluation, which also called performance evaluation language. We shall no longer distinguish between them in the rest of the paper.

Definition 1 (Complete lattice [22])**.** Let P be a non-empty ordered set.
(1). If $x \wedge y$ and $x \vee y$ exist for all $x, y \in P$, then P is called a lattice.
(2). If $\wedge S$ and $\vee S$ exist for all $S \subseteq P$, then P is called a complete lattice.

In addition, there is another definition of a complete lattice, which is based on a partially ordered set. We use this definition to prove a complete lattice.

Definition 2. A complete lattice is a partially ordered set (poset, for short) with all joins (i.e., all the subsets of the poset have a join).

In the paper, we will subsequently use reinforcement learning techniques to solve the performance evaluation problem. As reinforcement learning problem can be modeled as a Markov Decision Process, we first give the definition of Markov Decision Process, which serve as the starting point and the theoretical framework for describing transitions between states.

Definition 3 (Markov Decision Process, MDP [23])**.** A MDP is defined as a tuple $M = (S, A, P, Re, \gamma)$, where:
· S is the set of states,
· A is the set of actions,
· $P : S \times A \times S \to [0, 1]$ is the transition probability function,
· $Re : S \times A \to R$ is the reward function,
· $\gamma \in [0, 1)$ is the discount factor for long-horizon returns.

In the following, we give a very important definition, that is the state mapping function, which establishes a relation between states and real numbers.

Definition 4 (State mapping function)**.** The state mapping function f is a function, which represents the mapping from state space S to real numbers within a closed interval:

$$f : S \to [x, y]. \tag{1}$$

From this definition, every state formula on the state space is mapped to a real value within the closed interval. The operations on state mapping formulas are then transformed into numerically related

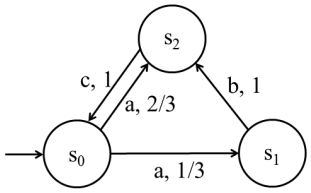


Figure 1 An MDP model.

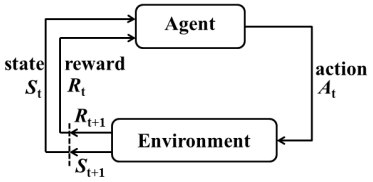


Figure 2 The agent-environment interaction in an MDP.

operations to calculate the final result.

Definition 5. Let f, g be two state mapping functions, define $f \leqslant g$ if and only if $f(s) \leqslant g(s)$ for any $s \in S$. In particular, $f = g$ if and only if $f \leqslant g$ and $g \leqslant f$.

Example 1. Figure 1 shows an example of a MDP, with $S = \{s_0, s_1, s_2\}, A = \{a, b, c\}$. Two state mapping functions f, g with $f(s_0) = f(s_1) = 1, f(s_2) = 0, g(s_0) = 1, g(s_1) = g(s_2) = 0$. By Definition 5, we can immediately get $g \leqslant f$. In this example, $f, g : S \to [0, 1]$.

With the above definition, the following two theorems are the theoretical basis for the existence of fixed points.

Theorem 1. Let $P = \{f\}$, then $< P, \leqslant >$ is a complete lattice.

Proof. It is not hard to prove that $< P, \leqslant >$ is a poset.
Let $B \subseteq P, X \triangleq \{f | f \in P \ \& \ \forall g \in B(g \leqslant f)\}, z \triangleq \sqcap X$
Let $g \in B \ \& \ g \leqslant f$
$\because g \leqslant f$ for any $g \in B$
$\therefore f \in X$
$\therefore g \leqslant \sqcap X = z$
$\because f \in X, z = \sqcap X$
$\therefore z \leqslant f$
$\therefore z$ is the join of B
$\therefore < P, \leqslant >$ is a complete lattice. □

Theorem 2 (Fixed Point Theorem [24, 25])**.** On a complete lattice, a monotone endofunction has a complete lattice of fixed points. In particular the greatest fixed point of the function is the join of all its post-fixed points, and the least fixed point is the meet of all its pre-fixed points.

In the following, we briefly introduce some knowledge related to reinforcement learning.

Reinforcement learning is a machine learning technique in which an agent learns the structure of the environment based on observations, and maximizes the rewards by acting according to the learnt knowledge [26]. The standard setting of RL consists of two parts: agent and environment, as shown in Figure 2, which is often formulated as a MDP. By observing the current state and reward from the environment, the agent returns the next action to the environment. Informally speaking, the agent's goal is to maximize the cumulative reward it receives in the long run. In order to ensure that the infinite sum of the reward has a finite value and meanwhile to emphasis that current reward have much value than future reward, an additional concept that we need is that of discounting. Therefore, the agent tries to select actions that maximize the sum of the discounted rewards it receives in the future.

Definition 6 (Expected discounted return [27])**.** The expected discounted return is:

$$G_t \triangleq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots = \sum_{i=0}^{T} \gamma^i R_{t+1+i}, \tag{2}$$

where $T \in N \cup \{\infty\}$ is the terminal state of the episode and $\gamma \in [0, 1]$ is the discount rate.

Assumption: (1).(bounded rewards) The reward sequence $< R_t >_{t \in N}$ is bounded.
(2).(finite returns) $T = \infty$ and $\gamma = 1$ do not hold at the same time.
This assumption ensures that all returns are finite.

Definition 7 (Reinforcement learning problem [27])**.** Given the states S, the actions A, and the transition function $S \times A \to S \times R$ of the environment, a reinforcement learning problem consists of finding policies for selecting the actions of an agent such that the expected discounted return is maximized.

Definition 8 (Policy [27])**.** A policy is a function $\pi : S \times A \to [0, 1]$. An agent is said to follow a policy π at time t, if $\pi(a|s)$ is the probability that the chosen action at time t is $A_t = a$ and the state at time t

is $S_t = s$. In the following, we denote the set of all policies by $\mathscr{B}$.

In reinforcement learning, two important functions, the (state-) value function and the action value function, are very useful for the agent because they indicate how expedient it is to be in a state or to be in a state and to take a certain action, respectively.

Definition 9 ((State-) Value function [27])**.** The value function of a state s under a policy π is:

$$v : \mathscr{B} \times S \to R, \tag{3}$$

$$v_\pi(s) := E_\pi[G_t|S_t = s], \tag{4}$$

i.e., it is the expected discounted return when starting in state s and following the policy π until the end of the episode.

Definition 10 (Action-value function [27])**.** The action-value function of a state action pair (s, a) under a policy π is:

$$q : \mathscr{B} \times S \times A \to R, \tag{5}$$

$$q_\pi(s, a) := E_\pi[G_t|S_t = s, A_t = a], \tag{6}$$

i.e., it is the expected discounted return when starting in state s, taking action a, and then following the policy π until the end of the episode.

Another commonly used equation is the Bellman equation for v_π, which expresses a relationship between the value of a state and the values of its successor states. Here, we do not give the definition of Bellman equation, but give the definition of Bellman optimal equation directly, which is the main method used in this paper.

Solving a reinforcement learning task means finding a policy than achieves a lot of reward over the long run. A policy π is defined to be better than or equal to a policy π' if its expected return is greater than ot equal to that of π' for all states. That is to say, $\pi \geqslant \pi'$ if and only if $v_\pi(s) \geqslant v_{\pi'}(s)$ for all $s \in S$. We call such a policy as an optimal policy, and all the optimal policies are denoted as π_*. Once we get an optimal policy, then we can get the optimal state-value function, optimal action-value function, and the Bellman equation will no longer refer to any specific policy. Hence, we can get two Bellman optimality equations for v_* and q_* respectively.

Definition 11 (Optimal state-value function, optimal action-value function, and Bellman optimality equation [28])**.** All the optimal policies share the same state-value function, called the optimal state-value function, denoted v_*, for all $s \in S$ it is defined as:

$$v_*(s) \doteq max_\pi v_\pi(s). \tag{7}$$

Optimal policies also share the same optimal action-value function, denoted q_*, is defined as

$$q_*(s, a) \doteq max_\pi q_\pi(s, a), \tag{8}$$

for all $s \in S$ and $a \in A(s)$.

The Bellman optimality equation for v_* is defined as:

$$\begin{aligned} v_*(s) &= max_{a \in A(s)} q_{\pi_*}(s, a) \\ &= max_a E_{\pi_*}[G_t|S_t = s, A_t = a] \\ &= max_a E_{\pi_*}[R_{t+1} + \gamma G_{t+1}|S_t = s, A_t = a] \\ &= max_a E_{\pi_*}[R_{t+1} + \gamma v_*(S_{t+1})|S_t = s, A_t = a] \\ &= max_a \sum_{s',r} p(s', r|s, a)[r + \gamma v_*(s')]. \end{aligned} \tag{9}$$

The Bellman optimality equation for q_* is defined as:

$$\begin{aligned} q_*(s, a) &= E[R_{t+1} + \gamma max_{a'} q_*(S_{t+1}, a')|S_t = s, A_t = a] \\ &= \sum_{s',r} p(s', r|s, a)[r + \gamma max_{a'} q_*(s', a')]. \end{aligned} \tag{10}$$

Apart from Bellman equation, another frequently used technology is Q-learning, which is a form of model-free reinforcement learning, and it is also an off-policy temporal-difference control method that directly approximates the optimal action-value function.

Theorem 3 (Convergence of Q-learning [27,29])**.** The Q-learning iterates

$$Q_{t+1}(s,a) := (1-\alpha_t(s,a))Q_t(s_t,a_t) + \alpha_t(s,a)(r_{t+1} + \gamma max_{a'} Q_{t+1}(s_{t+1},a')), \tag{11}$$

where $\alpha_t : S \times A \to [0,1)$, convergence to the optimal action-value function q^* if the following assumptions hold.

(1). The state space S and the action space A are finite.

(2). The equality $\sum_t \alpha_t(s,a) = \infty$ and the inequality $\sum_t \alpha_t(s,a)^2 < \infty$ hold for all $(s,a) \in S \times A$.

(3). The variance $\mathrm{V}[r_t]$ of the rewards is bounded.

During the training of an RL agent in order to determine the optimal policy, we need to update the Q-value, which is the output of the Action-Value Function, iteratively. The Q-learning algorithm, which is nothing but a technique to solve the optimal policy problem, iteratively updates the Q-values for each state-action pair using the Bellman Optimality Equation until the Q-function (Action-Value function) converges to the optimal Q-function, i.e., q^*.

4 Performance Evaluation Languages

In this section, we mainly give the syntax and semantics of the languages. First, we give the syntax and semantics of language L_1, which is intuitive and similar to the temporal logic style formulas that we are familiar with. Namely, language L_1 is closer to the natural language, similar to the form of temporal logic (like CTL, LTL [30,31]), and it can be easier to describe the performance of a system. Then, we give the syntax and semantics of language L_2, which is more expressive than language L_1 and almost all the formulas of L_1 can be described by using language L_2, and the results can be obtained by calculating the fixed points of the corresponding functions.

4.1 Performance evaluation language $\mathbf{L}_1$

The syntax of language L_1 is strongly influenced by CTML, HASL and RL. As it is mentioned previously, the language is used to describe the goal of reinforcement learning, it must contain the prefix, like *min* and *max*. Like LTL, CTL, and PCTL, it also uses temporal path operators, namely, "next" and "until", but the "until" operator is a temporal path operator with discount factor. That is to say, each step is multiplied by the discount factor.

Syntax

Let AF be a set of atomic state functions, and $p(s,a) \in AF$, then the syntax of L_1 can be defined recursively as follows:

$\varphi ::= p(s,a) \mid \varphi + \varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \mathrm{MX}\varphi \mid min\mathrm{X}\varphi \mid max\mathrm{X}\varphi \mid \mathrm{M}\varphi\ \mathrm{U}^{\mathrm{k}}_{+}\ \varphi \mid min\ \varphi\ \mathrm{U}^{\mathrm{k}}_{+}\ \varphi \mid max\ \varphi\ \mathrm{U}^{\mathrm{k}}_{+}\ \varphi$

where s is a state, a is an action, and $k \in [0,1)$ is the coefficient (or discount factor).

Semantics

The semantics of language L_1 are defined in terms of state mapping functions, which are real-valued functions, rather than a satisfaction relation, while still preserving the existing meaning of temporal logics like CTL and PCTL. Now, the formal semantics of the operators appearing in language L_1 are given as follows.

Let φ_1, φ_2 be two state functions, and then the semantics of L_1 can be recursively defined as follows:

- If $h = p(s,a)$, then $h(s_0) = \begin{cases} 1, & \text{if } M, s_0 \models p(s,a), \\ 0, & \text{otherwise.} \end{cases}$
- If $h = \varphi_1 + \varphi_2$, then $h(s_0) = \varphi_1(s_0) + \varphi_2(s_0)$.
- If $h = \varphi_1 \wedge \varphi_2$, then $h(s_0) = min\{\varphi_1(s_0), \varphi_2(s_0)\}$.
- If $h = \varphi_1 \vee \varphi_2$, then $h(s_0) = max\{\varphi_1(s_0), \varphi_2(s_0)\}$.
- If $h = \mathrm{MX}\varphi_1$, then $h(s_0) = \begin{cases} \sum_{t_i \in T} p(s_0,t_i) \cdot \varphi_1(t_i), & \text{if } T = \{t | s \to t\} \neq \emptyset, \\ \varphi_1(s_0), & \text{otherwise,} \end{cases}$ with $\sum_{t_i \in T} p(s_0,t_i) \cdot \varphi_1(t_i)$

is absolute convergence.

- If $h = min\mathrm{X}\varphi_1$, then $h(s_0) = \begin{cases} min\{\varphi_1(t_i)|t_i \in T\}, & \text{if } T = \{t|s \to t\} \neq \emptyset, \\ \varphi_1(s_0), & \text{otherwise.} \end{cases}$
- If $h = max\mathrm{X}\varphi_1$, then $h(s_0) = \begin{cases} max\{\varphi_1(t_i)|t_i \in T\}, & \text{if } T = \{t|s \to t\} \neq \emptyset, \\ \varphi_1(s_0), & \text{otherwise.} \end{cases}$
- If $h = min\varphi_1\mathrm{U}^{\mathrm{k}}_{+}\varphi_2$, then $h(s_0) = min_{\rho}\{\varphi_1\mathrm{U}^{\mathrm{k}}_{+}\varphi_2(\rho) \mid \rho_0 = s_0\}$.
- If $h = max\varphi_1\mathrm{U}^{\mathrm{k}}_{+}\varphi_2$, then $h(s_0) = max_{\rho}\{\varphi_1\mathrm{U}^{\mathrm{k}}_{+}\varphi_2(\rho) \mid \rho_0 = s_0\}$.
- If $h = \mathrm{M}\varphi_1\mathrm{U}^{\mathrm{k}}_{+}\varphi_2$, then $h(s_0) = \sum_{\rho}\{(\varphi_1\mathrm{U}^{\mathrm{k}}_{+}\varphi_2(\rho)) * \prod_{i=0}^{|\rho|-1} P(\rho_i, \rho_{i+1})|\rho_0 = s_0\}$.
- If for any (finite or infinite) path $\rho = s_0, s_1, s_2, s_3, \ldots, s_n, \ldots,\ n \in N \cup \{\infty\}$, we have

$$\varphi_1\mathrm{U}^{\mathrm{k}}_{+}\varphi_2(\rho) = \begin{cases} \sum_{i=0}^{j-1} k^i \cdot \varphi_1(s_i) \cdot \varphi_2(s_j), & \text{if } j = min\{i|\varphi_2(\rho[i]) > 0\} \geqslant 1 \wedge j \leqslant n, \\ \sum_{i=0}^{|\rho|-1} k^i \cdot \varphi_1(s_i), & \text{if } \forall i : 0 \leqslant i < |\rho|\ (\varphi_2(\rho[i]) = 0), \\ 0, & \text{if } \varphi_2([\rho[0]]) > 0. \end{cases}$$

Although the syntax of performance evaluation language L_1 is intuitive and closer to our natural language, the computing process is not necessarily the case. In other words, the performance query results of L_1 may not be easy to compute. Alternatively, we can convert it to another performance evaluation language, which can get the corresponding results easily. That is to say, we can take advantage of another language to get the performance results of L_1. Based on the above reasons and factual analysis, another performance evaluation language was proposed.

4.2 Performance evaluation language L_2

To overcome the drawback of L_1, an alternative performance evaluation language L_2 was proposed. Almost all the performance properties described by L_1 can be transformed to the performance properties described by L_2, and then the performance results can be obtained by calculating the fixed points with respect to the corresponding performance query formulas. As with language L_1, we first give the syntax and semantics of L_2 as follows:

Syntax

Let AF be a set of atomic state functions, and $p(s, a) \in AF$, then the syntax of L_2 can be defined recursively as follows:

$\varphi ::= p(s, a) \mid \varphi + \varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \mathrm{MX}\varphi \mid min\mathrm{X}\varphi \mid max\mathrm{X}\varphi$

where s is a state, a is an action.

Semantics

Like language L_1, the semantics of language L_2 are defined in terms of real-valued functions. The formal semantics of language L_2 are given as follows.

Let φ_1, φ_2 be two state functions, and then the semantics of L_2 can be recursively defined as follows:

- If $h = p(s, a)$, then $h(s_0) = \begin{cases} 1, & \text{if } M, s_0 \models p(s, a), \\ 0, & \text{if } M, s_0 \not\models p(s, a). \end{cases}$
- If $h = \varphi_1 + \varphi_2$, then $h(s_0) = \varphi_1(s_0) + \varphi_2(s_0)$.
- If $h = \varphi_1 \wedge \varphi_2$, then $h(s_0) = min\{\varphi_1(s_0), \varphi_2(s_0)\}$.
- If $h = \varphi_1 \vee \varphi_2$, then $h(s_0) = max\{\varphi_1(s_0), \varphi_2(s_0)\}$.
- If $h = \mathrm{MX}\varphi_1$, then $h(s_0) = \begin{cases} \sum_{t_i \in T} p(s_0, t_i) \cdot \varphi_1(t_i), & \text{if } T = \{t|s \to t\} \neq \emptyset, \\ \varphi_1(s_0), & \text{otherwise,} \end{cases}$ with $\sum_{t_i \in T} p(s_0, t_i) \cdot \varphi_1(t_i)$ is absolute convergence.
- If $h = min\mathrm{X}\varphi_1$, then $h(s_0) = \begin{cases} min\{\varphi_1(t_i)|t_i \in T\}, & \text{if } T = \{t|s \to t\} \neq \emptyset, \\ \varphi_1(s_0), & \text{otherwise.} \end{cases}$
- If $h = max\mathrm{X}\varphi_1$, then $h(s_0) = \begin{cases} max\{\varphi_1(t_i)|t_i \in T\}, & \text{if } T = \{t|s \to t\} \neq \emptyset, \\ \varphi_1(s_0), & \text{otherwise.} \end{cases}$

Compared with language L_1, language L_2 is concise and more expressive. Most importantly, L_2 can describe almost all the performance queries described by L_1, which will be described later.

To prove the existence of the fixed point, we first prove the monotonicity of the state mapping function in the form of language L_2, and then we can naturally obtain the fixed point on the complete lattice under the fixed point theorem.

Lemma 1. The state mapping function in the form of L_2 is monotonic.

Proof. It suffices to show that the operators of L_2 are monotonic. We do this by structural induction on the operators of L_2.

Case 1. Let $\varphi_1 = p(s,a) = \varphi_2$
$\therefore \forall s \in S(\varphi_1(s) = \varphi_2(s))$

Case 2. Let $\varphi_1(s) \leqslant \varphi_2(s)$ for any $s \in S$
$\therefore \varphi_1(s) + \varphi(s) \leqslant \varphi_2(s) + \varphi(s)$

Case 3. Let $\varphi_1(s) \leqslant \varphi_2(s)$ for any $s \in S$
$\therefore \varphi_1(s) \wedge \varphi(s) = min\{\varphi_1(s), \varphi(s)\},\ \varphi_2(s) \wedge \varphi(s) = min\{\varphi_2(s), \varphi(s)\}$

Case 3.1. $\varphi_1(s) \leqslant \varphi_2(s) \leqslant \varphi(s)$
$\therefore \varphi_1(s) \wedge \varphi(s) = min\{\varphi_1(s), \varphi(s)\} = \varphi_1(s) \leqslant \varphi_2(s) = min\{\varphi_2(s), \varphi(s)\} = \varphi_2(s) \wedge \varphi(s)$

Case 3.2. $\varphi_1(s) \leqslant \varphi(s) \leqslant \varphi_2(s)$
$\therefore \varphi_1(s) \wedge \varphi(s) = min\{\varphi_1(s), \varphi(s)\} = \varphi_1(s) \leqslant \varphi(s) = min\{\varphi_2(s), \varphi(s)\} = \varphi_2(s) \wedge \varphi(s)$

Case 3.3. $\varphi(s) \leqslant \varphi_1(s) \leqslant \varphi_2(s)$
$\therefore \varphi_1(s) \wedge \varphi(s) = min\{\varphi_1(s), \varphi(s)\} = \varphi(s) = min\{\varphi_2(s), \varphi(s)\} = \varphi_2(s) \wedge \varphi(s)$

By Case 3.1-3.3, $\varphi_1(s) \wedge \varphi(s) \leqslant \varphi_2(s) \wedge \varphi(s)$

Case 4. Let $\varphi_1(s) \leqslant \varphi_2(s)$ for any $s \in S$
$\therefore \varphi_1(s) \vee \varphi(s) = max\{\varphi_1(s), \varphi(s)\},\ \varphi_2(s) \vee \varphi(s) = max\{\varphi_2(s), \varphi(s)\}$
We can conclude that $\varphi_1(s) \vee \varphi(s) \leqslant \varphi_2(s) \vee \varphi(s)$, the proof is similar to that of Case 3.

Case 5. Let $\varphi_1(s) \leqslant \varphi_2(s)$ for any $s \in S$

Case 5.1. $T = \{t | s \to t\} = \emptyset$
then, $\mathrm{MX}\varphi_1(s) = \varphi_1(s) \leqslant \varphi_2(s) = \mathrm{MX}\varphi_2(s)$

Case 5.2. $T = \{t | s \to t\} \neq \emptyset$
then, $\mathrm{MX}\varphi_1(s) = \sum_{t_i \in T} p(s_0, t_i) \cdot \varphi_1(t_i) \leqslant \sum_{t_i \in T} p(s_0, t_i) \cdot \varphi_2(t_i) = \mathrm{MX}\varphi_2(s)$

By Case 5.1-5.2, $\mathrm{MX}\varphi_1(s) \leqslant \mathrm{MX}\varphi_2(s)$

Case 6. Let $\varphi_1(s) \leqslant \varphi_2(s)$ for any $s \in S$

Case 6.1. $T = \{t | s \to t\} = \emptyset$
then, $min\mathrm{X}\varphi_1(s) = \varphi_1(s) \leqslant \varphi_2(s) = min\mathrm{X}\varphi_2(s)$

Case 6.2. $T = \{t | s \to t\} \neq \emptyset$
then, $min\mathrm{X}\varphi_1(s) = min_{t_i \in T}\{\varphi_1(t_i)\} \leqslant \varphi_1(t_j) \leqslant \varphi_2(t_j) = min_{t_i \in T}\{\varphi_2(t_i)\} = min\mathrm{X}\varphi_2(s)$

By Case 6.1-6.2, $min\mathrm{X}\varphi_1(s) \leqslant min\mathrm{X}\varphi_2(s)$

Case 7. Let $\varphi_1(s) \leqslant \varphi_2(s)$ for any $s \in S$

Case 7.1. $T = \{t | s \to t\} = \emptyset$
then, $max\mathrm{X}\varphi_1(s) = \varphi_1(s) \leqslant \varphi_2(s) = max\mathrm{X}\varphi_2(s)$

Case 7.2. $T = \{t | s \to t\} \neq \emptyset$
then, $max\mathrm{X}\varphi_1(s) = max_{t_i \in T}\{\varphi_1(t_i)\} = \varphi_1(t_j) \leqslant \varphi_2(t_j) \leqslant max_{t_i \in T}\{\varphi_2(t_i)\} = max\mathrm{X}\varphi_2(s)$

By Case 7.1-7.2, $max\mathrm{X}\varphi_1(s) \leqslant max\mathrm{X}\varphi_2(s)$

In conclusion, the operators of L_2 is monotonic. In other words, the state mapping function in the form of L_2 is monotonic. □

Theorem 4. For formulas of type L_2, fixed points exist.

Proof. By Theorem 1, 2, and Lemma 1, the conclusion is obvious. □

Theorem 5. For formulas in the style of language L_2, the result convergences to the target value with probability 1.

4.3 Calculating properties described by language $\mathbf{L}_1$

In this section, we mainly discuss how the properties described by language L_1 can be calculated. As language L_2 was proposed based on L_1, therefore, most formulas described by L_1 can aslo be described by L_2. That is to say, most formulas in the form of L_1 can be described by formulas in the form of L_2. For the same form of formulas in L_1 and L_2, they are the same, therefore, we only need to focus on $\mathrm{M}\varphi\mathrm{U}^{\mathrm{k}}_{+}\varphi, min\varphi\mathrm{U}^{\mathrm{k}}_{+}\varphi$ and $max\varphi\mathrm{U}^{\mathrm{k}}_{+}\varphi$ three types of formulas.

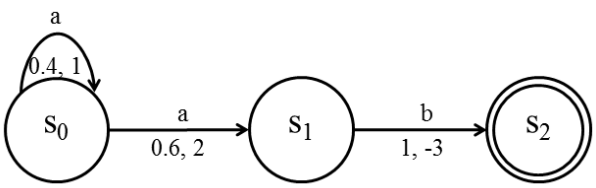


Figure 3 A simple MDP model.

(1). $min\varphi \mathrm{U}_{+}^{\mathrm{k}}\varphi$ and $max\varphi \mathrm{U}_{+}^{\mathrm{k}}\varphi$
For $min\varphi \mathrm{U}_{+}^{\mathrm{k}}\varphi$ and $max\varphi \mathrm{U}_{+}^{\mathrm{k}}\varphi$, they can be described by L_2. Suppose for any (finite or infinite) path $\rho = s_0, s_1, s_2, s_3, \ldots, s_n, \ldots,\ n \in N \cup \{\infty\}$, we have

$$\varphi_1 \mathrm{U}_{+}^{\mathrm{k}} \varphi_2(\rho) = \begin{cases} \sum_{i=0}^{j-1} k^i \cdot \varphi_1(s_i) \cdot \varphi_2(s_j), & \text{if } j = min\{i | \varphi_2(\rho[i]) > 0\} \geqslant 1 \wedge j \leqslant n, \\ \sum_{i=0}^{|\rho|-1} k^i \cdot \varphi_1(s_i), & \text{if } \forall i : 0 \leqslant i < |\rho|\ (\varphi_2(\rho[i]) = 0), \\ 0, & \text{if } \varphi_2([\rho[0]]) > 0. \end{cases}$$

For $\varphi_1 \mathrm{U}_{+}^{\mathrm{k}} \varphi_2(\rho)$, we discuss it by cases.
Case1. $\varphi_2(\pi[0]) = \varphi_2(s_0) > 0$
$\therefore \varphi_1 \mathrm{U}_{+}^{\mathrm{k}} \varphi_2(\rho) = 0$
Case2. $\forall i : 0 \leqslant i < |\rho|\ (\varphi_2(\rho[i]) = 0)$, i.e., $\neg \exists s_i\ s.t.\ \varphi_2(\rho[i+1]) = \varphi_2(s_i) > 0$
Let $\rho = s_0 \cdot \rho'$

$$\begin{aligned} \therefore \varphi_1 \mathrm{U}_{+}^{\mathrm{k}} \varphi_2(\rho) &= \sum_{i=0}^{n} k^i \cdot \varphi_1(s_i) \\ &= \varphi_1(s_0) + k^1 \cdot \varphi_1(s_1) + k^2 \cdot \varphi_1(s_2) + \ldots + k^n \cdot \varphi_1(s_n) \\ &= \varphi_1(s_0) + k(\varphi_1(s_1) + k^1 \cdot \varphi_1(s_2) + \ldots + k^n \cdot \varphi_1(s_n)) \\ &= \varphi_1(s_0) + k(\varphi_1 \mathrm{U}_{+}^{\mathrm{k}} \varphi_2(\rho')) \end{aligned}$$

Case3. $\exists s_j (1 \leqslant j \leqslant n)\ s.t.\ \varphi_2(\rho[j+1]) = \varphi_2(s_j) > 0$
Let $\rho = s_0 \cdot \rho'$

$$\begin{aligned} \therefore \varphi_1 \mathrm{U}_{+}^{\mathrm{k}} \varphi_2(\rho) &= \sum_{i=0}^{j-1} k^i \cdot \varphi_1(s_i) \cdot \varphi_2(s_j) \\ &= (\varphi_1(s_0) + k^1 \cdot \varphi_1(s_1) + k^2 \cdot \varphi_1(s_2) + \ldots + k^{j-1} \cdot \varphi_1(s_{j-1})) \cdot \varphi_2(s_j) \\ &= \varphi_1(s_0) \cdot \varphi_2(s_j) + k(\varphi_1(s_1) + k^1 \cdot \varphi_1(s_2) + \ldots + k^{j-1} \cdot \varphi_1(s_{j-1})) \cdot \varphi_2(s_j) \\ &= \varphi_1(s_0) \cdot \varphi_2(s_j) + k(\varphi U_{+}^{k} \varphi_2(\rho')) \end{aligned}$$

By Case 1-3, $\varphi_1 \mathrm{U}_{+}^{\mathrm{k}} \varphi_2$ is a fixed point of the form $\tau(\varphi) = k\varphi + \Psi(s, a)$ with $|\Psi(s, a)| \leqslant D$
$\therefore\ min\varphi_1 \mathrm{U}_{+}^{\mathrm{k}} \varphi_2 = \mu\varphi . k\varphi + \Psi(s, a)$
$max\varphi_1 \mathrm{U}_{+}^{\mathrm{k}} \varphi_2 = \nu\varphi . k\varphi + \Psi(s, a)$

(2). $\mathrm{M}\varphi_1 \mathrm{U}_{+}^{\mathrm{k}} \varphi_2$
For $\mathrm{M}\varphi_1 \mathrm{U}_{+}^{\mathrm{k}} \varphi_2$, we modify the Bellman optimality equation and transform it into Equation (12) for calculating the mean value.

$$v_{ave}(s) = \frac{1}{P(s)} \sum_{a \in A(s)} \sum_{s', r} p(s', r | s, a)[r + \gamma v(s')], \tag{12}$$

where, $P(s) = \sum_{a \in A(s)} \pi(a|s)$.

The feasibility of the method was verified by an experiment. In Figure 3, we give a simple MDP model, which can be viewed as an automaton. For this model, state s_2 is the only accepting state and the accepting language of the model, i.e. $(a)^* \cdot a \cdot b$, can be described by regular language.

To calculate the value of the formula, one of the most direct ideas is to enumerate the accepting languages (for example, sort by the length of the accepting language) according to the rule that the value, which equals to the probability of the currently accepted language times the cumulative discount benefit of the language that corresponding to reinforcement learning, is arranged from high to low and when the value of a new accepting language is less than the threshold (for example 1×10^{-10}), the mean value can be approximated as the cumulative value of the languages accepted so far. Then, use the modified Bellman equation to calculate the mean value. Finally, compare the results of the two methods and check whether the results are the same. Setting $\gamma = k = 0.9$, then the results of these two methods are shown in Table 1.

From Table 1, we can conclude that the results for these two methods are nearly the same. That is to say, we can use the modified Bellman equation to approximate the mean value.

Table 1 Comparison of experimental results of the two methods

Property	RL Method	Enumeration Method
$\mathrm{MRU}_{+}^{\mathrm{k}}\mathrm{E}(s_0)$	-0.031250000000000	-0.031250000004435
$\mathrm{MRU}_{+}^{\mathrm{k}}\mathrm{E}(s_1)$	-3.000000000000000	-3.000000000000000

5 Algorithm

In this section, we present algorithms for computing formulas of language L_1. The results of $p(s,a), \varphi + \varphi, \varphi \wedge \varphi, \varphi \vee \varphi, min\mathrm{X}\varphi, max\mathrm{X}\varphi$, and $\mathrm{MX}\varphi$ can be easily calculated by arithmetic operations. Below are the algorithms for calculating $min\varphi\mathrm{U}_{+}^{\mathrm{k}}\varphi, max\varphi\mathrm{U}_{+}^{\mathrm{k}}\varphi$, and $\mathrm{M}\varphi\mathrm{U}_{+}^{\mathrm{k}}\varphi$.

Algorithm 1 $max\varphi_1\mathrm{U}_{+}^{\mathrm{k}}\varphi_2$ algorithm

Input: $max\varphi_1\mathrm{U}_{+}^{\mathrm{k}}\varphi_2$(initial_state), MDP $M = (S, A, P, Re, \gamma)$, the number of maximal iterations I, error parameter ε, $k = \gamma$;
Output: The numerical result of the property;
1: V_table=$zeros$; /* initialize V_table */
2: $i = 0, err = 1$; /* i: count the number of iterations, err: measure the difference before and after updating of V_table */
3: initial_state=s_0;
4: **while** ($err > \varepsilon$ & $i <= I$) **do**
5: T_table=V_table; /* save the values of V_table before updating */
6: **for** ($j = 0; j < |S|; j++$) **do**
7: $V[s_j] = max_{a \in A}\{\sum_{k=0}^{|S|} p(s_k|s_j, a)(Re_{jk} + \gamma * V[s_k])\}$;
8: **end for**
9: $err = sum(abs(\text{T\_table}(:) - \text{V\_table}(:)))$; /* measure the difference */
10: $i = i + 1$;
11: **end while**
12: **return** V_table[initial_state]; /* return final result */

Algorithm 1 is used to compute $max\varphi_1\mathrm{U}_{+}^{\mathrm{k}}\varphi_2$, which uses Bellman equation to compute the numerical result of this property, and the algorithm for computing $min\varphi_1 U_{+}^{k}\varphi_2$ is similar to Algorithm 1, in which we only need to replace "max" with "min" in line 7. The complexity of Algorithm 1 is $O(I \cdot |S|)$, where I denotes the number of maximal iterations, $|S|$ denotes the number of states.

The numerical result of $\mathrm{M}\varphi_1\mathrm{U}_{+}^{\mathrm{k}}\varphi_2$ can be solved by using an algorithm similar to the Bellman optimality equation, but with a minor modification. The algorithm is given in Algorithm 2. The most critical step in the algorithm is line 10, which is not the maximum of the current state, but the average. When the cumulative difference between the values before and after the V table update is less than the error range or the number of iterations reaches the upper limit, the algorithm stops. The complexity of Algorithm 2 is $O(I|S|)$.

Algorithm 2 $\mathrm{M}\varphi_1\mathrm{U}_{+}^{\mathrm{k}}\varphi_2$ algorithm

Input: $\mathrm{M}\varphi_1\mathrm{U}_{+}^{\mathrm{k}}\varphi_2$(initial_state), MDP $M = (S, A, P, Re, \gamma)$, the number of maximal iterations I, error ε, $k = \gamma$;
Output: The numerical result of the property;
1: V_table=$zeros$; /* initialize V_table */
2: $i = 0, err = 1$; /* i: count the number of iterations, err: measure the difference before and after updating of V_table */
3: initial_state=s_0;
4: **while** ($err > \varepsilon$ & $i <= I$) **do**
5: $i = i + 1$;
6: current_state = initial_state;
7: V_table_old = V_table; /* record the data before the V table is updated */
8: **for** ($j = 0; j < |S|; j++$) **do**
9: $V[s_j] = \sum_{k=0}^{|S|} \frac{p(s_j,s_k)}{\sum_{k=0}^{|S|} p(s_j,s_k)} * (Re_{jk} + \gamma * V[s_k])$;
10: **end for**
11: $err = sum(abs(\text{V\_table}(:) - \text{V\_table\_old}(:)))$; /* measure the difference */
12: **end while**
13: **return** V_table[initial_state]; /* return final result */

6 Case Study and Performance Evaluation

Take Recycling Robot [28] as an example, which describes the behavior of a robot recycling cans. The behavior of the recycling robot can be turned into a MDP model. Decisions made by the agent were

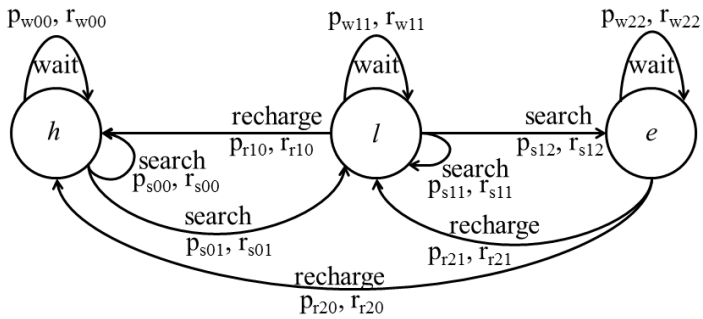


Figure 4 The MDP model of a Recycling Robot.

Table 2 The specific transition probabilities and expected rewards for the recycling robot

s	a	s'	$P(s'\|s,a)$	$Re(s'\|s,a)$
h	$wait$	h	1	1
h	$search$	h	0.8	2
h	$search$	l	0.2	2
l	$wait$	l	1	1
l	$search$	l	0.2	2
l	$search$	e	0.8	-3
l	$recharge$	h	1	-0.5
e	$wait$	e	1	0
e	$recharge$	h	0.4	-2
e	$recharge$	l	0.6	-1

determined by external events or by the robot. At each such time the robot decides whether it should (1) actively search for a can, (2) remain stationary and wait for someone to bring it a can, or (3) go back to home base to recharge its battery. Figure 4 depicts the behavior model of the recycling robot. In order to collect as many cans as possible, actively search for them seems to be the best approach, but it will run down the robot's battery, whereas waiting does not. It is possible that the battery will become depleted if the robot performs search behavior. Thus leading to the shutdown of the robot and waiting for rescue. For simplicity, the level of the battery is divided into high, low, and empty three levels, so the state set is $S = \{h, l, e\}$. The robot can perform *search*, *wait* and *recharge* three actions, so $A = \{search, wait, recharge\}$.

The behavior of the recycling robot is as follows: if the energy level of the battery is high, the robot can perform a *search* action with probability p_{s00} (resp. p_{s01}), get reward r_{s00} (resp. r_{s01}) and reach state h (resp. l), it can also perform a *wait* action with probability p_{w00}, get reward r_{w00} and stay on high battery level, i.e. state h. If the energy level is low, the robot can perform a *search* action with probability p_{s11} (resp. p_{s12}), get reward r_{s11} (resp. r_{s12}) and reach state l (resp. e), it can also perform a *wait* action with probability p_{w11}, get reward r_{w11} and stay on l, and it can also perform a *recharge* action with probability p_{r10}, get reward r_{r10} and reach state h. If the energy level is empty, the robot can perform a *wait* action with probability p_{w22}, get reward r_{w22} and stay on state e, it can also perform a *recharge* action with probability p_{r20} (resp. p_{r21}), get reward r_{r20} (resp. r_{r21}) and reach h (resp. l) state. This is a finite model, the specific transition probabilities and the expected rewards were shown in Table 2.

In order to describe performance queries, two state functions R and E were defined. Define E(e)=1 for empty battery energy, while R(h) and R(l) were not uniquely defined, their value depends on the rewards associated with the specific transition. For example, for the transition from state h to state l by performing a *search* action, then R(h) = r_{s01} = 2. The trick to perform this transformation is to use reinforcement learning techniques to solve the query results.

6.1 Performance Queries

In this section, we list some queries of the model for performance evaluation.

(1). "What is the maximal reward for performing a single step action when the robot starts with a high level of battery?" It can be expressed as:

$$max\mathrm{XR}(\ h\ ).$$

(2). "What is the minimal reward for the robot performing a single step action when it starts with a high level of battery?" It can be expressed as:

$$min\mathrm{XR}(\ h\ ).$$

(3). "What is the average reward for the robot performing a single step action when it starts with a high level of battery?" It can be expressed as:

$$\mathrm{MXR}(\ h\ ).$$

(4). "What is the maximal reward for performing a single step action when the robot starts with a low level of battery?" It can be expressed as:

$$max\mathrm{XR}(\ l\ ).$$

(5). "What is the minimal reward for the robot performing a single step action from the empty energy level of the battery?" It can be expressed as:

$$min\mathrm{XR}(\ e\ ).$$

(6). "What is the average reward for the robot performing a single step action when it starts with a low level of battery?" It can be expressed as:

$$\mathrm{MXR}(\ l\ ).$$

(7). "What's the maximal reward of the robot until the battery is empty when starting from the high level of the battery?" It can be expressed as:

$$max\mathrm{RU}_{+}^{\mathrm{k}}\mathrm{E}(\ h\ ).$$

(8). "What's the minimal reward of the robot until the battery is empty when starting from the high level of the battery?" It can be expressed as:

$$min\mathrm{RU}_{+}^{\mathrm{k}}\mathrm{E}(\ h\ ).$$

(9). "What's the maximal reward of the robot until the battery is empty when starting from the low level of the battery?" It can be expressed as:

$$max\mathrm{RU}_{+}^{\mathrm{k}}\mathrm{E}(\ l\ ).$$

(10). "What's the minimal reward of the robot until the battery is empty when starting from the low level of the battery?" It can be expressed as:

$$min\mathrm{RU}_{+}^{\mathrm{k}}\mathrm{E}(\ l\ ).$$

(11). "What's the average reward of the robot until the battery is empty when starting from the high level of the battery?" It can be expressed as:

$$\mathrm{MRU}_{+}^{\mathrm{k}}\mathrm{E}(\ h\ ).$$

(12). "What's the average reward of the robot until the battery is empty when starting from the low level of the battery?" It can be expressed as:

$$\mathrm{MRU}_{+}^{\mathrm{k}}\mathrm{E}(\ l\ ).$$

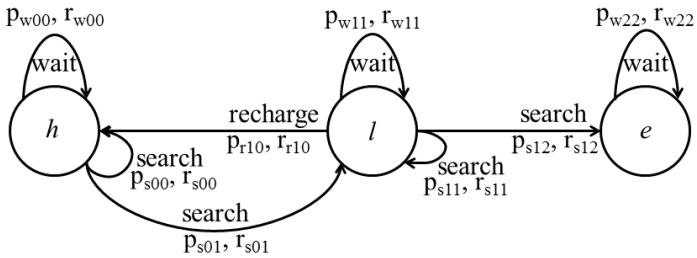
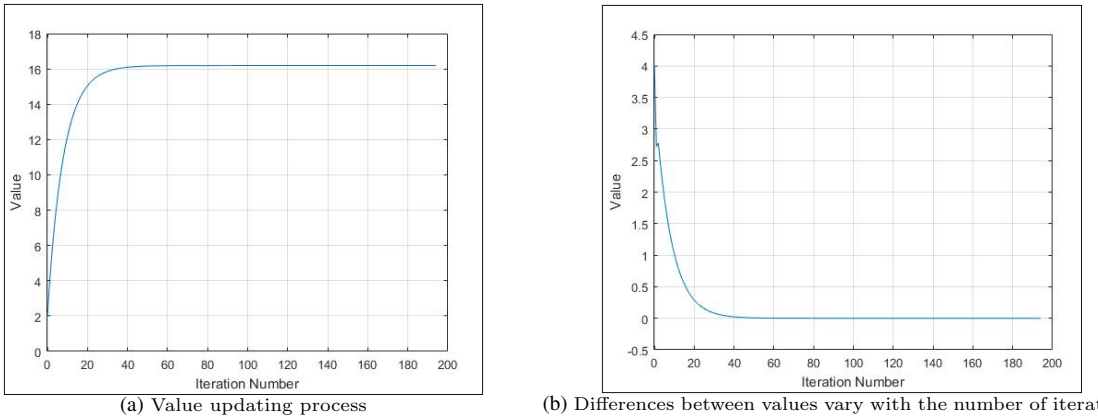


Figure 5 Modified MDP model of Recycling Robot.

Table 3 The numerical results for the queries of recycling robot.

No	Query	Results	No	Query	Results
1	maxXR(h)	2	7	$max\mathrm{RU}_{+}^{\mathrm{k}}\mathrm{E}(\ h\ )$	16.1864
2	minXR(h)	1	8	$min\mathrm{RU}_{+}^{\mathrm{k}}\mathrm{E}(\ h\ )$	8.7108
3	MXR(h)	1.5	9	$\mathrm{MRU}_{+}^{\mathrm{k}}\mathrm{E}(\ h\ )$	10.9408
4	maxXR(l)	2	10	$\mathrm{MRU}_{+}^{\mathrm{k}}\mathrm{E}(\ l\ )$	6.4306
5	minXR(e)	0	11	$max\mathrm{RU}_{+}^{\mathrm{k}}\mathrm{E}(\ l\ )$	14.0678
6	MXR(l)	-0.5	12	$min\mathrm{RU}_{+}^{\mathrm{k}}\mathrm{E}(\ l\ )$	2.4390



(a) Value updating process

(b) Differences between values vary with the number of iterations

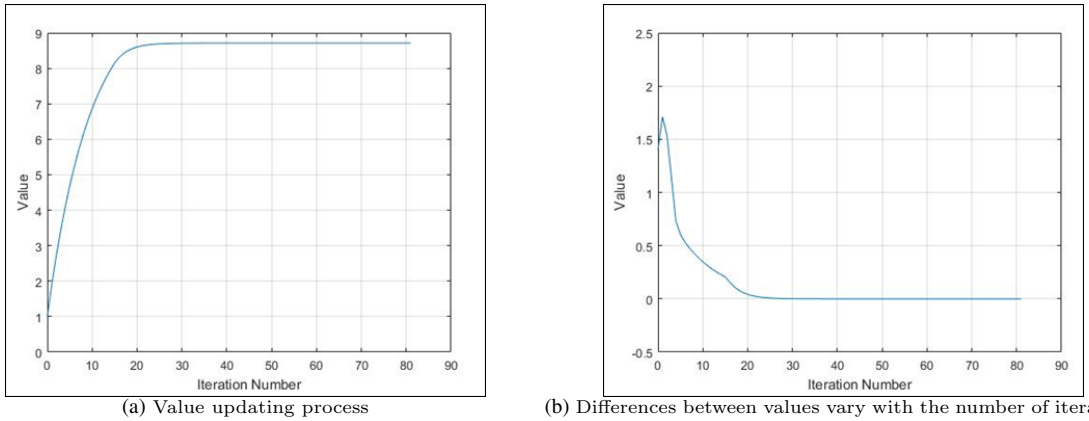
Figure 6 $max\mathrm{RU}_{+}^{\mathrm{k}}\mathrm{E}(\ h\ )$ curve.

6.2 Experiments and analysis

For performance queries (7)-(12), and by the semantics of $\mathrm{U}^{\mathrm{k}}_{+}$, once reached the target state (i.e. e), the algorithm stops and outputs the results. In other words, the transitions of performing a *recharge* action from state e to state h and state l will no longer be executed. Based on the above analysis, the recycling robot model was modified, and we get a simplified MDP model shown in Figure 5 for solving performance queries (7)-(12). All experiments are undertaken in Matlab 2016b by a personal computer with Intel Core i5-4200M CPU at 2.5 GHz with 12 GB of RAM. Setting parameter $\gamma = k = 0.9$. The experiment results are shown in Table 3.

Below, we present an analysis for the results of these three types formulas in the above experiment.

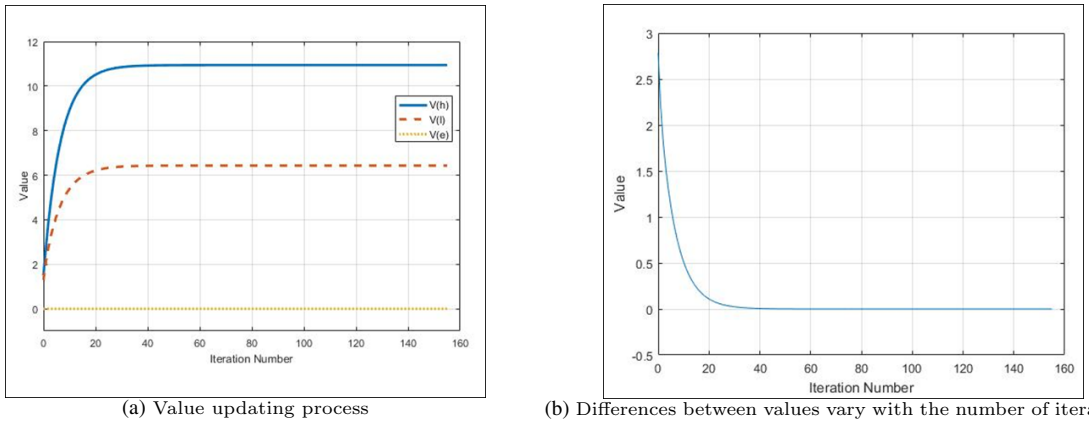
The curvilinear variation of $max\mathrm{RU}^{\mathrm{k}}_{+}\mathrm{E}(h)$ is plotted in Figure 6. From Figure 6(a), we can conclude that the final result of $max\mathrm{RU}^{\mathrm{k}}_{+}\mathrm{E}(h)$ converges to 16.1864 when the number of iterations reaches 194. From Figure 6(b), we can get that when the number of iterations reaches 40 times, the difference between values before and after updating is close to 0. When the iteration number reaches 194, the cumulative difference between the values is less than 1×10^{-10}, and the algorithm stops.



(a) Value updating process (b) Differences between values vary with the number of iterations

Figure 7 $min\mathrm{RU}^{\mathrm{k}}_{+}\mathrm{E}(\ h\ )$ curve.

Figure 7 gives the curvilinear variation of $min\mathrm{RU}^{\mathrm{k}}_{+}\mathrm{E}(h)$. The final result of $min\mathrm{RU}^{\mathrm{k}}_{+}\mathrm{E}(h)$ converges to 8.7108 when the number of iterations reaches 81. From Figure 7(b), we can conclude that when the number of iterations reaches 81, the cumulative difference between values is less than 1×10^{-10}, and the algorithm stops.



(a) Value updating process (b) Differences between values vary with the number of iterations

Figure 8 $\mathrm{MRU}^{\mathrm{k}}_{+}\mathrm{E}$ curve.

Figure 8 depicts the curvilinear variation of $\mathrm{MRU}^{\mathrm{k}}_{+}\mathrm{E}$. The final results of $\mathrm{MRU}^{\mathrm{k}}_{+}\mathrm{E}(h)$, $\mathrm{MRU}^{\mathrm{k}}_{+}\mathrm{E}(l)$ and $\mathrm{MRU}^{\mathrm{k}}_{+}\mathrm{E}(e)$ can be obtained from Figure 8(a), namely, 10.9408, 6.4306, and 0, respectively. When the

number of iterations reaches 155, the cumulative difference between values is less than 1×10^{-10}, and the algorithm stops. The experimental results for U_+^k are shown in Table 4.

6.3 Discussion

In the above experiment, we use the novel performance evaluation language to describe the objective function of reinforcement learning, like performance queries (7)-(12) in Table 3, and describe other queries, like (1)-(6). In addition, for performance queries (7)-(12), the results are computed by using reinforcement learning techniques. Furthermore, by modifying the reinforcement learning algorithm, we can also use it to calculate the mean of all paths that satisfy the formula. To our knowledge, our language is the first formal language that can not only describe the goal of reinforcement learning but also use it to get the results.

7 Conclusion and Future Work

With the prevalence of reinforcement learning and the development of performance evaluation, there is still no performance evaluation language that can describe the goal function of reinforcement learning. To tackle this issue, a novel language is proposed in the paper, which can not only describe the goal of reinforcement learning but also use reinforcement learning techniques to calculate the fixed points to get the target value of the corresponding formulas. In other words, it can describe the maximum and minimum of a given state or a particular path, and most importantly, it can also describe the mean value of all paths that satisfy the performance query formula.

As for future works, it can be briefly summarized as follows. First, we consider combining the novel language with model checking techniques towards the unification of model checking and performance evaluation. Second, there are already reinforcement learning methods for continuous-time models, so extending or modifying the language to adapt to continuous-time models is also a possible direction. Third, we aim at applying the novel language to more reinforcement learning cases, in which the language can be used to describe the goals of reinforcement learning.

Acknowledgements This work was supported in part by the Aviation Science Foundation of China under Grant 20185152035 and Grant 20150652008, in part by the National Natural Science Foundation of China under Grant 61572253, in part by the Fundamental Research Funds for the Central Universities (NJ2020022, NJ2019010, NJ20170007). The authors would also like to thank anonymous reviewers for their valuable comments.

References

1 Hermanns H, Herzog U, and Katoen J P. Process algebra for performance evaluation. Theoretical computer science, 2002, 274: 43-87
2 Herzog U. Formal methods for performance evaluation. In: School Organized by the European Educational Forum, Berg en Dal, The Netherlands, 2000. 1-37
3 Baier C, Haverkort B R, Hermanns H, et al. Performance evaluation and model checking join forces. Communications of the ACM, 2010, 53: 76-85
4 Haring G, Lindemann C and Reiser M. Performance evaluation: Origins and directions. Springer, 2003. 1-3
5 Zhang L, Li D, Xi Y, et al. Reinforcement learning with actor-critic for knowledge graph reasoning. Science China Information Sciences, 2020, 63: 1-3
6 Zhou T, Chen M, Yang C, et al. Data fusion using Bayesian theory and reinforcement learning method. Science China Information Sciences, 2020, 63: 1-3
7 Yan X, Zhu J, Kuang M, et al. Missile aerodynamic design using reinforcement learning and transfer learning. Science China Information Sciences, 2018, 61: 119204:1-119204:3
8 Droste M and Kuske D. Skew and infinitary formal power series. In: International Colloquium on Automata, Languages, and Programming, Eindhoven, The Netherlands, 2003. 426-438
9 Chatterjee K, Doyen L and Henzinger T A. Quantitative languages. In: International Workshop on Computer Science Logic, Bertinoro, Italy, 2008. 385-400
10 Chatterjee K, Doyen L and Henzinger T A. Quantitative languages. ACM Transactions on Computational Logic, 2010, 11: 1-38
11 Alfaro L, Faella M, Henzinger T A, et al. Model checking discounted temporal properties. In: International Conference on Tools and Algorithms for the Construction and Analysis of Systems, Barcelona, Spain, 2004. 77-92
12 Alfaro L, Faella M, Henzinger T A, et al. Model checking discounted temporal properties. Theoretical Computer Science, 2005, 345: 139-170
13 Alfaro L, Henzinger T A and Majumdar R. Discounting the future in systems theory. In: International Colloquium on Automata, Languages, and Programming, Eindhoven, The Netherlands, 2003. 1022-1037
14 Alfaro L, Faella M and Stoelinga M. Linear and branching metrics for quantitative transition systems. In: International Colloquium on Automata, Languages, and Programming, Turku, Finland, 2004. 97-109
15 Jamroga W. A temporal logic for Markov chains. In: Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems, Estoril, Portugal, 2008. 697-704

16 Jamroga W. A temporal logic for stochastic multi-agent systems. In: Proceedings of the 11th Pacific Rim International Conference on Multi-Agents, Hanoi, Vietnam, 2008. 239-250
17 Almagor S, Boker U, Kupferman O. Discounting in LTL. In: International Conference on Tools and Algorithms for the Construction and Analysis of Systems, Grenoble, France, 2014. 424-439
18 Jing Y P and Miner A S. Computation tree measurement language (CTML). Formal Aspects of Computing, 2018, 30: 443-462
19 Ballarini P, Barbot B, Duflot M, et al. HASL: A new approach for performance evaluation and model checking from concepts to experimentation. Performance Evaluation, 2015, 90: 53-77
20 Wang F J, Cao Z N, Tan L X and Li Z. Formal modeling and performance evaluation for hybrid systems: a probabilistic hybrid process algebra-based approach. International Journal of Software Engineeringand Knowledge Engineering, in press
21 Wang F J, Cao Z N, Wang S Y, et al. A Language for Performance Evaluation Based on the Combination of CTRML and MMTD and its Algorithm. In: 2021 IEEE International Conference on Intelligent Systems and Knowledge Engineering, in press
22 Davey, B A and Priestley, H A. Introduction to lattices and order. Cambridge: Cambridge University Press, 2002. 33-40
23 Gan Y Z, Zhang Z and Tan X Y. Stabilizing Q Learning Via Soft Mellowmax. In: Proceedings of the Thirty-Fifth AAAI Conference on Artificial Intelligence, Virtual Event, 2021. 7501–7509
24 Tarski A. A lattice-theoretical fixpoint theorem and its applications. Pacific journal of Mathematics, 1955, 5: 285-309
25 Sangiorgi D. On the origins of bisimulation and coinduction. ACM Transactions on Programming Languages and Systems (TOPLAS), 2009, 31: 1-41
26 Yamagata Y, Liu S, Akazaki T, Duan Y H and Hao J Y. Falsification of cyber-physical systems using deep reinforcement learning. IEEE Transactions on Software Engineering, 2021, 47: 2823-2840
27 Heitzinger C. Reinforcement Learning: Algorithms and Convergence. 2019.
28 Sutton R S and Barto A G. Reinforcement learning: An introduction. Cambridge: MIT press, 2018. 40-51
29 Watkins C J and Dayan P. Q-learning. Machine learning, 1992, 8: 279-292
30 Baier C and Katoen J P. Principles of model checking. Cambridge: MIT press, 2008. 229-433
31 Clarke E M, Grumberg O, Kroening D, et al. Model checking. Cambridge: MIT press, 2018. 27-32