Università di Genova | DIPARTIMENTO INFORMATICA, BIOINGEGNERIA, ROBOTICA E INGEGNERIA DEI SISTEMI

Computer Science Workshop
PhD program in Computer Science and Systems Engineering

IFRIT: Focused Testing through Deep Reinforcement Learning

Andrea Romdhana, Mariano Ceccato, Alessio Merlo, Paolo Tonella

Introduction

Software is constantly changing as developers add new features or make changes. This directly impacts the effectiveness of the test suite associated with that software, especially when the new modifications are in an area where no test case exists. Our article addresses the issue of developing a high-quality test suite to repeatedly cover a given point in a program, with the ultimate goal of exposing faults affecting the given program point. Our approach, IFRIT, uses Deep Reinforcement Learning to generate diverse inputs while keeping a high level of reachability of the desired program point. IFRIT achieves better results than state-of-the-art and baseline tools, improving reachability, diversity and fault detection.

Main Contributions

Our paper gives the following major contributions to the state-of-the-art:

- The first Deep RL approach to focused testing. By relying just on runtime coverage feedback, this approach is applicable to a wide range of programs.
- IFRIT, an open source tool, whose code is available at the url: *https://github.com/H2SO4T/IFRIT*.
- An empirical study showing the effectiveness of our approach in comparison with existing and baseline techniques.

Materials and Methods

0.1 Problem Definition

Given the space of the program inputs X, we denote by X_{pp} the sub-space of inputs whose execution traverses pp. A focused test input generator aims at producing inputs x that belong to X_{pp}. In addition to this, the generator should produce a *diverse* set of inputs that belongs to X_{pp}.

0.2 Overview

IFRIT (reInFoRcement learnIng for focused Testing) is an approach to focused testing based on Deep RL. Figure 1 shows an overview of IFRIT. The RL *environment* is represented by a program P under test, which is subject to several interaction steps. The objective is to generate inputs that reach a program point $pp \in P$ (see Section II). At each time step, IFRIT observes the code coverage measured on the program, computes the state s_t, the reward r_t, and chooses an action a_t used to generate a new input for the program. Then, it iterates, receiving the next code coverage s_{t+1} and reward r_{t+1}. Intuitively, if the new state s_{t+1} reaches the target point in the program with a new input, the reward is positive; it is neutral if such input is not new. Otherwise, if the target is not reached, the reward is negative. The reward is used to update the neural network, which learns how to guide the Deep RL algorithm to generate useful inputs for the program.

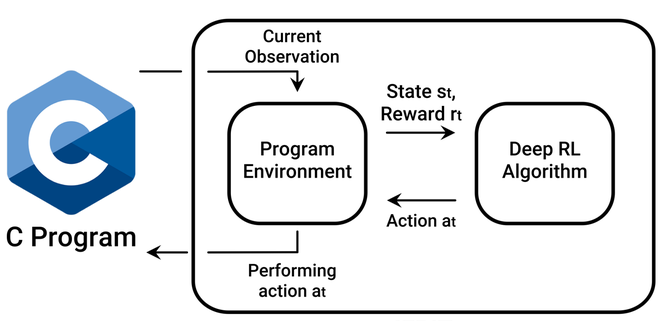

Figure 1: The IFRIT testing workflow. Code coverage is extracted (e.g., by gcovr, a utility for generating summarized code coverage results), from which IFRIT generates the state s_t and then determines the reward r_t for the chosen action a_t. By choosing an action a_t, IFRIT generates a new input, that stimulates the program under test.

0.3 Approach

To apply RL, we have to map the problem of focused testing to the standard mathematical formalization of RL: an MDP, defined by the 5-tuple, $\langle S, A, R, P, \rho_0 \rangle$. Moreover, we have to map the testing problem onto an RL task divided into several finite-length episodes.

State Representation. The state $s_t \in S$ is defined as a combined state $(b_0, ...b_n, i_0, ...i_m)$. The first part of the state $b_0, ...b_n$ represents the frequency of branch coverage during the last program execution, i.e., b_i is equal to $k > 0$ if the i-th branch of the program has been taken k times; it is equal to 0 if it was not traversed at all. The second part of the state vector, $i_0, ...i_m$ is equal to the last vector of input values generated by IFRIT. This means that the last execution of program P was performed by calling $P(i_0, ..., i_m)$, where $\langle i_0, ...i_m \rangle$ is called the *input vector*.

Action Representation. Each time IFRIT takes an action, it manipulates a previously generated input (e.g., a number or a string) by using modifiers. Modifiers mutate an input value based on the type of such input. Hence, an action $a = \langle a_0, a_1, a_2 \rangle$ is 3-dimensional: the first component a_0 encodes the index of the input vector to be manipulated. In fact, in the general case a program accepts a vector of input values as input and an action a will manipulate only one of them. The second component a_1 specifies the data to use to manipulate the input, depending on the context. The third component a_2 specifies how to use the second component on the input, i.e., what operation to apply using the second component as a parameter for such operation.

Reward Function. The RL algorithm used by IFRIT receives a reward $r_t \in R$ every time it executes an action a_t. We define the following reward function:

$$r_t = \begin{cases} \Gamma_1 & \text{if } input(s_{t+1}) \notin inputs(E_j) \ \wedge \\ & x \in X_{pp} \\ \Gamma_0 & \text{if } input(s_{t+1}) \in inputs(E_j) \ \wedge \\ & x \in X_{pp} \\ -\Gamma_1 & x \notin X_{pp}. \end{cases} \quad (1)$$

with E_j the current episode and $\Gamma_1 > \Gamma_0 \geq 0$ (in our implementation $\Gamma_0 = 0$, $\Gamma_1 = 1$). The exploration of IFRIT is divided into *episodes*. At time t, the reward r_t is positive (Γ_1) if IFRIT was able to reach the selected program point pp with an input never generated during the current episode E_j (i.e., the current input does not belong to the set of inputs generated so far in E_j): if a new episode E_{j+1} is started at $t + 1$, its set of inputs is reset: $inputs(E_{j+1}) = \emptyset$. When an input that reaches the target has already been generated before in the same episode, the reward is zero (Γ_0), as it is no more useful during the current episode, but it remains a good input for the given task. Resetting the set of generated inputs at the beginning of each new episode is a technique that encourages IFRIT to generate a high number of different inputs in each episode, which in turn makes the reward positive several times in the episode.

Conclusions

IFRIT improves the quality of fault detection and mutation killing when a focused test suite that reach a given target is to be generated automatically. By using Deep RL, our approach enhances the diversity of the test suite, making the input distribution close to a uniform distribution. Empirical results show that the quality of the test suites generated by IFRIT significantly outperforms random generation and the state-of-the-art tool DFT [1]. Moreover, IFRIT is faster in generating big test suites since it does not rely on symbolic execution.

References

[1] Héctor D Menéndez, Gunel Jahangirova, Federica Sarro, Paolo Tonella, and David Clark. Diversifying focused testing for unit testing. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 30(4):1–24, 2021.

CONTACTS

Andrea Romdhana
andrea.romdhana@dibris.unige.it