Negative Test Case Generation from an Extended Place/Transition Net-Based Mutants

Tomohiko Takagi
Faculty of Engineering, Kagawa University
2217-20 Hayashi-cho, Takamatsu-shi, Kagawa 761-0396, Japan

Tetsuro Katayama
Institute of Education and Research for Engineering, University of Miyazaki
1-1 Gakuen-kibanadai nishi, Miyazaki-shi, Miyazaki 889-2192, Japan
E-mail: takagi@eng.kagawa-u.ac.jp, kat@cs.miyazaki-u.ac.jp

Abstract

In negative testing, the large state space and feasibility problems cause the difficulty of generating negative test cases from EPN (Extended Place/transition Net)-based mutants. This paper shows the overview of EPN-based mutants, and the details of our negative test case generation technique. In the technique using ACO (Ant Colony Optimization), ants heuristically search better paths (i.e., negative test cases) to find foods (i.e., intended failures) on a field (i.e., an EPN-based mutant).

Keywords: Software Testing, Negative Testing, Model-Based Testing, Test Case Generation

1. Introduction

EPN (Extended Place/transition Net)-based mutants are formal behavioral models of software that contain intended failures.[1] In negative testing, well-selected EPN-based mutants are used to generate negative test cases to confirm that software does not include serious or possible failures. However, the large state space and feasibility problems on EPNs cause the difficulty of generating the negative test cases from EPN-based mutants.

The aim of this research is to construct a technique in which effective negative test cases to reach intended failures are generated from EPN-based mutants. In this technique using ACO (Ant Colony Optimization), ants heuristically search better paths (that is, negative test cases) to find foods (that is, intended failures) on a field (that is, an EPN-based mutant). This paper shows the overview of EPN-based mutants in section 2, the details of our negative test case generation technique in section 3, discussion in section 4, and, conclusion and future work in section 5.

2. Overview of EPN-Based Mutants

This section shows the overview of EPN-based mutants.

An EPN is a PN (Place/transition Net) that is extended by VDM++ (one of specification description languages that are used in VDM)[2] in order to formally define actions and guards on transitions, and it is suitable to represent the essential behavior of complex software. A simple example of an EPN is given in Fig. 1. This example has three places (p_1, p_2, and p_3) and four transitions (t_1, t_2, t_3, and t_4), and some transitions have guards and actions. For example, t_3 has "$v>=p$" as a guard and "$v:=v-p;$" as an action. v is an instance variable of nat type that was initialized to 0, and p is an input parameter (an argument) of nat type for t_3. When p_2 contains one or more tokens and also the guard of t_3 is satisfied, the firing of t_3 can be done with the execution of its action. Thus guards cause the feasibility problem

when specific transitions need to be selected to construct good test cases. Additionally, a state is expressed as a set of a marking (an array of the number of tokens on places) and values of instance variables, and therefore the introduction of instance variables accelerates the large state space problem.

EPN-based mutants (hereinafter, referred to as mutant models) are EPNs that contain intended failures of software, and they can be constructed by applying one or more existing mutation operators to an EPN that represents the expected behavior of software (hereinafter, referred to as an original model). Fig. 2 shows a simple example of a mutant model that was constructed by an arc insertion operator.

3. Negative Test Case Generation Technique

A negative test case in this study is a sequence of successive markings and transitions that starts from an initial marking and ends with an occurrence of an intended failure. It can be created based on a mutant model that were discussed in the previous section. For example, the mutant model shown in Fig. 2 can produce a negative test case "{[0,2,0], v=0} → t_1(2) → {[1,1,0], v=2} → t_3(1) → {[1,1,1], v=1}". The last marking of this example negative test case originally should be [1,0,1], and it succeeds in reaching the intended failure. Effective negative test cases are created as shorter sequences to reach intended failures, so as to be able to be executed by test engineers within a limited period of time. However, the large state space and feasibility problems that were discussed in the previous section cause the difficulty of constructing such negative test cases.

In this section, we propose an effective negative test case generation technique using ACO. ACO is one of meta-heuristic algorithms, and it is suitable to solve complex and time-consuming problems. In this technique, ants heuristically search better paths (that is, shorter negative test cases) to find foods (that is, intended failures) on a field (that is, a mutant model).

This technique consists of the following six steps.

(i) a ants are created (a is a natural number), and they are placed onto an initial marking of a mutant model. Each ant remembers the initial values of instance variables of the mutant model.
For example, when the mutant model shown in Fig. 2 is given to this technique, ants are placed onto [0,2,0], and they remember "v=0".

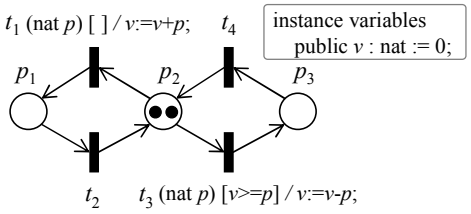

Fig. 1. Simple example of an EPN.

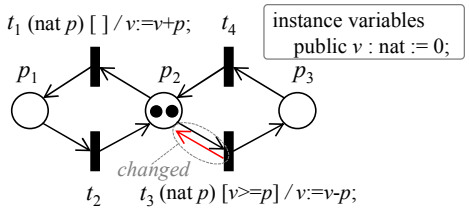

Fig. 2. Simple example of EPN-based mutants.

(ii) Each ant starts searching a path as follows.
First, an ant finds all feasible transitions on its current state (that is, a marking that the ant currently stays on, and values of instance variables that the ant remembers). If the ants find a new 3-tuple (*from-marking*, *transition*, *to-marking*), the pheromone value of the 3-tuple is initialized to an initial pheromone value τ_0 (τ_0 is 0 or a positive real number). The ant then selects a feasible transition at random in proportion to the pheromone values of all feasible 3-tuples, and goes to a next marking. If the selected transition has actions, the ant executes the actions and updates the values of instance variables that the ant remembers.
For example, in Fig. 2, an ant on the state {[1,1,0], v=2} finds feasible transitions t_1, t_2 and t_3. When the pheromone values of the 3-tuples ([1,1,0], t_1, [2,0,0]), ([1,1,0], t_2, [0,2,0]) and ([1,1,0], t_3, [1,1,1]) are τ_1, τ_2 and τ_3, respectively, the probabilities of selecting t_1, t_2 and t_3 are $\tau_1/(\tau_1+\tau_2+\tau_3)$, $\tau_2/(\tau_1+\tau_2+\tau_3)$ and $\tau_3/(\tau_1+\tau_2+\tau_3)$, respectively.
The ant repeats the above-mentioned selection of a transition, until an intended failure has appeared or the length of a path from the initial marking (that is, the number of transitions that the ant has selected) has reached l (l is a natural number) or the ant has reached a final marking (the definition of a final marking is optional). In this technique, an intended

failure appears if there are differences on markings, values of instance variables, and feasible transitions between the mutant model and its original model.

(iii) If there are ants that have succeeded in reaching an intended failure (hereinafter, referred to as effective ants), an ant that has found the shortest path among the effective ants (hereinafter, referred to as an iteration-best ant) is found. If the path of the iteration-best ant is shorter than the path of a best-so-far ant (that is, an effective ant that has found the shortest path through all the previous iterations in this technique), the iteration-best ant becomes a new best-so-far ant, and it is kept as a candidate solution.

(iv) If the number of iterations of this technique reaches i (i is a natural number), the path of the best-so-far ant is determined as a final solution (that is, the best negative test case on the mutant model), but if a best-so-far ant does not exist (that is, if there are no effective ants through all iterations), it is concluded that the mutant model cannot produce a negative test case.

(v) The pheromone values of all 3-tuples that have been found are updated by the following equation.

$$\tau_t(x+1) = (1-\rho)\cdot\tau_t(x) + \sum_{k=1}^{a} \Delta\tau_t^k \quad (1)$$

In Eq. (1), $\tau_t(x)$ expresses a pheromone value of a 3-tuple t in xth iteration. ρ is a positive real number that is less than 1.0, and it means an evaporation rate of pheromone. $\Delta\tau_t^k$ expresses a pheromone value that an ant k adds onto a 3-tuple t, and it is given by the following equation.

$$\Delta\tau_t^k = \begin{cases} \frac{1}{L_k}, & \text{if the ant } k \text{ is effective and } t \in P_k \\ 0, & \text{otherwise} \end{cases} \quad (2)$$

In Eq. (2), P_k is a path that has been found by an ant k, and L_k is the length of P_k (that is, the number of transitions of P_k).

(vi) All the ants are removed from the markings of the mutant model, and this technique returns to step (i).

This technique has five parameters (a, τ_0, l, i, and ρ) to adjust its performance. There are no parameter values that are appropriate for all software development projects, and therefore test engineers need to determine them by using a trial-and-error method.

4. Discussion

We developed a prototype of a tool that supports this technique, and applied it to examples on a trial basis. In this section, we discuss the effectiveness of this technique based on its experimental results.

The examples are based on an EPN of an electronic money charging system that was introduced to discuss a positive testing technique in Ref. 3. That is, four mutant models were manually constructed by applying four kinds of mutation operators to the EPN. The mutation operators used in this experiment were two kinds of model-based mutation operators (arc insertion and arc omission) and two kinds of code-based mutation operators (numeric constant replacement and inequality sign replacement), and they were applied to different transitions of the EPN.

Subsequently, we applied the prototype of the tool to each mutant model, and successfully got a short negative test case from each mutant model. The processing time per one mutant model was about 80 seconds. The environment of this experiment was a laptop computer with i7-4650U processor (1.70 GHz, up to 3.30 GHz) and 8 GB RAM. The parameter values of this technique is a=2, τ_0=0.1, l=20, i=50, and ρ=0.03.

The processing time will be acceptable to most of test engineers. However, the performance of this technique depends on the structure of a mutant model, characteristics of an intended failure, and parameter values of this technique. Thus further experiments are expected in future study. Test engineers need to determine the parameter values for this technique by using a trial-and-error method, but it may not be easy. Additionally, the quality of negative test cases depends also on the quality of mutant models, which should be discussed in another study.

5. Conclusion and Future Work

In this paper, we proposed a new technique to generate an effective negative test case from an EPN-based mutant.

The negative test case generation from an EPN-based mutant is a complex problem, since guards and instance variables in an EPN-based mutant cause the feasibility problem and the large state space problem, respectively. Therefore, ACO that is one of meta-heuristic algorithms was introduced into this technique. In this technique, ants heuristically search better paths (that is, shorter negative test cases) to find foods (that is, intended failures) on a field (that is, an EPN-based mutant). We developed a prototype of a tool that supports this technique, and applied it to examples on a trial basis. We found that short

negative test cases could be generated within a reasonable time. However, the performance of this technique depends on several conditions, and further experiments are expected in future study. Also, it may not be easy for test engineers to determine the parameter values for this technique.

There is still room to improve the performance of this technique. For example, we plan to add a pheromone value to each sequence of successive 3-tuples, in order to enable ants to search paths effectively. Subsequently, we will apply the improved technique to non-trivial examples in order to evaluate its effectiveness.

Acknowledgements

This work was supported by JSPS KAKENHI Grant Number 26730038.

References

1. T. Takagi, S. Morimoto and T. Katayama, Development of a Tool for Extended Place/Transition Net-Based Mutation Testing and Its Application Example, *Journal of Robotics, Networking and Artificial Life (JRNAL)*, Vol.4, No.2 (Sept. 2017), pp.168-174.
2. J. Fitzgerald, P.G. Larsen, P. Mukherjee, N. Plat and M. Verhoef, *Validated Designs for Object-Oriented Systems*, (Springer-Verlag London, 2005).
3. T. Takagi, A. Akagi and T. Katayama, Heuristic Test Case Generation Technique Using Extended Place/Transition Nets, *Applied Computing and Information Technology*, *Studies in Computational Intelligence*, Vol.727 (2017), pp.103-115.