Temporal White-Box Testing Using Evolutionary Algorithms

Noura Al Moubayed
Daimler AG
Research and Advanced Engineering
Software Technology, Specification and Test (GR/EST)
Böblingen, Germany
noura.al_moubayed@daimler.com

Andreas Windisch
Technische Universität Berlin
Daimler Center for Automotive IT Innovations
Berlin, Germany
andreas.windisch@dcaiti.com

Abstract

Embedded computer systems should fulfill real-time requirements which need to be checked in order to assure system quality. This paper stands to propose some ideas for testing the temporal behavior of real-time systems. The goal is to achieve white-box temporal testing using evolutionary techniques to detect system failures in reasonable time and little effort.

1. Introduction

Real-time systems raise up many constraints, the most important one is timing. In real-time systems, each functionality must be executed during a specific time interval, otherwise an error will raise due to an encountered system violation. Testing of real-time systems is possible by going through all possible paths and catching any risky functionality which might violate the time constraint. Testing real-time systems is cost-intensive and time-consuming. This paper thus proposes ideas on how to test real-time systems using evolutionary algorithms.

2. Evolutionary Testing (ET)

Evolutionary Testing (ET) automates the test data generation process by transformation into an optimization problem that is solved by applying metaheuristic search techniques such as genetic algorithms [1]. The input domain of the system under test represents the search space in which test data fulfilling the test objectives under consideration are searched for [2]. Every test datum, referred to as individual, will be evaluated using a fitness function which assigns a value expressing the performance of the current individual so that different individuals are comparable. Good individuals which violate the constraints of the system under test or get closer to violate them survive and are used to create the next generation. This process stops as soon as an individual violating these constraints is found or a stopping condition is reached. Evolutionary Structural Testing (EST) is an ET approach for generating test data achieving high structural coverage. Evolutionary temporal behavior testing (ETBT) aims at finding test data that produces particularly long or short execution times when used for executing the system under test [3]. This way, test data are supposed to be found which cause timeliness violations.

3. Evolutionary temporal white-box testing

Evolutionary temporal behavior testing can be supported by considering the internal structure of the system under test and applying the principles of EST in order to direct the search towards reaching worst case execution times (WCET) or best case execution times (BCET) respectively. Therefore the system needs to be instrumented with timestamps, which allow for calculating the execution time of any statement, path, block and sub-structure.

The following subsections propose ideas on how the code can be partitioned and evaluated and how to fitness function can be designed.

3.1. Code partitioning and evaluation

In order to apply temporal behavior testing to embedded systems, their code structure must be partitioned into several code segments, e.g. loops, conditions or statements. Every code segment may contain smaller code segments, such as inner loops or conditions within loops. In case of loops, the body of the loop is considered as a code segment repeated for n times. Subsequently weights are assigned to these segments, which will be used in the calculation of the fitness function.

Program paths can be evaluated according to the blocks and branches it executes, by using the control flow graph that represents all the possible paths in the code of the system under test during execution. Possible evaluations of the code segments are described as follows.

1. Blocks: a block is a sub-structure consisting of one or more nodes of the control flow graph. By applying static code analysis blocks can be classified into dependent or independent. The execution of the independent blocks is independent from the input arguments while the execution of the dependent blocks is dependent on the input arguments. Therefore the execution time of the independent blocks in the system under test does not need to be considered when comparing the execution times of two paths. Reaching WCET will be possible by selecting the paths including the dependent and independent blocks that require long execution times. In contrast, short execution times are favored when searching for BCET. The goal is to get closer to the global WCET and BCET as fast as possible.

2. Branches: A branch is a static path in the code. A branch can contain other branches (e.g. if-else) and blocks. A weight will be assigned to every branch relative to WCET and BCET with a special consideration for independent blocks inside branches, loop boundaries and recursive function stopping conditions. When looking for BCET for example, a branch containing a "jump forward" is assigned a bigger weight than another branch which does not contain the jump, given that the destination of the jump is located near the exit statement.

Static code analysis is a promising approach for partitioning. It is also possible to exclude the execution time of independent blocks from the evaluated fitness value. By doing so, the search space will be reduced and the performance will hence be improved. One example is a loop, whose condition depends on the input whereas its body is independent of it. One path executes the loop n times and another one executes the loop m times, where $n > m$. In this case, there is no need to execute both paths as it is clear that the execution time of the second path is less than the first path.

3.2. Fitness Function

The individuals generated by Evolutionary Structural Testing are evaluated using the fitness function. It assigns a specific value to every test datum to evaluate the input. Smaller fitness values are favored when looking for BCET and bigger values are favored when looking for WCET. To guide the optimization process towards interesting test scenarios, i.e. reaching the longest and shortest execution times faster, the fitness function needs to be changed. The execution time would be the main part of the fitness function in addition to some other factors. Every code segment will be assigned a weight as discussed before. This weight will be considered in the evaluated fitness value for every individual. When looking for WCET for instance, *break* branches have a smaller weight than *continue* branches.

Every individual represents a path. This path must be examined to exclude the code segments and evaluate their weights. In order to determine the additional factors and improve the search performance some strategies must be defined, e.g. avoiding break statements when looking for the longest execution time and try to include them in the execution path when looking for the shortest execution path. The fitness function is minimized or maximized to reach the BCET or WCET respectively.

4. Conclusion

White-box testing methods can be used for temporal testing techniques by providing information about the internal structure of the system under test. This can be done by assigning weights to each code segment depending on execution times and its structure. These weights extend evolutionary structural testing and shape its fitness function in order to detect temporal system failures in less time and effort.

References

[1] Joachim Wegener, André Baresel, and Harmen Sthamer. Suitability of evolutionary algorithms for evolutionary testing. *Annual International Conference on Computer Software and Applications*, 0:287, 2002.

[2] Phil McMinn. Search-based software test data generation: A survey. *Software Testing, Verification and Reliability*, 14(2):105–156, 2004.

[3] Joachim Wegener, Matthias Grochtmann, and Bryan Jones. Testing temporal correctness of real-time systems by means of genetic algorithms. Quality Week 97, 1997.