Journal of KIIT. Vol. 21, No. 7, pp. 13-22, Jul. 31, 2023. pISSN 1598-8619, eISSN 2093-7571
http://dx.doi.org/10.14801/jkiit.2023.21.7.13

Generating Controller of GR(1) Synthesis and Reinforcement Learning in Game-Solving

Ryeonggu Kwon*[1], Gihwon Kwon*[2]

This work was supported by the Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korean government (MSIT) (No. 2021-0-00122, Safety Analysis and Verification Tool Technology Development for High Safety Software Development)

Abstract

This paper presents a comparative study of GR(1) Synthesis and Reinforcement Learning in the context of controller generation for the Moving Obstacle Evasion Problem. GR(1) Synthesis, a formal method in computer science, provides a systematic approach to automatically generate a controller that satisfies a given set of logical specifications. On the other hand, Reinforcement Learning, a type of machine learning, approximates the optimal solution by learning from the environment and updating its strategy based on a loss function. While GR(1) Synthesis guarantees an optimal solution, it may fail to generate a controller in certain configurations. In contrast, Reinforcement Learning can provide a suboptimal but feasible solution in all configurations. The results demonstrate the potential of Reinforcement Learning as a viable alternative for controller generation in scenarios where GR(1) Synthesis is unrealizable. This study contributes to the understanding of the strengths and limitations of both methods and provides insights for their application in system design.

요 약

이 논문은 이동 장애물 회피 문제에 대한 컨트롤러 생성을 위한 GR(1) Synthesis와 강화 학습의 비교 연구를 제시한다. 컴퓨터 과학의 정형 합성인 GR(1) Synthesis는 주어진 논리적 명세를 만족하는 컨트롤러를 자동으로 생성하는 체계적인 접근법을 제공한다. 반면에, 기계 학습의 한 유형인 강화 학습은 환경에서 학습하고 손실 함수를 기반으로 전략을 업데이트함으로써 최적의 해를 근사화한다. GR(1) Synthesis는 최적의 해를 보장하지만, 특정 구성에서는 컨트롤러를 생성하지 못할 수 있다. 반면에, 강화 학습은 모든 구성에서 최적이 아닌 실행 가능한 최선의 해를 제공할 수 있다. 결과적으로 GR(1) Synthesis가 실현 불가능한 시나리오에서 컨트롤러 생성을 위한 강화 학습의 잠재력을 보여준다. 이 연구는 두 방법의 장점과 한계에 대한 이해에 기여하고 시스템 설계에서의 그들의 응용에 대한 통찰력을 제공한다.

Keywords
gr(1) synthesis, reinforcement learning, controller synthesis, formal method, artificial intelligence

* Dept. of Computer Science, Kyonggi University
- ORCID[1]: https://orcid.org/0000-0002-4942-247X
- ORCID[2]: https://orcid.org/0000-0002-8221-4939

· Received: Jun. 23, 2023, Revised: Jun. 30, 2023, Accepted: Jul. 03, 2023
· Corresponding Author: Ryeonggu Kwon
Dept. of Computer Science, Kyonggi University, 154-42, Gwanggyosan-ro, Yeongtong-gu, Suwon-si, Gyeonggi-do, South Korea
Tel.: +82-31-249-9666, Email: rkkwon@kyonggi.ac.kr

Ⅰ. Introduction

Generalized Reactivity(1) (GR(1)) Synthesis has been a significant tool in the automatic generation of controllers that satisfy a given set of logical specifications[1][2]. Since its inception, it has provided a systematic approach to describe the desired behavior of a system in response to its environment. The theoretical foundation of GR(1) Synthesis lies in the intersection of logic, automata, and system design. It uses a fragment of Linear Temporal Logic(LTL)[3] to express properties about the future behavior of a system, translating these logical specifications into an automaton, a mathematical model of computation. The automaton represents the system's behavior as a state-transition graph, where the system and the environment interact. The goal of GR(1) Synthesis is to design a controller for the system that ensures the system's behavior satisfies the given specifications, regardless of the environment's actions.

On the other hand, Reinforcement Learning (RL)[4]-[6] is a type of machine learning where an agent learns to make decisions by interacting with its environment. The agent learns from the consequences of its actions, rather than from being explicitly taught, adjusting its behavior based on the positive or negative feedback it receives. This trial-and-error approach allows the agent to learn the optimal policy that maximizes the cumulative reward over time. Reinforcement learning has been successfully applied in various fields such as game playing, robotics, resource management, and many others.

From the perspective of system controller generation, comparing GR(1) Synthesis and Reinforcement Learning is meaningful. While GR(1) Synthesis provides an optimal solution by exploring all possible states of the system and the environment, Reinforcement Learning offers a suboptimal but feasible solution even in large state spaces where finding an optimal solution is computationally infeasible.

The contributions of this paper are manifold and significant to the field of controller generation. Firstly, this paper presents a novel comparative study of GR(1) Synthesis and Reinforcement Learning, two methods that have been traditionally studied in isolation. By juxtaposing these two methods in the context of the Moving Obstacle Evasion Problem, this paper provides new insights into their relative strengths and weaknesses. Secondly, this paper demonstrates, through empirical results, that Reinforcement Learning can serve as a viable alternative to GR(1) Synthesis in scenarios where the latter is unrealizable. This is a significant finding as it expands the range of problems that can be effectively addressed using Reinforcement Learning. Thirdly, this paper provides a detailed analysis of the performance of the controllers generated by both methods. The analysis reveals that while GR(1) Synthesis generates optimal controllers, Reinforcement Learning generates suboptimal but satisfactory controllers. This finding contributes to our understanding of the trade-offs between optimality and realizability in controller generation. Lastly, this paper suggests that in scenarios with large state spaces and where finding an optimal solution is challenging, Reinforcement Learning can be an appropriate choice. This is a valuable insight for practitioners in the field, guiding them in choosing the most suitable method for their specific problem setting. In sum, this paper makes significant contributions to the field by providing a comprehensive comparison of GR(1) Synthesis and Reinforcement Learning, demonstrating the viability of Reinforcement Learning as an alternative to GR(1) Synthesis, and offering valuable insights into the trade-offs in controller generation.

In conclusion, this paper provides a comparative study between GR(1) Synthesis and RL from the perspective of system controller generation, highlighting the strengths and weaknesses of both methods and offering insights into their applicability in different scenarios.

The structure of this paper is as follows. In Chapters 2 and 3, the theoretical backgrounds of GR(1) Synthesis and RL are introduced. Chapter 4 describes the experiment of solving the Moving Obstacle Evasion Problem using reinforcement learning. Finally, Chapter 5 concludes with a summary and conclusion.

Ⅱ. Generalized Reactivity(1) Synthesis

GR(1) synthesis is a formal method in computer science that provides a systematic approach to automatically generate a controller, a strategy, plan, or policy, that satisfies a given set of logical specifications. These specifications describe the desired behavior of a system in response to its environment. The theoretical foundation of GR(1) synthesis lies in the intersection of logic, automata, and system design. The specifications for the system are expressed in a logical formalism using variables to represent the state of the system and logical conditions to describe how these variables evolve over time. The logic used in GR(1) synthesis is a fragment of LTL, which allows expressing properties about the future behavior of a system.

The logical specifications are translated into an automaton, a mathematical model of computation. The automaton represents the system's behavior as a state-transition graph. In the context of GR(1) synthesis, the automaton is a game structure where two players, the system and the environment, interact. The goal of GR(1) synthesis is to design a controller for the system that ensures the system's behavior satisfies the given specifications, regardless of the environment's actions. This is achieved by constructing a winning strategy for the system in the game structure.

The term generalized reactivity(1) refers to the class of specifications that GR(1) synthesis can handle. These specifications consist of assumptions about the environment and guarantees about the system's behavior. The assumptions and guarantees are expressed as LTL formulas. The 1 in GR(1) indicates that the synthesis problem can be solved in polynomial time in the size of the game structure, making GR(1) synthesis a practical method for controller synthesis in many applications.

A GR(1) specification is a type of specification used in the field of formal methods, particularly in the context of controller synthesis. It is a fragment of LTL that allows for the expression of assumptions about the environment and guarantees about the system's behavior. The GR(1) specification is used to describe the desired behavior of a system in response to its environment.

The GR(1) specification is structured as follows: It consists of environment variables X and system variables Y. The environment variables represent the state of the environment in which the system operates, and the system has no control over these variables; they are determined by external factors. The system variables represent the state of the system, and the system has control over these variables and can change them in response to the environment.

The GR(1) specification also includes initial conditions ϕ_i, transition relations ϕ_t, and guarantees ϕ_g. The initial conditions are a propositional logic formula over $X \cup Y$ that specifies the initial state of the system and the environment. The transition relations is sub-formula $\phi_x \wedge \phi_y$, where ϕ_x is a formula in LTL over $X \cup Y \cup X'$ and ϕ_y is a formula in LTL over $X \cup Y \cup X' \cup Y'$. Here, X' and Y' represent the next state of the environment and system variables, respectively. ϕ_x specifies the possible transitions of the environment variables, and ϕ_y specifies the possible transitions of the system variables in response to the current state of the system and the environment.

The guarantees are a sub-formula $\phi_j \wedge \phi_b$, where

ϕ_j and ϕ_p are sets of LTL formulas over $X \cup Y$. The formulas in ϕ_j are called justice requirements, and the formulas in ϕ_p are called progress requirements. The justice requirements specify conditions that the system must satisfy infinitely often, and the progress requirements specify conditions that the system must eventually satisfy.

In the context of GR(1) specifications, the terms realizable and unrealizable refer to whether a controller can be synthesized that satisfies the given specifications.

A GR(1) specification is said to be realizable if there exists a controller that, for every possible behavior of the environment that satisfies the environment assumptions, can ensure the system behavior satisfies the system guarantees. In other words, a realizable specification is one for which there exists a winning strategy for the system in the game structure defined by the specification.

Formally, a GR(1) specification $\phi_i \wedge \phi_t \wedge \phi_g$ is realizable if there exists a function $f: X \rightarrow Y$ such that for every sequence $x_0, x_1, x_2, ...$ of environment states that satisfies the environment transition relation ϕ_x, the sequence $y_0, y_1, y_2, ...$ of system states defined by $y_i = f(x_i)\,\mathrm{for}\, all\, i \geq 0$ satisfies the system transition relation ϕ_y and the system guarantees ϕ_g.

On the other hand, a GR(1) specification is said to be unrealizable if no such controller exists. That is, no matter what strategy the system follows, there is some behavior of the environment that satisfies the environment assumptions but leads to a violation of the system guarantees. An unrealizable specification is one for which the environment has a winning strategy in the game structure defined by the specification.

III. Reinforcement Learning

RL is a subfield of machine learning that focuses on how an agent can learn to make optimal decisions by interacting with an environment. The fundamental goal of RL is to find a policy, denoted as π, which is a mapping from states to actions $\pi: S \rightarrow A$, that maximizes the expected cumulative reward over time. The agent, through a process of trial and error, learns to associate states of the environment with actions that yield the highest reward.

The general model of RL involves several key components: states S, actions A, and rewards R. The agent interacts with the environment in discrete time steps. At each time step t, the agent observes the current state of the environment $s_t \in S$, selects an action $a_t \in A(s_t)$ based on its policy π, receives a reward $r_t = R(s_t, a_t)$, and transitions to a new state s_{t+1}. The process continues until a termination condition is met.

In RL, the process of approximating a function often involves the use of a loss function. The loss function quantifies the difference between the predicted output of the function approximator and the actual output. The goal is to adjust the parameters of the function approximator to minimize this loss.

Let's denote the function approximator as f, which is parameterized by θ. Given an input state s, the function approximator predicts the value of each possible action a under the current policy π, denoted as $f(s, avert\theta)$.

The actual value of taking action a in state s under policy π, denoted as $Q \cdot \pi(s,a)$, is typically estimated using a technique such as Temporal Difference(TD) learning or Monte Carlo methods.

The loss function $L(\theta)$ is then defined as the squared difference between the predicted and actual action values:

$$L(\theta) = E[(Q \cdot \pi(s,a) - f(s, avert\theta))^2] \qquad (1)$$

The expectation E is taken over the distribution of states and actions experienced by the agent.

The parameters θ of the function approximator are updated to minimize this loss. This is typically done using gradient descent, a popular optimization algorithm. The update rule for gradient descent is:

$$\theta \leftarrow \theta - \alpha \cdot \nabla\theta \qquad (2)$$

where α is the learning rate, and $\nabla\theta$ is the gradient of the loss function with respect to the parameters θ.

By iteratively applying this update rule, the function approximator learns to better predict the action values, and thus the agent learns to make better decisions.

Ⅳ. Experimentation

4.1 Methodology

This research aims to compare the effectiveness and efficiency of GR(1) Synthesis and RL in generating controllers for the same problem. The experimental setup and procedure are detailed as follows:

1. The GR(1) Synthesis is performed using the Spectra[7] tool, which is an Eclipse plugin-based tool. The results obtained from the GR(1) Synthesis will be cited from relevant reports. For RL, we use the PyTorch-based Stable-baselines3[8] with the Proximal Policy Optimization (PPO)[9] algorithm.
2. The requirements of the identical problem are defined and programmed to suit RL. This step ensures that both GR(1) Synthesis and RL are applied to the same problem, allowing for a fair comparison.
3. The parameters of the problem are set for the RL process. And then the trained model is evaluated.
4. The outcomes of the RL approach are subsequently juxtaposed with those of the GR(1) Synthesis in several respects. One of the key comparisons made is to examine instances where the GR(1) Synthesis was deemed unrealizable and thus failed to generate a controller. In these specific scenarios, the question posed is whether RL could step in and successfully generate a controller where GR(1) Synthesis could not. This comparison provides a valuable perspective on the relative strengths and potential limitations of each method in the context of controller generation.

4.2 Problem formulation

The Moving Obstacle Evasion Problem[10] is defined on an n-size discrete grid where an agent must avoid a moving obstacle. Initially, the agent is located at grid (0, 0) and occupies one cell. In contrast, the obstacle occupies a 2x2 cell area and is defined by the top-left cell, thus it is located at (n-1, n-1). The agent and the obstacle take turns moving in one of four directions: left, right, up, or down. The agent can move twice in one turn, while the obstacle moves one cell after the agent has moved twice, under the basic settings. However, a glitch is introduced in this problem where the obstacle can violate the basic settings and move immediately after the agent has moved once. To account for this, a glitch count is defined, which allows the obstacle to violate the basic settings a certain number of times.

List 1 defines the variables necessary for RL of this problem, as well as the action space and observation space. In this context, SIZE represents the length of the cells along the x and y axes of the grid. GLITCHES denotes the number of glitches allowed for the obstacle. GLITCH_COUNTER keeps track of the number of glitches used so far. AGENT_TURN indicates the number of times the agent has moved in its turn. The ACTION_SPACE is discretized and takes a value from 0 to 4 at each step, representing STAY, LEFT, RIGHT, UP, and DOWN respectively. The OBSERVATION_SPACE is defined by the x and y coordinates of the obstacle and the x and y coordinates of the agent.

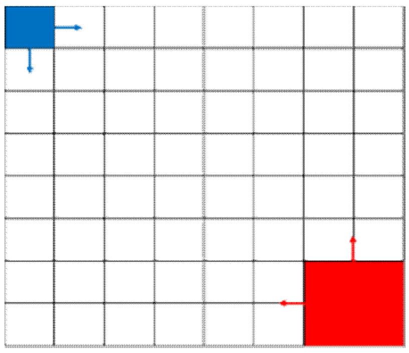
Fig. 1. Environment of the problem

List 1. Definitions of parameters

```
SIZE := N
GLITCHES := M
ACTION_SPACE := {STAY, LEFT, RIGHT, UP,
                 DOWN}
OBERVATION_SPACE := (N, N, N-1, N-1)
```

The procedure STEP(ACTION) is defined to represent a single step in the environment based on the action taken by the agent. The action is passed as a parameter to this procedure. The agent's location is updated based on the action taken by calling the UPDATE_AGENT_LOCATION procedure. The AGENT_TURN counter is then incremented to keep track of the number of moves made by the agent in its turn. A conditional check is performed to see if it's still the agent's turn (i.e., the agent has moved less than twice) and if the number of glitches used is less than the maximum allowed glitches. If both conditions are met, a random choice is made to decide if a glitch occurs. If a glitch occurs, the obstacle's location is updated by calling the UPDATE_OBSTACLE_LOCATION procedure. The GLITCHES_COUNTER is incremented to keep track of the number of glitches used so far, and the AGENT_TURN counter is reset to 0, indicating the start of a new turn for the agent. If it's not the agent's turn or if the maximum number of allowed glitches has been reached, the obstacle's location is updated, and the AGENT_TURN counter is reset to 0. Next, a check is performed to see if the agent and the obstacle are not in the same location or adjacent locations. If they are not, a reward of 1 is given, and the TERMINATED flag is set to False, indicating that the episode is not over. If the agent and the obstacle are in the same location or adjacent locations, the TERMINATED flag is set to True, indicating that the episode is over. Finally, the current state of the environment, the reward, and the TERMINATED flag are returned as the output of the STEP procedure. The state represents the current locations of the agent and the obstacle, the reward is the reward obtained in this step, and the TERMINATED flag indicates whether the episode is over.

List 2. Procedure for step function

```
Procedure STEP(ACTION):
  Call UPDATE_AGENT_LOCATION(ACTION)
  AGENT_TURN ← AGENT_TURN + 1

  If AGENT_TURN < 2 and GLITCHES_COUNTER
                          < GLITCHES Then
     IS_GLITCH ← {True, False}
     If IS_GLITCH is True Then
          Call UPDATE_OBSTACLE_LOCATION
          GLITCHES_COUNTER ←
                     GLITCHES_COUNTER + 1
          AGENT_TURN ← 0
     End If
  Else
     Call UPDATE_OBSTACLE_LOCATION
     AGENT_TURN ← 0
  End If

  If the agent and the obstacle are not in the same
     location or adjacent locations Then
     REWARD ← 1
     TERMINATED ← False
  Else
     TERMINATED ← True
  End If

  Return STATE, REWARD, TERMINATED
End Procedure
```

4.3 Result and analysis

We conducted RL using PPO for a total of 12 configurations, ranging from a configuration with a grid size of 8 and 1 glitch count to a configuration with 64 and 30 respectively. The results can be seen in Figure 2. From the graph, we were able to obtain a controller that can solve the problem for all configurations. Since the agent receives a reward of 1 for each successful step, the total amount of reward obtained can be understood as equivalent to the number of steps taken.

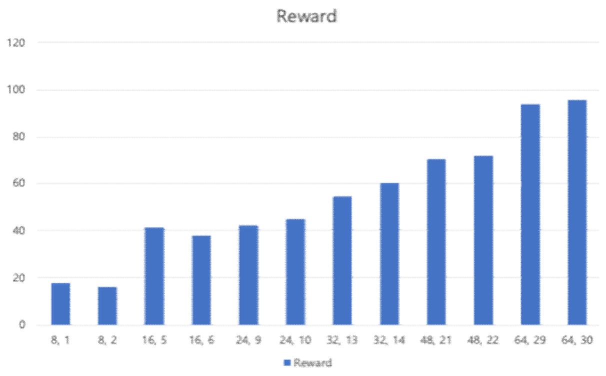

Fig. 2. Evaluation result

We will compare this with the results obtained using GR(1) Synthesis for the same problem. The following Table 1 presents a comparison with the results of RL.

Out of a total of 12 configurations, synthesis was only possible in 6 configurations using GR(1) Synthesis. On the other hand, RL was able to obtain a controller that solved the problem in all configurations. GR(1) Synthesis algorithmically explores all states of obstacles and agents to generate a controller that realizes the agent's goal. This controller never gets caught by obstacles over infinite behaviors (in other words, infinite sequences of discretized states). Due to this characteristic, the controller generated by GR(1) Synthesis fully realizes the requirements. However, since GR(1) Synthesis checks the realizability for all states of the system and environment, if there exists even a single case that is unrealizable (i.e., the system's goal cannot be achieved), it fails to generate a controller at all.

Table 1. Comparison with GR(1) synthesis and RL

Size	Glitches	Synthesizable (GR(1) synthesis)	Average steps (RL)
8	1	Yes	17
8	2	No	16
16	5	Yes	41
16	6	No	37
24	9	Yes	42
24	10	No	44
32	13	Yes	54
32	14	No	60
48	21	Yes	70
48	22	No	71
64	29	Yes	93
64	30	No	95

The controller obtained through RL was able to solve all problems. However, if the controller generated by GR(1) Synthesis is considered optimal, then the controller obtained through RL is the best. This is because it approximates the functions of the environment and system, and there can certainly be cases where the agent's goal cannot be achieved. Therefore, as seen in Table 1, the average steps of the agent for each configuration are presented. This can be understood as the minimum performance guaranteed in that configuration (in this case, not being caught by obstacles). However, since it is not a controller that is realizable in infinite behaviors like GR(1) Synthesis, measures will be needed for exceptions or situations where the agent's goal fails.

In summary, the theoretical difference between GR(1) Synthesis and RL lies in how they model the input-output relation of the environment and the system.

GR(1) Synthesis generates an optimal function for the input-output relation of the system. This function defines the optimal action that the system should take for all possible states of the environment. The advantage of this method is that it predefines the system's actions for all possible environmental states, so there is little computation needed to make decisions at runtime. However, the downside of this method is that it requires a lot of computation to generate this function if the state space is large, as it needs to predefine actions for all possible environmental states.

On the other hand, RL generates a function that approximates the input-output relation of the system, and this function is updated iteratively using a loss function. This function predicts the action that the system should take for a given environmental state. The advantage of RL is that it can learn efficiently even if the state space is large. However, the downside of this method is that it requires a lot of trial and error in the learning process, and there is no guarantee that the learned policy is optimal.

Therefore, the choice between GR(1) Synthesis and RL depends on the characteristics of the problem and the available computational resources. GR(1) Synthesis is suitable for cases where the state space is small and an optimal solution needs to be found, while RL is suitable for cases where the state space is large and finding an approximate solution is sufficient.

Ⅴ. Conclusion

In conclusion, this paper presented a comparative study of GR(1) Synthesis and RL in the context of controller synthesis for the Moving Obstacle Evasion Problem. The study demonstrated that while GR(1) Synthesis provides an optimal solution when it is realizable, it fails to generate a controller in certain configurations. On the other hand, RL, though not providing an optimal solution, is capable of generating a controller in all configurations, offering the best-effort solution.

The research highlighted the strengths and weaknesses of both methods. GR(1) Synthesis, grounded in formal methods, guarantees the realization of the system's goals under all possible behaviors of the environment, given that the synthesis is realizable. GR(1) Synthesis faces a challenge when the state space of the environment and system is large, as the computational load increases exponentially. This is due to the fact that GR(1) Synthesis attempts to predefine the controller's actions for all possible states of the environment and system. While this method of predefining actions for all states is effective when the state space is small, it becomes computationally intensive and difficult to solve real problems as the state space grows.

On the other hand, RL can address this limitation of GR(1) Synthesis. Reinforcement learning provides a method that can efficiently learn even when the state space is large. RL approximates a function of the states of the environment and system, predicting the actions the system should take given a state. By approximating this function, RL can efficiently solve problems even when the state space is large. However, the solutions generated by RL do not offer the same guarantees as those generated by GR(1) Synthesis.

The primary contribution of this paper is the demonstration of the applicability of Reinforcement Learning (RL) in scenarios where GR(1) Synthesis is unrealizable. This opens up new possibilities for controller synthesis in complex environments where traditional formal methods may fall short. Furthermore, this study shows that RL succeeds in all configurations, providing solutions even in configurations where GR(1) Synthesis fails to generate a controller, albeit with suboptimal solutions. This suggests that RL can be an appropriate choice in scenarios with large state spaces and where finding an optimal solution is challenging. Additionally, this study provides valuable insights for researchers and practitioners in the field of controller synthesis, offering a new perspective on the potential of RL as a complementary approach to formal methods. These results demonstrate that RL can overcome the limitations of GR(1) Synthesis and serve as a powerful tool for generating controllers in larger state spaces and more complex problems. This presents a new paradigm for controller synthesis and is expected to stimulate further research in this field.

In terms of future work, this study opens up several promising directions. Firstly, while the current study has focused on the Moving Obstacle Evasion Problem, the comparative approach of GR(1) Synthesis and Reinforcement Learning could be extended to other types of control problems. It would be interesting to investigate how the two methods perform in different problem settings and whether the advantages of Reinforcement Learning observed in this study hold in other contexts. Secondly, the Reinforcement Learning method used in this study is a basic implementation. There are many advanced Reinforcement Learning algorithms and techniques that could potentially improve the performance of the controller. Future work could explore the use of these advanced methods and compare their performance with GR(1) Synthesis. Thirdly, this study has used a specific set of parameters for the Reinforcement Learning algorithm. The choice of parameters can significantly affect the performance of the algorithm. Future work could investigate the impact of different parameter settings on the results. Lastly, this study has considered a deterministic environment. In many real-world situations, the environment is stochastic and unpredictable. Future work could explore how GR(1) Synthesis and Reinforcement Learning perform in such stochastic environments. By exploring these directions, future work can build on the findings of this study and further our understanding of the strengths and weaknesses of GR(1) Synthesis and Reinforcement Learning in controller generation.

References

[1] R. Majumdar, N. Piterman, and A.-K. Schmuck, "Environmentally-Friendly GR (1) Synthesis", TACAS 2019: Tools and Algorithms for the Construction and Analysis of Systems, Vol. 11428, pp. 229-246, Apr. 2019. https://doi.org/10.1007/978-3-030-17465-1_13.

[2] U. Klein and A. Pnueli, "Revisiting Synthesis of GR(1) Specifications", HVC 2010: Hardware and Software: Verification and Testing, Vol. 6504, pp. 161-181, 2010. https://doi.org/10.1007/978-3-642-19583-9_16.

[3] A. Pnueli, "The temporal logic of programs", 18th Annual Symposium on Foundations of Computer Science (sfcs 1977), Providence, RI, USA, pp. 46-57, Oct. 1977. https://doi.org/10.1109/SFCS.1977.32.

[4] R. S. Sutton and A. G. Barto, "Reinforcement Learning: An Introduction", IEEE Transactions on Neural Networks, Vol. 9, No. 5, pp. 1054, Sep. 2018. https://doi.org/10.1109/TNN.1998.712192.

[5] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement Learning: A Survey", Journal of Artificial Intelligence Research, Vol. 4, pp. 237-285, May 1996, https://doi.org/10.1613/jair.301.

[6] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey", The International Journal of Robotics Research, Vol. 32, No. 11, pp. 1238-1274, Aug. 2013. https://doi.org/10.1177/0278364913495721.

[7] S. Maoz and J. O. Ringert, "Spectra: a specification language for reactive systems", Software and Systems Modeling, pp. 1553-1586, Apr. 2021. https://doi.org/10.1007/s10270-021-00868-z.

[8] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations", The Journal of Machine Learning Research, Vol. 22, No. 1, pp. 12348-12355, Jan. 2021.

[9] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms", arXiv preprint arXiv:1707.06347, Jul. 2017. https://doi.org/10.48550/arXiv.1707.06347.

[10] M. Yossef, "Spectra Example: Moving Obstacle Evasion Problem", 2020. [accessed: Jun. 10, 2023]

Authors

Ryeonggu Kwon

2011 : BS degree in Department of Computer Science, Kyonggi University

2013 : MS degree in Department of Computer Science, Kyonggi University

2013 ~ Present : Ph.D. candidate

Research interests : software engineering, formal verification, formal synthesis, artificial intelligence

Gihwon Kwon

1985 : BS degree in Department of Computer Science, Kyonggi University

1987 : MS degree in Department of Computer Science, Chung-Ang University

1991 : Ph.D. degree in Department of Computer Science, Chung-Ang University

1991 ~ Present : Professor, Department of Computer Engineering, Kyonggi University

2006 ~ 2007 : Visiting Professor, Department of Computer Science, Carnegie Mellon University

2014 ~ 2016 : President, Software Engineering Society, Korean Institute of Information Scientists and Engineers

2021 ~ Present : Director, SW Central University Project Team, Kyonggi University

2022 ~ Present : Dean, College of Software Business, Kyonggi University

Research interests : Software Engineering, Software Safety