The University of Bradford Institutional Repository

http://bradscholars.brad.ac.uk

This work is made available online in accordance with publisher policies. Please refer to the repository record for this item and our Policy Document available from the repository home page for further information.

Author(s): Doungsa-ard, C., Dahal, K. P., Hossain, M. A. and Suwannasart, T.
Title: An automatic test data generation from UML state diagram using genetic algorithm.
Publication year: 2007
Publisher: IEEE
Publisher's site: http://www2.computer.org/portal/web/csdl/proceedings/i#I
This material is posted here with permission of the IEEE. Such permission of the IEEE does not in any way imply IEEE endorsement of any of the University of Bradford's products or services. Internal or personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution must be obtained from the IEEE by writing to pubs-permissions@ieee.org.
By choosing to view this document, you agree to all provisions of the copyright laws protecting it.
An Automatic Test Data Generation from UML State Diagram using Genetic Algorithm

Chartchai Doungsa-ard, Keshav Dahal, Alamgir Hossain, and Taratip Suwannasart

Abstract—Software testing is a part of software development process. However, this part is the first one to miss by software developers if there is a limited time to complete the project. Software developers often finish their software construction closed to the delivery time, they usually don’t have enough time to create effective test cases for testing their programs. Creating test cases manually is a huge work for software developers in the rush hours. A tool which automatically generates test cases and test data can help the software developers to create test cases from software designs/models in early stage of the software development (before coding). Heuristic techniques can be applied for creating quality test data. In this paper, a GA-based test data generation technique has been proposed to generate test data from UML state diagram, so that test data can be generated before coding. The paper details the GA implementation to generate sequences of triggers for UML state diagram as test cases. The proposed algorithm has been demonstrated manually for an example of a vending machine.

Index Terms—test data generation, gray-box testing, artificial intelligence, genetic algorithm

I. INTRODUCTION

Software testing is an important activity to assure the quality of software. Unfortunately, software testing is very labor intensive and very expensive. It can take about 50 percents of total cost in software developing process [1]. Automated test data generation reduces an effort of software developers for creating test cases. The software testers may need to spend a longer time using many test cases if the test data used are not of high quality. Therefore, a performance of executing test data is an important issue to reduce the testing time.

Software testing is usually the first part of software development stages, which software developers decide to omit when there is a limited time to deliver the software. In other word, developers may not have enough time after they finished their coding to create test cases to test their code. Generating test cases before coding can resolve these problems. This not only helps developers to test their program when they finish coding but also controls the developers to program the software as defined in the software specification [2]. In this case the software specifications are the main sources for generating test cases as these documents describe the software system to be developed in detail. Software specifications may be UML diagrams, formal language specification, or natural language description.

In this paper, an approach for generating test data from software specification using heuristic technique is proposed. Our approach uses heuristic technique for determining appropriate test data for testing software. Genetic algorithm is selected due to its effectiveness and simplicity. Test data is generated from UML state diagram. In section 2, a review of testing problem and some automated test data generation techniques are presented. A short description of UML state diagram and an automatic test data generation technique are given in section 3. Section 4 illustrates an example of the use of the proposed approach. Finally, section 5 presents conclusion and future work.

II. RELATED WORKS

A. Software Testing

During a testing stage, software testers use test data as input to drive the software program. The testers usually have an expected outcome from these data, which is also test oracle. It may include a return value, a sequence of method calls, or any types of output that the testers want to inspect. After program has been executed, it returns output which is an actual output. Testers have to compare between test oracle and actual result to decide whether a program executes correctly for the given test, and make a verdict of “pass” or “fail”.

If testers want to test functional requirements, they may use black-box testing technique. Black-box testing [3] does not need knowledge of how software is programmed. Test oracles are specified by software design or software specifications. Testers inject test data to execute program, then compare actual result with the specified test oracle. By contrast, white-box testing needs knowledge of how software is programmed. In white-box testing, paths or statements which has been executed are test oracle. These are called coverage criteria. There are three main types of coverage criteria: statement coverage, branch coverage, and path coverage. Statement coverage reports whether each statement is encountered by the test suite or not. Branch coverage reports whether every
branch structure (if – else clause or while clause) has been executed for true and false condition in each branch. Finally, path coverage reports whether all possible paths in function has been tested.

In Object-oriented context, the structure of software is more complicated than the structural one. Conventional test approaches may not be enough for testing. The combination of those two traditional approaches is called Gray-box testing [4]. In Gray-box testing, test data generates based on the high level design which specifies the expected structure and behavior of system. Gray-box testing investigates the coverage criteria of white-box method and finds all possible coverage paths. In addition, the generated test case should be satisfied with functional requirement as in the black-box testing criteria.

B. Test Data Generation on Gray-Box Techniques

Many automated test data generation techniques produce test data based on gray-box method. Not only does gray-box testing concern functional requirement as black box testing, but also concerns on behaviors of system. Clarke [5] proposed an empirical study which compared efforts between automatic test generation and manual test generation. In his report test data was generated from extended finite state model (EFSM). The research shows that the automate test data generation could reduce an effort from manual test data generation for more than 88 percents. Xu and Yang [6] proposed test data generation framework called JMLAutoTest framework. JMLAutoTest framework generates test data from Java Modeling Language(JML) [7, 8]. JML is a notation for specifying behavior and interface in Java class and method. Since JML is a formal specification, developers should spend efforts to understand JML before writing specification.

Because UML diagrams are now widely used for software development [9], generating test data from UML diagrams should help developer to reduce a great number of effort. Wang, et.al [10] proposed test data generation from activity diagram. They extracted a test scenario from activity diagram. The test scenario is a sequence of possible paths in activity diagram. From these paths, the executing sequence of program has been generated in order to cover all possible paths. However, activity diagram describes flows of system, not the behavior of the system.

Due to performance of generating test data and a concern of size of test data set, heuristic techniques are applied for test data generation. GADGET [11] and TGEN [12] use genetic algorithm to improve quality of generating test data. GADGET generates test data from a control flow graph generated from source code. A fitness function is defined for each condition node in control flow graph. An empirical study showed that test data generated by GADGET covers more than 93 percents of source code, while random testing achieves around 55 percents.

TGEN transforms a control flow graph to a control dependency graph (CDG). Each part of CDG represents the smallest set of predicate to traverse every node in control flow graph. Both GADGET and TGEN generate test data using white box method; therefore, test data can be generated only after software is finished.

Using Genetic algorithm to generate test data from software model is proposed in [13]. JML is a model for generating test data. Fitness function is calculated by coverage of paths and post condition defined by JML.

III. A TEST DATA GENERATION FROM UML STATE DIAGRAM USING GENETIC ALGORITHM

A. UML State Diagram

In our approach, we focus on generate test data from UML state diagram [14]. UML state diagram is a graph-like diagram. It describes the system in a state machine. The system has states at a time. States of system are changed due to an event trigger that happens to the system. Trigger and attributes of system specify the transition of current states into new states. Fig 1 is an example of state diagram of a coffee vending machine.

The coffee vending machine starts from “off” state. When user turns on the machine, the status of machine is changed to “on”. If user adds money to the machine, then a money sub state is moved from “Empty” state to “notEmpty” state and attribute “money” is increased. At this moment, money is greater then zero; therefore, if a user requests for coffee, the state of the vending machine in supply part is moved from “Idle” to “busy” to give the user a glass of coffee and attribute “money” is reduced. After finishing coffee preparation, the status of the machine is returned to “Idle”.

From the example, every state contains a state name and transitions. The state name is used for specifying a particular state. Transition defines how status of the system is changed. From the transition “t3:coffee[money > 0]/dec”, t3 is a name of transition. Coffee is an event trigger name, this mean that if the system is “On” an “Idle” state and user makes a coffee event-trigger, state may change form ‘Idle” to “Busy”, but
guard condition should be considered before state changed. Some event triggers may consist of parameters. These parameters are for changing attributes of system. Guard condition is an option for each transition. If it is specified, it is defined in block bracket. Guard condition may be mathematical expressions or sentences that can be evaluated as be true or false. If sentences or statements in guard condition are true, the state of the system is changed to a target state of this transition. The last component of transition is an action. An action is an executable and atomic computation. It may be a statement which changes attribute of the system, an event trigger for the system itself, or other system. In the example, an action for transition “t3” is placed after “/” symbol which is “dec”. “dec” is a event trigger for this system. “dec” may trig a transition “t7”. A number of decisions must be made in order to implement the GA for test data generation. There are problem specific decisions which are concerned with the search space (and thus the chromosome representation) of feasible solutions and the form of the fitness function.

### Table 1: An Example of Tracing UML State Diagram

<table>
<thead>
<tr>
<th>Trigger</th>
<th>State executed</th>
<th>Current State</th>
<th>Attribute Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>Initial state, Off</td>
<td>Off</td>
<td>money = 0</td>
</tr>
<tr>
<td>power-on</td>
<td>t1, On, Initial state: supply, Idle</td>
<td>Idle, Empty</td>
<td>money = 0</td>
</tr>
<tr>
<td>inc</td>
<td>t5, noEmpty</td>
<td>Idle, NoEmpty</td>
<td>money = 1</td>
</tr>
<tr>
<td>inc</td>
<td>t6, noEmpty</td>
<td>Idle, NoEmpty</td>
<td>money = 2</td>
</tr>
<tr>
<td>tm</td>
<td>end</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tm</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

We propose to use a sequence of triggers for UML state diagram as a chromosome. The sequence of triggers is an input for the state diagram. Our approach search for a state and transition coverage occurred by the sequence. The sequence of triggers is extracted one by one for tracing for the coverage. When the first trigger is consider, the trigger is determined whether it can activate any transitions connected to the current status. If the trigger can make a transition from current state, the current state is moved to new state. Then next event trigger is selected to consider again and so on. If no transition can be made, tracing for the state coverage will be stopped and the state and transition coverage are recorded without taking the rest of the sequence to consider. Extracting new event trigger in a sequence after a trigger which cannot make any transition in state may break an order of reaching state. For example, in case of Fig. 1 if a chromosome is shown as a sequence of “power-on, inc, tm, dec”, after “inc” triggers, current state is “Idle”, and “NotEmpty”. From this status “tm” can not make any new coverage state or transitions. If “dec” is considered to unveil new coverage transition which is “t7”, it is look like “power-on, inc, tm, dec” can cover transition “t7”. But “tm” trigger does not involve any transitions which help to reach transition “t7”. Omitting that trigger may make the length of chromosome changed, suggesting to use a GA with variable chromosome length.

The coverage of state and transition is considered as shown in Table 1. A sequence of trigger is power-on - inc - inc - tm - inc - tm, which is generated randomly. The sequence is applied for revealing state and transition coverage as follows. Starting from initial state, there is one transition connected to initial state and there is no event trigger on that transition. Hence, current state has been move to off. Then, the tool extracts the first trigger in the sequence, “power-on” to execute. With event trigger “power-on”, there is a transition from state “off”, which needs “power-on” trigger. That is a
Transition “t1”. Current state is now moved to state “on”. Now there are two sub initial states in state “on”, therefore, current state has been moved to two initial states. These two initial states bring system to new current states “idle” and “empty”. Next event trigger has been taken, “inc”. Transitions from state “idle” and “empty” is checked whether they contain a trigger “inc”. From the diagram, transition “t5” from “empty” contains “inc” trigger, as a consequence, current state is moved from “empty” to “notEmpty”. Moreover, during the transition of “t5”, there is an action declared in diagram as “money = 1”, so an attribute of system has been set as money = 1. “Idle” state is still a current state because “inc” does not make any changes to “idle”. So, current statuses are “notEmpty” and “Idle”. Next event trigger is “inc”. “Inc” drives transition “t6” and changes current status. Transition “t6” is a loop transition, so, the current status from “notEmpty” is still “notEmpty”. Next event trigger is “tm”. Unfortunately, event “tm” does not drive any transitions, which are connected state “notEmpty” and “idle”. The search for coverage is stopped, because no new state is covered.

Fitness function is calculated by a number of states and transitions covered by the chromosome. In addition, states and transitions, which are covered by a pool of chromosome in each generation, are recorded. If a chromosome in a new generation covers new state or transition has not been covered before, there will be an addition score for that chromosome. These values used for analysis in selecting function and evolutionary operation. An algorithm of proposed system is shown in Fig 4.

IV. EXAMPLES

In this section we demonstrate the application of the proposed algorithm manually to generate test data for the state diagram shown in Fig 1. In this experiment, we fix the size of pool to 6. While the crossover probability used is 0.4, the mutation probability is around 0.3. The mutation probability is set high in order to evaluate the technique manually.
Firstly, a list of coverage states and transitions should be created as a coverage repository. For the initial round, the coverage repository is empty. Then chromosomes are generated randomly, and stored in an initial pool. After all sequences are executed, a list of coverage states and transitions is recorded. Fitness value is calculated for each chromosome using the fitness function. Then a selection operator based on the fitness values is used to select chromosomes for crossover and mutation operators. After a new generation of chromosome is generated, UML state diagram is executed again.

In the third round, the generated sequence could not cover any new states or transitions, then a list of coverage states and transition is added to the coverage repository. A new fitness function is generated. The new function gives an extra score for a sequence which covers states or transitions which are not stored in the coverage repository. Then, a new generation of chromosomes is generated.

In this experiment, GAs ran for two generations to cover all states and transitions. Table 2 shows detail of each iteration.

V. CONCLUSION AND FUTURE WORK

An approach for generating test data from UML state diagram using genetic algorithm is proposed. This approach will help software developers to reduce their effort in generating test data before coding. In order to create an effective and robust solution we have demonstrated how genetic algorithm can be applied as a concept.

A proof-of-concept tool development is in progress. The tool contains two parts. The first part is for executing or tracing for coverage states or transitions when an input has been given to an UML state diagram. A search for coverage part is described in section III. The second part of the tool is for calculating fitness values and GA evolution process. We expect this approach to be a useful tool for software developers to help generating test data from UML state diagram to test their software program.

VI. ACKNOWLEDGEMENT

This research has been supported by the EU Asia-link project - TH/Asia Link/004 (91712) - Euro-Asia Collaboration and Networking in Information Engineering System Technology (EAST-WEST).

VII. REFERENCES