Novel Fuzzy Approach for Ranking Test Vectors

Gergely Takács*
*Óbuda University, Doctoral School of Applied Informatics, Bécsi út 96/b, 1034 Budapest, Hungary,
Email: takacs.gergely@phd.uni-obuda.hu

***Abstract*—Software testing is essential part of the software development especially when the software has safety critical classification. In this paper authors introduce a method for select the right values and ranges for unit and functional testing. The key feature of this novel approach is that, test vectors can be ranked before selection in order to gain test time or quality. Fuzzy set theory is used for the ranking of the test sets. The introduced method presented on a example code, which helps the understanding of its basic principals. The authors take notes on the implementation of the test vector generation. Directions of further research are also discussed.**

I. Introduction

Challenge of the software testing is growing. Program code of the applications contains more and more code and the complexity is increased as well. The counterpart of the program is the test. Basic code level non-intrusive tests like unit tests are based on a simple principal; sets the inputs then checks the desired outputs. The generic goal could be that every instruction to be executed during a test procedure, however in practice it is not possible due to the high number of input combinations.

The authors give a short overview on the software testing in the first sections. The problem of the test vector selection is detailed to give a view why this field is require intensive research. The proposed method is illustrated with examples.

II. Software Testing

Testing is a wide terminology, first of all some clarification is given here. Generally test could be a verification or a validation. The verification checks the software is in match with the specification. Validation is the check of the software is implemented correctly. Test approaches are differentiated from the tester view point. If the tester has no access to the internal structure of the code then the method is called black box testing. If the tester knows the internal code structure then it is called white box testing. Gray box testing is also know. Construction of the test itself the internal structure of the code is not used, but it is used for the evaluation of the test results.

The code coverage measurement is a widely accepted technique for evaluation. Usually the 100% coverage [11] test coverage is required in case of safety critical application[7]. Software is safety critical if the consequence of the faulty operation is loss of life or causes major injuries. Safety level is describes impact of the consequence. In the automotive and the aerospace industry the software development process is defined according the safety standards like ISO26262 and DO-178-B. These standards require the Modified Condition/Decision Coverage - MC/DC[6] in case of high safety levels. Test case must satisfies at least one of the followings:

- Each decision tries every possible outcome
- Each condition in a decision takes on every possible outcome
- Each entry and exit point is invoked
- Each condition in a decision is shown to independently affect the outcome of the decision

Unit testing

Unit testing is a test methodology, which tests the code on the lowest non intrusive level and targeting the code correctness only.[8] The aim is that test the smallest callable part of the program. Unit test guaranties the correctness the function and during the regression. Practically this means the tests cases defines function calls and their expected return values. According to the mentioned safety standards a test should check all conditions and their all possible outcome have been executed. A unit test is usually some of function calls of the target function and comparison of the return value.

In case of deep function with high cyclomatic complexity test vectors of the unit test can be numerous and execution time of the test is crucial. State machine testing requires multiple sequences of test vectors. This implies that new methods needed to extract the values form code and create the test vectors automatically based on static analysis.

Functional testing

Functional test executed on a higher level then unit test. A module consists of multiple units while realizes a functionality. Functional test checks the functionality of the module. Testers write these tests based on specification and requirements. Functional test usually are black box tests. The same coverage requirements are demanded on the functional tests as mentioned before. Coverage report shows the program code and the test are match with the specification. For example if there is a code segment which is never executed that it may not needed or the test or the specification is incomplete.

Test may performs 100% coverage while the test has no strong connection with the code. Let us bring an example here: the tested function may consider the input variable is bigger or smaller the zero. This function could be tested with $\pm$ one or $\pm$ maximal value pairs. The first case focuses on the decision while the second test case uses two possible test values. Both of the cases are valid, however it is a big difference between them. Hard to measure the quality of the test.

III. Detecting Branch Points with AST Analyzis

In order to reach the highest test coverage the input source code shall be analyzed and branch points have to be found. This should be done without executing the code itself, in order to get a platform independent method. A static method shall be used. Some research points out that the dynamic analysis during the test itself may be able to increase the effectiveness, in this article it is not used only considered as a further improvement [9].

Abstract syntax tree (AST) analysis is one of the most accepted method to walk through the code and getting information its future execution [4]. The goal is to detect branch points where a variable conditions are checked. These points are the key in the test vector generation and also will be used for ranking of the test vectors. The AST is used for many purpose in software engineering:

- Code optimization
- Static code analysis
- Code statistics
- Input for other analysis

In case of the last there points the execution of the following compilation steps is not necessary, the goal here is to prepare some information which can be used for further analysis. These results can be used as statistics, as well as an input for other analyses.

IV. Test Vector Ranking

In this section a complete method is introduced for selecting and ranking the test vector sets based on fuzzy theory and give example for its usage through testing a basic C function. For the sake of simplicity, in this example we concentrate only on unit testing. In this chapter we consider only one input parameter and only constant operand for testing.

In a homogeneous range only boundary values are emphasized and it makes hard to select any other values for the test vectors. Of course the value relevancy is not the same in the whole set.

Current techniques define the variable and decision points are as a flat range. The proposed approach is based on fuzzy set theory. Instead of the classical methods, the branch points are represented with a triangular fuzzy number. The logical operation between the conditions could be modeled with fuzzy operators.

Definition 1:

$$F_{gt_{c,s}}(x) = \begin{cases} 0 & \text{if } x < c - s \\ (x - (c - s))/s & \text{if } c - s < x < c \\ ((c + s) - x)/s & \text{if } c < x < c + s \end{cases}$$

triangular fuzzy set represents the input value x relevancy at comparison greater than with a constant value $(x > c)\ where$

1) x is the input value of variable
2) c is the constant value the variable is compared with
3) s is the parameter a fuzzy set support can be set

From this point of view the greater than and the less than comparison are the same and having the same functions.

One input variable can be a basis of branch decision more than one (n) times in a function or a logical block. In this case n fuzzy set aggregate shall be calculated. Aggregating two sets with the Łukasiewicz strong disjunction operator produced a good results. Simple Min/Max$\{$a,b$\}$ were not given sufficient results.

Traditional understanding of deffuzzyfication is that reforming a fuzzy set to a crisp value. In this case the expected result is a set or range of values what the test can be used as input vectors. Current state of the research the alpha cut was used for define the range of $p1$. Change of the cut level could be used for tune the result of the method. This parameter has effect on the precision, coverage and execution time of the test. The highest alpha cut selects the most relevant values for the unit test. If required to generate more test cases then it is allowed to use lower alpha cuts.

Example

In this section test vector ranking method is described which relies on the previously introduced technique. The following example is a simple C function with one input parameter. The analysis aims to find the input values which are the decisive points at the branches. The analysis steps will be the following:

1) Build an abstract tree and find the values the input operands are tested with
2) Create fuzzy set for each found point
3) Aggregate the given fuzzy sets to have result fuzzy set
4) Defuzzyficate the result to get the set of test vectors

See the example code C code 1

```c
int func_01_simple(int p1) {
    if ( p1 > 2 ) {
        if ( p1 < 4 ) {
            return 1;
        }
        else {
            return 2;
        }
    }
    return 3;
}
```

Listing 1. Example test target function

In this simple case the solution is obvious, the branch values of the variable $p1$ are 2 and 4. The minimal set to reach the maximal branch coverage is $\{$2,3,4$\}$.

Step 1. Build AST: The simplified abstract syntax tree is shown on figure 1.

As result there is two condition based on the value of $p1$; the constant 2 and the constant 4 are marked with gray leaves in the figure. This algorithm is implemented in a python script based on pycparser library[5].

Step 2. and 3. Create fuzzy sets and their aggregate: Core of the triangular fuzzy sets are 2 and 4. The support size should be more than one but can depend on many aspects like severity of the variable, deepness of the branch point etc.. In this example the support is 2.4.

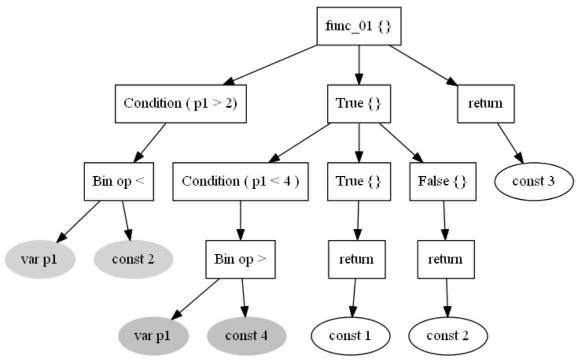

Fig. 1. Abstract syntax tree of func_01_simple()

Branch point in line 2: $F_{gt}(p1, 2, 1.2)$
Branch point in line 3: $F_{lt}(p1, 4, 1.2)$

Using Łukasiewicz strong disjunction to aggregate of the sets gives one fuzzy set for one variable.

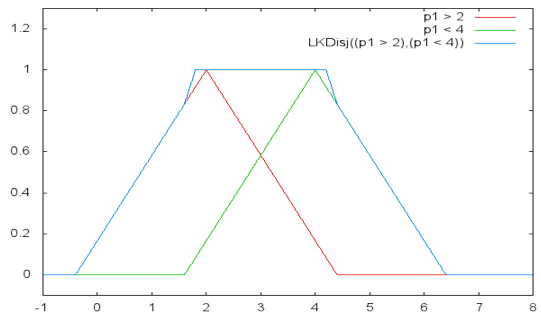

Fig. 2. Łukasiewicz disjunction $p1(2)$, $p1(4)$

4. Defuzzyficate the result to test vector set: To show the scalability, in this example different alpha cuts are shown in the figure 3 : $\alpha = 0.9$, $\alpha = 0.5$, $\alpha = 0.1$.

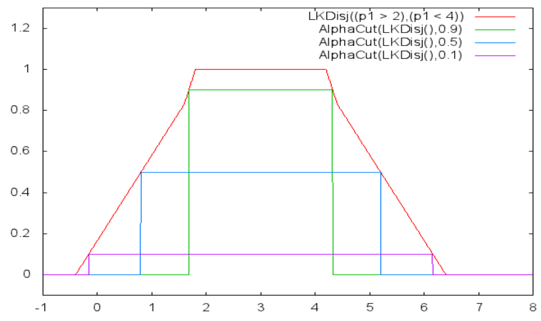

Fig. 3. Alpha cuts of the aggregated $p1$

It is also a quality specific decision how to handle of the fractions on fix point domain, in this example we chose to rounding to integer values.
The results are:

$\alpha = 0.9 \Rightarrow \{2, 3, 4\}$
$\alpha = 0.5 \Rightarrow \{1, 2, 3, 4, 5\}$
$\alpha = 0.1 \Rightarrow \{0, 1, 2, 3, 4, 5, 6\}$

In this case the most strict $\alpha >= 0.9$ set is enough to reach the 100% branch coverage, so the final solution is:

```
void test_func_01_simple() {
    TEST_EQ(func_01_simple(2),3);
    TEST_EQ(func_01_simple(3),1);
    TEST_EQ(func_01_simple(4),2);
}
```

Listing 2. UnitTest of func_01_simple()

V. Handling More than One Input Variable

It has been shown how this method works with one input variable. The dimension of the problem is increasing with the number of the variables. The relevancy of test vectors can be shown only in three dimensions in case of two variables in a decision.

```
int func_02(int p1,int p2) {
    if ( (p1 + p2 ) > 1) {
            return 1;
            }
    return 2;
}
```

Listing 3. Example for function with 2 variables

Code 3 shows an example for this case. The combination of the P1 and the P2 defines a plain. This area gives 2^{64} value pairs in case of integer values in a 4 byte representation. About the half of the pairs are negative and the other half is the positive samples. To test all of the value pairs is practically impossible.The figure 4 show the possible value pairs.

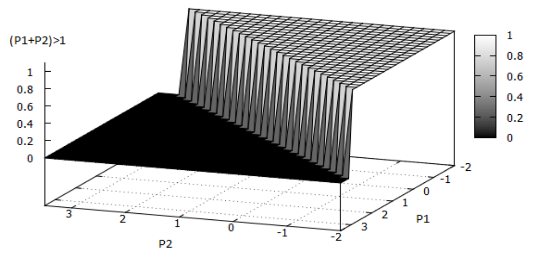

Fig. 4. The possible value pairs space

In order to reach a 100% branch coverage only two vectors shall be chosen, one true, and one false. Selection of two value pairs is simple until there is no expectation on the values. The value pair distance be close to the decision point could be an expectation. It is possible that not all expectations can be satisfied with one value pair.

The shown method solves this problem in the following way. Consider P1 and P2 as a fuzzy set in its own domain. The + operator is defined for fuzzy numbers. Let Ps is a fuzzy number defined by $F_{add}(P1, P2)$ in the joint domain. Ps represent the decision in a fuzzy way in tree dimensional space. The figure 5 shows it.

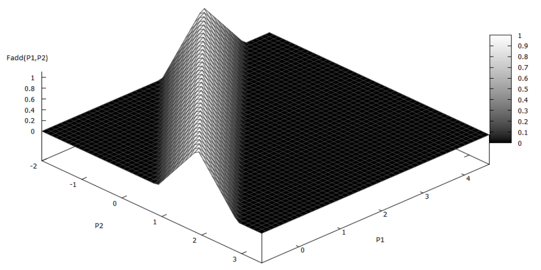

Fig. 5. $F_{add}(P1, P2)$ function

The number of the defined test case pairs in Ps is still numerous. The next step of the method is to decrease the number of the accepted value pairs. The expectation on the value pairs holds the required information for the filtering the values. Let F_{exp} be the fuzzy operator of a given expectation. In this example we chose to expect for the mid values of the range. The figure 6 shows the result of the F_{exp} on the P_s fuzzy number.

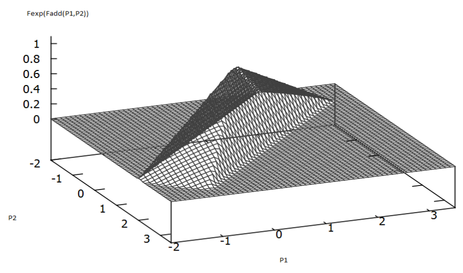

Fig. 6. $F_{ext}(F_{add}(P1, P2))$ function

It is easy to see the number of the possible test cases are highly decreased. The last step is to get the alpha cuts.

VI. Further Works

In the current state of the research the method works with triangular fuzzy numbers. The problem of this class is that class of the operator is not closed. This means the result of the operation is may out of the class. Computation with a non uniform number is complex. We are going to switch to a closed operator family. This could significantly increase the computation speed and decrease the calculations complexity. The size of the core and the support is currently undefined. Different fuzzy class has different parameters. We will analyze the impact on the result of the change of the fuzzy class. The method opens a new way to introduce additional expectations on the test vectors.

VII. Conclusion

Authors give a method for rank test vectors. The method could be used in an automatic test vector generator or it could be used to analyze a existing test. The introduced method able to handle additional expectation on the test vectors. The introduce method could increase the generated test vectors also.

VIII. ACKNOWLEDGMENT

The author(s) gratefully acknowledge the grant provided by the project TÁMOP-4.2.2/B-10/1-2010-0020, Support of the scientific training, workshops, and establish talent management system at the buda University.

References

[1] L.A. Zadeh, Fuzzy logic, neural networks, and soft computing, Communications of the ACM, vol. 37, No. 3, pp. 77-84, 1994.
[2] The C Programming Language - Brian Kernighan and Dennis Ritchie, Prentice Hall, Inc., 1988. ISBN 0-13-110362-8
[3] ISO/IEC 9899 - Programming languages - C
[4] Compilers : principles, techniques, and tools - Alfred V. Aho, Monica S. Lam, Ravi Sethi, Jeffrey D. Ullman, Addison-Wesley, 1986. ISBN 0-321-48681-1
[5] Pycparser - http://code.google.com/p/pycparser/
[6] Certification Authorities Software Team (CAST) - What is a Decision in Application of Modified Condition/Decision Coverage (MC/DC) and Decision Coverage (DC)? - Completed June 2002 - http://www.faa.gov/aircraft/air_cert/design_approvals/air_software/ cast/-cast_papers/media/cast-10.pdf
[7] The art of the software testing - Glenford J. Myers, John Wiley & Sons, Inc., 2011. ISBN 1-118-03196-4
[8] xUnit Test Patterns: Refactoring Test Code - Gerard Meszaros, Pearson Education, Inc., 2009 ISBN 0-13-149505-4
[9] Simons, A.J.H., Wenwen Zhao - Dynamic Analysis of Algebraic Structure to Optimize Test Generation and Test Case Selection - Testing: Academic and Industrial Conference - Practice and Research Techniques, 2009 - 978-0-7695-3820-4
[10] Toeppe, S., Ranville, S., Bostic, D., "Automating Software Specification, Design and Synthesis for Computer Aided Control System Design Tools", 2000, Proceedings of the 19th AIAA/IEEE/SAE Digital Avionics System Conf.
[11] Chilenski, J. J., Miller, S., Applicability of modified condition/decision coverage to software testing, 1994,