Reinforcement Learning Guided Symbolic Execution

Jie Wu
East China Normal University
Shanghai, China
51184501162@stu.ecnu.edu.cn

Chengyu Zhang
East China Normal University
Shanghai, China
dale.chengyu.zhang@gmail.com

Geguang Pu
East China Normal University
Shanghai, China
ggpu@sei.ecnu.edu.cn

***Abstract*—Symbolic execution is an indispensable technique for software testing and program analysis. Path-explosion is one of the key challenges in symbolic execution. To relieve the challenge, this paper leverages the Q-learning algorithm to guide symbolic execution. Our guided symbolic execution technique focuses on generating a test input for triggering a particular statement in the program. In our approach, we first obtain the dominators with respect to a particular statement with static analysis. Such dominators are the statements that have to be visited before reaching the particular statement. Then we start the symbolic execution with the branch choice controlled by the policy in Q-learning. Only when symbolic execution encounters a dominator, it returns a positive reward to Q-learning. Otherwise, it will return a negative reward. And we update the Q-table in Q-learning accordingly. Our initial evaluation results indicate that in average more than 90% of exploration paths and instructions are reduced for reaching the target statement compared with the default search strategy in KLEE, which shows the promise of this work.**

***Index Terms*—symbolic execution, reinforcement learning, debugging**

I. Introduction

Symbolic execution was first proposed by King [1] in 1976, which is a widely used technique for software testing and program analysis [2]. Path-explosion is one of the key challenges in symbolic execution [2] as the process to explore all the paths, which is usually exponential to the number of the static branches, is extremely time-consuming [2], [3].

In this paper, we propose a reinforcement learning guided symbolic execution framework which focuses on the problem of generating a test input for triggering a particular statement in the program. This problem arises in many debugging scenarios. For example, a developer may receive a bug report of an error at a particular line in one program (e.g. an assertion failure that resulted in an error message at that line), lacking a test case correspondingly [4]. In our approach, we combine the Q-learning algorithm [5], one of the algorithms of reinforcement learning, with KLEE [6], a symbolic execution tool, to efficiently generate test input for a statement. We summarize the contributions of this paper as follows:

1) We propose a novel guided symbolic execution algorithm which leverages Q-learning to guide the symbolic execution to a particular program statement.
2) We have realized our algorithm with KLEE and a light-weight Q-learning framework.
3) We evaluate the realization using software verification competition benchmark programs [7]. The initial evaluation results show that more than 90% of exploration paths and instructions are reduced with our algorithm , which shows the promise of this work.

We discuss our detailed methodology in Section II, the evaluation in Section III, and we conclude this work in Section IV.

II. Methodology

We apply the Q-learning to the symbolic execution. Q-learning is a model-free off-policy algorithm for estimating the long-term expected return of executing an action from a given state [5]. The expected return is called Q-values and a higher Q-value means that taking action A_t will yield better long-term results in state S_t. Algorithm 1 describes the Q-learning algorithm [5], [8]. The terms used are explained below:

- $Q(S, A)$: a table with horizontal coordinate S and vertical coordinate A.
- S: current state of the agent.
- A: current action picked according to some policy.
- S': next state where the agent ends up.
- A': next best action to be picked using current Q-value.
- R: current reward observed from the environment in response of current action.
- γ: discounting factor for future rewards.
- α: steps length taken to update the estimation of $Q(S, A)$.

Algorithm 1 Q-learning algorithm

```
Initialize: $Q(S, A)$
Repeat (for each episode)
  Initialize $S$;
  Repeat (for each step of episode):
    Choose $A$ from $S$ using policy derived from $Q$;
    Take action $A$, observe $R$, $S'$;
    $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_{A'} Q(S', A') - Q(S, A)]$;
    $S \leftarrow S'$;
  Until: $S$ is terminal
```

In our case, the state and action correspond to one line of statement and true or false when symbolic execution encounters a branch statement, respectively. The reward of our Q-learning is based on the dominators which are the statements

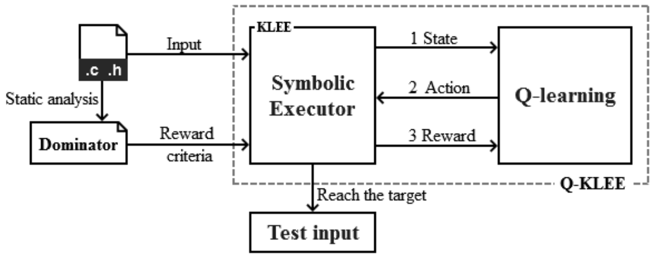

Fig. 1. Workflow of the reinforcement learning guided symbolic execution.

have to be visited before reaching the target statement. Only when symbolic execution encounters a dominator, Q-learning will receive a positive reward.

We realize our methodology using the symbolic execution tool KLEE [6] and a light-weight Q-learning framework by python. Figure 1 shows the interaction process. The right part is Q-learning guided KLEE (Q-KLEE). First, we manually set a target statement in the source code and then obtain the dominator of the target statement by static analysis. Then, when KLEE encounters a branch statement, it will send current information of state to Q-learning and get an instruction suggesting which branch (true or false) to move forward. According to ϵ-greedy policy [5], KLEE has a probability of ϵ to follow the instruction and another probability of $1 - \epsilon$ to choose a random action. KLEE returns a reward to Q-learning after every branch statement. Only when KLEE encounters a dominator, it returns a positive reward. Otherwise, it will return a negative reward. And we update the Q-table accordingly.

The execution would be restarted if it has been terminated but not yet reached the target statement. In this case, KLEE would send a *fail* signal to Q-learning. Q-learning then chooses a new execution state for KLEE, which is selected by three steps: 1) List out all possible other candidate states in the former execution process. 2) Calculate the probability of each candidate state. 3) Choose the one with maximum probability as the new state. Let us use 0 and 1 to represent *true* and *false* for convenience and make an example here. The execution terminates in the first round without reaching the target line and has a branch selection sequence of 101 (i.e. KLEE chooses *true*, *false*, *true*). Since every fork generates two states, the candidate states are 0, 11, and 100. The probability of the current state is the multiplication of probabilities of all the former states (e.g. $P(100) = P(1) * P(1|0) * P(10|0)$). We then calculate all the probabilities and choose the maximum. This process repeats until the target statement is reached or meeting the time-bound.

III. Evaluation

A. Experiment Setup

We use the software verification competition benchmark programs [7]. We choose the programs in reach-safety category which has the error labels. Our target statements are the error labels. We compare the time, number of paths and instructions needed in KLEE default strategy and Q-learning guided strategy for reaching the target statement. The execution terminates once reaching the target statement or meeting the 2-minute time-bound.

B. Initial Results

We successfully ran 60 programs except for timeout cases. Table I shows the evaluation results of KLEE and Q-learning guided KLEE, where *# Paths* represents the average paths of 60 programs, *# Instructions* represents the average explored instructions of 60 programs and *# Time* represents the average running time of 60 programs. It shows that Q-KLEE is able to reduce more than 90% of exploration paths and instructions for reaching the error labels while it takes more time. The reasons for extra time consumption concluded as follows: 1) We use the socket technology to link the symbolic executor and Q-learning and every interaction process takes time. 2) Q-learning calculation takes time.

TABLE I
Evaluation results of KLEE and Q-KLEE.

KLEE			Q-KLEE		
# Paths	# Instructions	# Time(s)	# Paths	# Instructions	# Time(s)
1,489	1,020,765	13.68	139	90,892	18.34

IV. Conclusions

In this paper, we propose a reinforcement learning guided symbolic execution algorithm to relieve the path-explosion problem. We have realized it using the symbolic execution tool KLEE. Our initial evaluation results show that our algorithm can significantly reduce the number of paths and instructions needed to reach a target statement, indicating the potential of combining the symbolic execution with machine learning algorithm. For future work, we would like to explore how to design a more effective and efficient reinforcement learning algorithm especially for symbolic execution and integrate the algorithm inside the symbolic execution tool to reduce the time consumption.

References

[1] J. C. King, "Symbolic execution and program testing," *Communications of the ACM*, vol. 19, no. 7, pp. 385–394, 1976.
[2] C. Cadar and K. Sen, "Symbolic execution for software testing: three decades later." *Commun. ACM*, vol. 56, no. 2, pp. 82–90, 2013.
[3] P. Boonstoppel, C. Cadar, and D. Engler, "Rwset: Attacking path explosion in constraint-based test generation," in *TACAS*, 2008, pp. 351–366.
[4] K.-K. Ma, K. Y. Phang, J. S. Foster, and M. Hicks, "Directed symbolic execution," in *International Static Analysis Symposium*. Springer, 2011, pp. 95–111.
[5] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.
[6] C. Cadar, D. Dunbar, and D. Engler, "Klee: Unassisted and automatic generation of high-coverage tests for complex systems programs," in *OSDI*, 2008, pp. 209–224.
[7] SoSy-Lab. (2019) Collection of verification tasks. [Online]. Available: https://github.com/sosy-lab/sv-benchmarks
[8] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of artificial intelligence research*, vol. 4, pp. 237–285, 1996.