Test Case Classification using Category-Partition Finite State Machine

Penpicha Suphapala, Umaporn Leelanuntakul,

Nuchakorn Ngamsaowaros, Peraphon Sophatsathit

Advanced Virtual and Intelligent Computing (AVIC) Center
Department of Mathematics, Faculty of Science, Chulalongkorn University
Email: xuixui5@hotmail.com, amp_le@hotmail.com

ABSTRACT

Testing is an essential activity in software development process. Testers and developers alike are facing a formidable expectation of delivery bug-free software. Certifying bug-free with exhaustive test is commonly known to be impossible. Numerous efforts have been attempted to arrive at a plausible test scenario wherein thorough coverage can be attained. Conventional approaches usually require large amount of test data (or input domain) to generate necessary test cases at premium expenses which, in many cases, end up to be a recalcitrant test process. This paper proposes a straightforward, yet practical algorithmic method to reduce all relevant test cases. The central idea rests upon identifying the relationships among category partition of input specifications and program constraints that subsequently are employed to construct a finite state machine. As such, all paths connecting the start and end states represent the required test cases. Reduction on the number of generated test frames based on the proposed method in comparison with conventional approaches proves to be quite significant.

Keywords: Black-box testing, Category-Partition, Pairwise Testing, Software Component, Test Case Generation

1. INTRODUCTION

Software development, or in a simpler term---programming, often cannot avoid a formidable obstacle, that is, error. One way software developers can uncover those programming errors is through testing, with the help of suitable test cases. As programs grow more complex and size to become programming systems and software, test cases follow suit. Numerous research endeavors have been attempted to devise large scale software testing procedures via statistical testing approach [4] using random inputs. In so doing, a test profile and test size of the test data can be determined from structural functionality or black-box technique of the designated software. The criteria are usually derived from the specification of the corresponding behavioral model.

No matter how hard software developers have tried, such errors seem to be inherent to software. Some forms of test measures have been devised to assess the effectiveness of the selected test procedure so that the software product under test meets the required quality prior to acceptance. As a consequence, the need to create full test coverage based on dependability theory [3], test data selection from large data pool, and automatic test case generation calls for various attributes, stopping rules, and test indicator selection guidelines. These guidelines will be applied to adequately generate test statements, paths, and branch coverage [6]. In practice, testing often carries out under tight schedule and financial constraints. Hence, careful planning and selection of test cases are prerequisite to arrive at minimal test cases, yet effective enough to detect many errors.

This paper proposes an approach to reduce the number of test cases using Category-Partition Method, working in conjunction with Pairwise Method. The paper encompasses the followings. Section 2 describes test case generation process using Category-Partition Method. The governing test frame is established by means of Pairwise Method in Section 3. The resulting test frames are then used to create test cases which are demonstrated by a concise yet thorough example in Section 4. An inference on the applicability of the proposed test case generation algorithm for software testing and future enhancement are given in the conclusion.

2. TEST CASE GENERATION PROCEDURES

Early black-box testing employed random test data generation technique to create test cases. This approach required enormous test data to guarantee complete coverage of all possible combinations. The use of Pairwise Method [7] helped reduce the number of test cases required, yet at the expense of additional parameter independence stipulation.

Partitioning test data to generate appropriate test cases is an alternative approach that helps reduce the number of test cases generated from representative test

data. This paper proposes an effective test case generation algorithm based on the above idea by using Pairwise Method and Category Partition Method [1, 2]. The proposed algorithm utilizes program specifications to classify relevant relationships among input test data, whereby adequate coverage can be attained. As such, the number of test cases obtaining from representative input data partitions or categories is reduced, yet equally test coverage in comparison with other methods is still maintained owing to non-redundant data selection.

Data relationship analysis is imperative to classify input data category for complete test coverage without sacrificing extraneous verification and speed. The basic idea rests on the scope of potential input data, in conjunction with environmental conditions that are pertinent to the test process. The analysis dichotomizes input data and their corresponding environment domains into partitions or classes called "parameter." Each parameter is further divided into "choice" according to program execution conditions. The relationship is then formed based on the attributes of each choice and its access conditions to construct a finite state machine (hereafter referred to as FSM or state machine). The state machine will be subsequently used to determine possible paths emanating from start state to final state. Each path represents a test frame of the test data necessary for test case generation process described below.

Step A: Analysis of test program execution stipulations

Analyze the scope of input data and their possible environment conditions necessary for test program execution. Classify the data into separate groups called "parameters." Each parameter is further split into choices according to the test program conditions. Parameter grouping is carried out as follows:

- input data scope

-determine the associated attributes of program's parameters

-determine the required parameter specifications from the test program

- environmental conditions

-determine various relevant conditions of the parameters

-determine the characteristic of parameters that effect program execution

Consolidate all possible parameters and designate each choice corresponding to the parameter's characteristics.

Step B: Establishment of relationship of choices

-Determine the relationships among the parameters obtained from step A by assigning basic attributes to the choices of each environmental parameter group, as well as viable access conditions that reflect the relationship among those choices. Determine the choice(s) that causes program defects (hereafter denoted by [error]). Annotate the causes of defect for the choice and arrange them by priority of individual choice.

Step C: State machine construction from each choice

-Construct a state machine from the above sets of relationships beginning with the first data node as the start state and the leaf data node as the final state. The remaining data nodes represent possible states. Accessibility of each choice designates the relationship among the choices. State change denotes possible paths among categories. The first state points to data nodes of the choice in the first category, whereas all leaf nodes point to the final state.

-Determine the direction from one node to the next according to selection conditions for each choice. The attributes corresponding to the selected node are then forwarded to the next (or destination) node. These attributes are compared with those of the destination node to see if any conflicts exist. If it is the case, the destination node is partitioned in accordance with the number of conflicting cases.

-Apply conditioned direction from one node to its successors for state transition.

Step D: Derivation of test frame construction from test paths

-Determine test paths using Pairwise technique by considering the parameters between participating categories one pair at a time. The path so obtained will be unique and independent.

-Denote each test path as the characteristic of input domain for test case generation.

Step E: Test case generation procedures

Create one test case based on every test frame derived above since each test frame designates the conditions required to construct the necessary test cases. A test case consists of the following components:

-test ID

-description which narrates the test procedures and input description for each test condition, including the preconditions of the test case

-expected output which exhibits the results to be obtained from each test case

-actual output which is recorded after the test is run. If a test passes, the actual output will indicate “Pass.” If a test fails, it is helpful to record “Fail” and a description of the failure.

3. SAMPLE TEST FRAME CONSTRUCTION

The experiment demonstrates the procedures on generating test cases for a bookstore search utility. The program searches for books with less than or equal to the given price. The output displays all titles and prices from search results. Some specific program characteristics are given below.

Command: search
Syntax: search <title> <price>

Function:

search command looks for book titles based on the input keyword of which their respective price is less than or equal to the given price.

<title> denotes the keyword used as search criteria, where <title> must begin and end with double-quotes ("). If the <title> contains a double-quote("), simply denote it by two consecutive double-quotes (""").

<price> represents the maximum price to be searched for, all of which must be greater than zero. If this search item is omitted, all books matching <title> keyword will be retrieved, regardless of their price.

Example:

-search "computer" 1000

display all book titles having computer as their subject, along with selling price less than or equal to 1000

-search "computer network" 1000

display all book titles having computer network as their subject, along with selling price less than or equal to 1000

-search "computer "" network"

display all book titles having computer " network computer as their subject, along with their selling price

Step A: analysis of various relevant stipulations for execution of the search program

Based on the scope of input data and working environment of the program, the first task is to analyze the impact on program execution by considering the syntax and function specifications. The above book title search example encompasses two parameters, namely, <title> and <price> which can be further examined their syntax and function as follows:

Syntax parameter

-the format of <title> open/close double-quotes

-the length of each keyword used in search process

-existence of embedded double-quotes in the keyword

-existence of embedded blanks in the keyword

-the value of <price>

The output environment of the function parameter consists of

-title search display

At which point, analysis of all possible parameters' specific characteristics constitutes the following individual parameters' choices

Parameters of input domains

<title> pattern:

choice1<A1>: correctly quoted
choice2<A2>: improperly quoted
choice3<A3>: not quoted

<title> size:

choice1<B1>: non-empty
choice2<B2>: empty

embedded quotes:

choice1<C1>: no embedded quote
choice2<C2>: one embedded quote
choice3<C3>: several embedded quotes

embedded blanks:

choice1<D1>: no embedded blank
choice2<D2>: one embedded blank
choice3<D3>: several embedded blanks

boundary <price>:

choice1<E1>: more than or equal to zero
choice2<E2>: omitted
choice3<E3>: less than zero

Parameters of environment

display <title> and <price> search results

choice1<F1>: exactly one
choice2<F2>: more than one
choice3<F3>: none

Step B: establish the relationships among the choices

In order to prevent impossible combinations, some preliminary specifications of the choices, access criteria, and error notes of each choice are described for all parameters. These relationships are then arranged, including their respective choice within the relationship as follows:

Parameters of input domains

<title> pattern:

choice1<A1>: correctly quoted [attribute Quote]
choice2<A2>: improperly quoted [error]
choice3<A3>: not quoted [error]

<title> size:

choice1<B1>: non-empty [if Quote] [attribute NonEmpty]
choice2<B2>: empty [if Quote] [attribute Empty]

embedded quotes:

choice1<C1>: no embedded quote
choice2<C2>: one embedded quote [if NonEmpty]
choice3<C3>: several embedded quotes [if NonEmpty]

embedded blanks:

choice1<D1>: no embedded blank
choice2<D2>: one embedded blank [if NonEmpty]
choice3<D3>: several embedded blanks [if NonEmpty]

boundary <price>:

choice1<E1>: more than or equal to zero
choice2<E2>: omitted
choice3<E3>: less than zero [error]

Parameters of environment

display <title> and <price> search results
choice1<F1>: exactly one
choice2<F2>: more than one
choice3<F3>: none

It is apparent that the choices with no associating specification for access conditions are those which can be accessed by any attributes of the given specification.

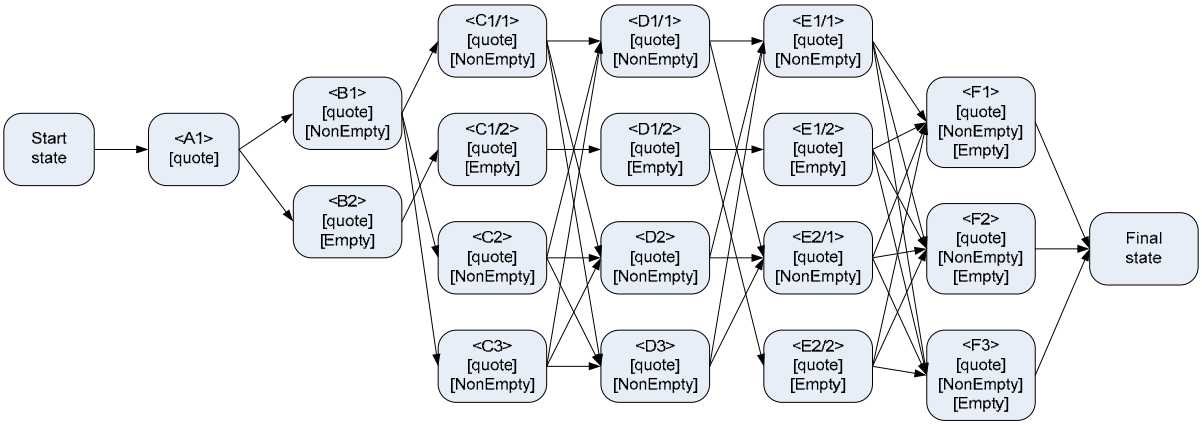

Fig.1: *State machine of total Test Frames*

Step C: construct state machine from the choices' relationship

Given the first node as start node and the leaf nodes as end nodes, construct new nodes to represent each choice of the parameters along with their attributes. Connect a node to other nodes based on access conditions to form a path. The resultant state machine is depicted in Figure 1.

Step D: derive test frames from test paths of the state machine

Every path of the state machine denotes the specification of input test data. If a path is visited in the state machine, that path is considered tested and will not be revisited. Bearing this principle in mind, the total number of paths created from step C within the test frame is 15 as follows:

1. {A1, B1, C1/1, D1/1, E1/1, F1}
2. {A1, B2, C1/2, D1/2, E1/2, F1}
3. {A1, B1, C2, D1/1, E2/1, F1}
4. {A1, B1, C3, D1/1, E1/1, F2}
5. {A1, B1, C1/1, D2, E1/1, F3}
6. {A1, B1, C1/1, D3, E1/1, F2}
7. {A1, B1, C2, D2, E2/1, F2}
8. {A1, B1, C2, D3, E2/1, F3}
9. {A1, B1, C3, D2, E2/1, F3}
10. {A1, B1, C3, D3, E1/1, F1}
11. {A1, B2, C1/2, D1/2, E2/2, F1}
12. {A1, B2, C1/2, D1/2, E1/2, F2}
13. {A1, B2, C1/2, D1/2, E1/2, F3}
14. {A1, B2, C1/2, D1/2, E2/2, F2}
15. {A1, B2, C1/2, D1/2, E2/2, F3}

Step E: Test case generation procedures

According to the test frames from previous step, each test frame can generate one test case. A sample test frame from #6, i.e., {A1, B1, C1/1, D3, E1/1, F2} is given below.

<title> pattern: correctly quoted
<title> size: non-empty
Embedded quotes: no embedded quote
Embedded blank: several embedded blanks
Boundary <price>: more than or equals to zero
Environment Display <title> and <price> search results: more than one

And the corresponding sample test case is as follows:

Test ID: 6
Description: the database may contain more than one book records having“life and love” as the title and price over 200.
<title>: “ life and love ”
<price>: 200
Expected results: Database search may result in more than one match, wherein the corresponding title will be displayed according to the above conditions.
Actual results: Pass / Fail

Based on all combinations obtained from the aforementioned search utility, there are 1*2*3*3*2*3 or 108 cases of input data to be tested. Using Category-Partition Method, on the contrary, and assigning appropriate attributes to the parameters and environment variables, the yield of the state machine so constructed to derive all possible input test cases is 15, which is an astounding 86.11% reduction.

4. EXPERIMENTAL RESULTS

This study conducted 3 test programs to assess the validity of the proposed approach. The first program was a book title keyword search utility that output a list of titles and its corresponding price. The second program was a boat reservation utility. The last program was a time-table search for any given train departing from Hua Lum Phong Railway Station to destination.

The experiment was carried out using the four techniques, excluding [error] condition when last test case generation was performed. Comparative statistics with all-combination approach are given in Table 1.

Table 1: *number of test case in each method*

Method	Pair wise	Base case	Category partition	FSM	Combi-nation
Test 1	16	9	30	15	108
Test 2	14	9	22	16	108
Test 3	9	5	7	3	9

Table 2 compares the percentage of test case reduction obtained from all four techniques with all-combination approach. A visual comparative graph is illustrated in Figure 2.

Table 2: *Percent reduction as compared with all-combination techniques*

Method	Pair Wise (%)	Base case (%)	Category partition (%)	FSM (%)	Combi-nation (%)
Test 1	85.19	91.67	72.22	86.11	0
Test 2	87.04	91.67	79.63	85.19	0
Test 3	0	44.44	22.22	66.67	0

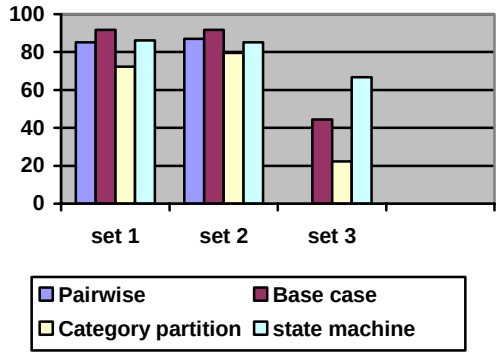

Fig.2: *Graph of reduction rate of representative test cases*

Table 3 depicts the reduction rate of number of test cases in comparison with all-combination technique.

Table 3: *the reduction rate of number of test cases*

Method	Pair Wise (%)	Base case (%)	Category partition (%)	FSM (%)	Combi-nation (%)
Percent	57.41	75.93	58.02	79.32	57.41

5. CONCLUSION AND FUTURE WORK

Test case classification using Category-Partition state machine to separate input test data based on their specifications yields competitive results in comparison with existing techniques such as Base Case, Category-Partition Method, Pairwise, and all-combination. Base Case approach is suitable for single normal case program, leaving multiple normal cases open. This was the case as in the first and second test programs where some test coverage were missing, despite apparent low number of test cases. The proposed approach, albeit slightly higher yield, furnishes more coverage.

The benefits from the proposed approach are three folds. First and foremost, test coverage is as complete as Category-Partition since the same classification technique is employed. The use of Pairwise method to create test frame eliminates duplicate cases, whereby fewer test cases are resulted. Next, Pairwise and all-combination techniques eliminate all examined cases, disregarding their inter-relationship. As such, certain needed test cases would not be taken into account, while others are contrived cases. The proposed approach incorporates input data relationship into consideration, whereby thorough inter-relationship coverage is attained. Lastly, the proposed technique is adaptable to matrix form, hence highly suitable for machine learning applications.

Due to the small size of data set, test path analysis was straightforward without any singularity. What remains to be considered is how to apply the proposed technique to large software with overlapping input domains. The positive precipitation from this approach is the use of state machine to construct test cases as long as those complex inputs can be categorically classified (there are myriad of researches on data clustering/classification). A concern from the theoretical stand point is to prove data adequacy [5] that represents the smallest set of input domain. This is to ensure a stopping criterion in deriving a state machine for all test cases.

Such incorporation of predicate and measurement theory calls for white-box analysis technique to aid the state machine construction. The processing time and cost incurred by white-box analysis is unfavorably justified from business point of view, and prolonging the software product to market. Thus, tradeoffs between a compelling challenge for in-depth study and development are to reach the ultimatum of "Software Engineering" philosophy.

6. REFERENCES

[1] P. Ammnann and J. Offutt. “Using Formal Methods To Derive Test Frames In Category-Partition Testing”, *Proceedings of the Ninth Annual Conference on Computer Assurance, 1994--COMPASS '94', Safety, Reliability, Fault Tolerance, Concurrency and Real Time, Security*, June 27-July 1, 1994, pp. 69-79.

[2] Thomas J. Ostrand and Marc J. Balcer. “The Category-Partition Method for Specifying and Generating Functional Tests”, *Communications of the ACM*, Volume 31, Issue 6, June 1988, pp. 676-686.

[3] D. Hamlet. “Foundations of Software Testing: Dependability Theory”, ACM SIGSOFT Software Engineering Notes, *Proceedings of the 2nd ACM SIGSOFT Symposium on Foundations of Software Engineering SIGSOFT '94, December 1994*, Volume 19 Issue 5, pp. 128-139.

[4] P. Thevenod-Fosse and H. Waeselynck. “STATEMATE Applied to Statistical Software Testing”, ACM SIGSOFT Software Engineering Notes, *Proceedings of the 1993 ACM SIGSOFT International Symposium on Software Testing and Analysis ISSTA '93*, Volume 18 Issue 3, pp. 99-109.

[5] E.J. Weyuker. “The Evaluation of Program-based Software Test Data Adequacy Criteria”, *Communications of the ACM, June 1988*, Vol. 31, No. 6, pp. 668-675.

[6] H. Zhu, P.A.V. Hall, and J.H.R. May. “Software Unit Test Coverage and Adequacy” *ACM Computing Surveys*, Vol. 29, No. 4, December 1997, pp. 366-427.

[7] Kuo-Chung Tai and Yu Lei, “A Test Generation Strategy for Pairwise Testing”, *IEEE Transactions on Software Engineering*, Vol. 28, No. 1, January 2002, pp. 1-2.