法政大学学術機関リポジトリ
HOSEI UNIVERSITY REPOSITORY

PDF issue: 2024-09-03

Supporting Tool for Automatic Specification-Based Test Result Analysis

LI, Ji

(出版者 / Publisher)
法政大学大学院情報科学研究科
(雑誌名 / Journal or Publication Title)
法政大学大学院紀要. 情報科学研究科編 / 法政大学大学院紀要. 情報科学研究科編
(巻 / Volume)
8
(開始ページ / Start Page)
23
(終了ページ / End Page)
28
(発行年 / Year)
2013-03
(URL)
https://doi.org/10.15002/00009518

Supporting Tool for Automatic Specification-Based Test Result Analysis

Li Ji
Graduate School of Computer and Information sciences
Hosei University, Tokyo,184-8584, Japan
Email: ljsoar02138@gmail.com

Abstract—**Automatic Specification-Based test result analysis tool support is crucial for applying specification-based test in practice .It fills in the blank of lacking in supporting tool for formal specification test. In this paper, we present a supporting tool, which enables us to detect errors from various predicate expressions. The target objective of the testing supported by our tool is the formal specification written in the Structured Object-oriented Formal Language (SOFL).The experiment result indicates that the tool can handle most of types in specification, besides some rarely used types.**

Keywords: specification-based , result analysis, tool

I. INTRODUCTION

In software engineering field, there exist several major challenges. For instance, engineers have to record what they plan to build, their solutions to potential problems, the credibility of the solution, etc. Since formal methods have made significant contributions to software engineering by offering formal specification, refinement, and verification techniques for achieving correct programs. They are, however, "too good to be true" for most industrial software development projects. Formal methods can theoretically ensure that a program is correct with respect to a formal specification, but have no means to guarantee that the specification accurately and completely reflects the user's requirements. Therefore formal methods have been regarded as an effective way to solve these challenges. Formal methods emphasize the use of mathematical notation in writing specifications, both functional and non-functional, and the employment of formal proofs based on logical calculus for verifying designs and programs. It can significantly save development cost and time while allowing the avoidance of many human errors during a testing process[7]. However, for such a long time, there is always a lack of supporting tool for automatic specification-based test. So it is hard to apply theory into reality. For this reason, it is necessary to develop a practical tool which can test the specification automatically.

The Automatic specification-based test result analysis is an effective technique in software industry. We generate test case automatically based on specification and assign the test case to both specification and program, program is refined from the . specification. Then we use the test oracle to test if we have found a bug or not.

The goal of our research is to take the component-based software engineering approach to produce a package in C# to support automatic test result analysis from various predicate expressions. The package includes several classes, each class contains necessary methods for evaluating various kinds of predicate expression.

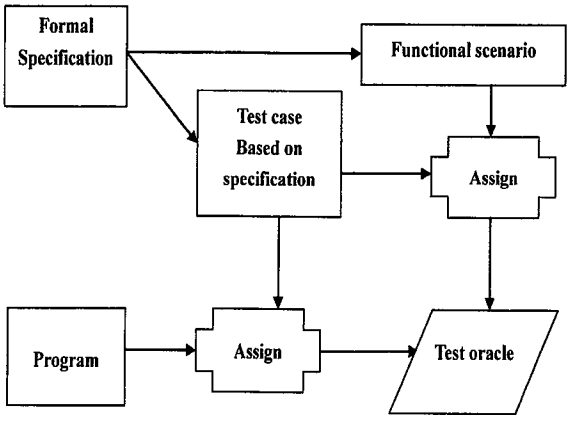

Fig. I. Principle of Specification-Based Test

In figure 1, formal specification is written in SOFL, SOFL stands for Structured Objected-Oriented Formal Language[6]. It is both a language and a method for constructing functional specification and designs for software systems. The essential structures of a SOFL specification are modules and Condition Data Flow Diagrams. A CDFD is a DFD with an operational semantics, is a directed graph composed of processes describing functional operations, data flows for data communications among processes and data stores. In this paper, we regarded it as specification language. Figure2 is an illustration of process specification. It shows the structure of a process specification. Below is an example of one type of specific structure of the package:

Package structure

Package automatic_test_evaluation;

Class Evaluation From Relations;

Supervisor: Prof. Shaoying Liu

Method Boolean Evaluation From Numeric Exp (operator, Exp1, Exp2, Test case){...}

/* operator is a member of{>,<,=,>=,<=,<>} */

The remainder of the paper is organized as follows .Section 2 describes strategies for automatically testing specification, including different kind of specification's recognition and test case assignment. In section 3, we present a tool that deals with specification and result display. Section 4 introduces related work, and finally, we summarize the paper and point out the challenges and future research in Section 5.In Chapter II, the prediction system is proposed. In chapter III, after the determination of the parameters to optimize the system, the proposed prediction system is evaluated by using real wine-sales data. In Chapter IV, we concludes the result and states the future study.

II. AUTOMATIC SPECIFICATION-BASED TEST RESULT ANALYSIS STRATEGY

Automatic specification-based test result analysis is the last step in software testing. It has 2 goals at different levels: ideal goal and practical goal[3]. Ideal goal is to find all of the bugs in the program which is extremely difficult as we know. Practical goal is to meet required coverage criteria, such as statement coverage or path. Compared to the ideal goal, the practical goal is relatively easier to achieve but there are still many challenges lying ahead. In this paper, we have made efforts to reach the practical goal.

A. Hoare Logic

Hoare logic is the fundamental of our evaluation. It is based on predicate logic and provides a set of axioms to define the semantics of programming languages.

{~Spre} P {C∧D}

B. Test Oracle

Definition 1: Let Spost≡(C1∧D1)∨(C2∧D2)∨⋯∨ (Cn∧Dn),where each Ci (iϵ{1,....,n}) is a predicate called a "guard condition "without output variable, Di is called "defining condition" contains at least one output variable. Then a functional scenario fs is a conjunction $S_{pre}^{\sim}$∧Ci∧Di, and the expression ($S_{pre}^{\sim}$∧C1∧D1)∨($S_{pre}^{\sim}$∧C2∧D2)∨⋯ ∨($S_{pre}^{\sim}$∧Cn∧Dn)is called a functional scenario form(FSF) of S.

Definition2:Let f≡$S_{pre}^{\sim}$∧C∧D be a functional scenario of specification[3].

S, and P is program which is refined from the specification, t is the test case which is automatically generated, r is the result of P using t, then, if the following test oracle holds:

$S_{pre}^{\sim}$ (t)∧C(t)∧¬D(t,r)

it indicates that a bug is found in program P by t.[8]

As formal specification needs to use mathematical formula, in order to contain all the types to describe software requirements, we need to divide the specification into several kinds of type.

They are numeric, string, char, set, sequence, map, product, composite, compound, conjunction and disjunction. Following are detail technique for numeric, set, sequence and compound types.

III. TEST RESULT ANALYSIS

We apply bottom-up approach to describe test result analysis algorithms, starting from atomic predicate expressions and proceeding to compound, conjunctions and disjunctions. In the package which I developed, it contains 4 kinds of class. We first discuss the algorithms for predicate expressions involving only variables of numerical types, set types and sequence types, respectively, and then extend to their combination-compound types.

A. Structure of my package is as follows:

Package structure

Package automatic_test_evaluation;

Class Evaluation From Relations;

1) Method Boolean Evaluation From Numeric Exp(operator, Exp1, Exp2, Test case){...}

/* operator is a member of{>,<,=,>=,<=,<>}, Exp1 is numeric specification , Exp2 is corresponding program, which we assume that the result have been given. Test case is a file address where stores automatically generated test cases, which can be extracted and assigned to Exp1, the following 2) to 7) are similar */

2) Method Boolean Evaluation From String

Exp(operator,Exp1,Exp2,Testcase){...}

/* operator is a member of{=,<>} */

3) Method Boolean Evaluation From Enumeration

Exp(operator,Exp1,Exp2,Testcase){...}

/* operator is a member of{=,<>} */

4) Method Boolean Evaluation From Char

Exp(operator,Exp1,Exp2,Testcase){...}

/* operator is a member of{=,<>} */

5) Method Boolean Evaluation From Set

Exp(operator,Exp1,Exp2,Testcase){...}

/* operator is a member of{inset, notin, union, inter, diff, subset, psubset, =, <> } */

6) Method Boolean Evaluation From Sequence

Exp(operator,Exp1,Exp2,Testcase){...}

/* operator is a member of{len, S(i), conc, elems, inds=,<>} */

7) Method Boolean Evaluation From Compound

Exp(operator,Exp1,Exp2,Testcase){...}

/* operator is a member of
{numeric,set,sequence,string,char,enumeration and their combinations} */

B. Test case

Similar to automatic specification-based test result analysis, test case generation is also an automatic specification-based test case generation. Details are in another paper. As we know, the consistency of specifications can't be decided only on one group of test case, because only one group of test case's correctness can't be guaranteed. So we need several groups of test case.

Definition 1 : Let P be an atomic specification contains variables x1, x2....xn. Let Tc be the test case generated based on P, like t1,t2,....tn, they are a group of values bound to x1,x2....xn, respectively[12].

Definition 2 : Let Td be a test set for P. A test of P is a set of evaluations of P with all the test cases in the test set Td[12].

Definition 3 : A test suite for P is a set of pairs of test set and expected results corresponding to test cases[12].

For example, suppose P= x>0 ∧ y=x+1, where x and y are real numbers, respectively, then

(1) (x=1, y=2) is a test case for P.

(2) {(x=1, y=2),(x=3, y=4),(x=-4, y=-3)} is a test set for P.

(3)

x	y	Er (expected results)
1	3	false
-2	-1	true

According to the tool we developed, test case generated and test result analysis's result are stored respectively, which we use XML to store. Tool function will be shown later.

C. Test Objects

1) For numeric types.

For numeric types in this paper, we indicate four types defined in SOFL: zero, natural numbers, integers and real numbers. They are denoted by the symbols nat0,nat,int and real, respectively.

*Operators:*Unary,minus,Addition,Subtraction, Division Multiplication, , Less than, Greater than, Less or equal, Greater or equal, Less-between, Equal, Not equal

Operator	Name	Type
-X	Unary minus	real->real
X+Y	Addition	real*real->real
X-Y	Subtraction	real*real->real
X*Y	Multiplication	real*real->real
X/Y	Division	real*real->real

Table I. Relational Operators

Operator	Name	Type
X<Y	Less than	real*real->bool
X>Y	Greater than	real*real->bool
X<=Y	Less or equal	real*real->bool
X>=Y	Greater or equal	real*real->bool
X<Y<Z	Less-between	real*real*real->bool
X<=Y<=Z	Less-equal-between	real*real*real->bool
X=Y	Equal	real*real->bool
X<>Y	Not equal	real*real->bool

Table II. Relational Operators

Each operator above is a predicate that takes some arguments and yields a truth value.

Testing steps

We have 3 basic steps in automatic specification-based test result analysis for numeric type .

Step 1: Automatically test case generation. This step is described in Liu's another paper[3].

Step 2: Automatically recognize the numeric specification and assign the test case to both specification and program.

Step 3: Inspect the test oracle to find if there exists a bug in the program.

In figure 2, *x* is input variable and *y* is output variable. According to the post condition, the functional scenario is: x>=0 ∧ x>=5 ∧ y=x+3.

In the functional scenario, x>=0 is, x>=5 and x<5 is C(guard condition), y=x+3 and y=x-3 is D(defining condition).Then we assign the test case to functional scenario, and guard condition assignment is convenient, but defining condition is complicated. We use Reversed Polish Notation to transform the equation to postfix expression.

Reversed Polish Notation can transform an expression into postfix order, which is easy for computer to recognize and calculate. For example, a*(b+c)/d+e is a prefix notation, taking account into the operator precedence, we assign each operator a natural value to represent their precedence. As the follow shows:

Operators	"("	")"	* ,/	+,-
Precedence value	5	5	4	3

Among the operators, "("and ")" have the highest priority, we create 2 stack A and B, stack A stores operators, stack B stores the final expression.

Step 1:Scan the expression from left to right, if the char is operator, put it into stack A, or put it into stack B.

Step 2:Compare the next operator OP with the operator in A, if OP lager, put OP into A, or put OP into B. If OP is ")",then pop operators in A until " (" , and put them into B

according to first out first in principle, canceling a pair of "(" and ")" at the same time.
Step 3: Repeat step2 until the expression is completed scanned. Pop the operators in A to B according to first out first in principle.
Step 4: Pop all the chars in B.
Then the expression is transformed into:
a, b, c,+, *, d, /, e, +
Following is the data situation in stack A and B:

A	B
*	a
*,(	a
*,(	a, b
*,(,+	a, b
*,(,+	a, b, c
*	a, b, c,+
/	a, b, c,+, *
/	a, b, c,+, *, d
+	a, b, c,+, *, d, /
+	a, b, c,+, *, d, /, e
	a, b, c,+, *, d, /, e, +

Table III. Transforming Process

So "x+3" can be transformed to "x,3,+". After assignment, we go to next step.
In this paper, we focus on step 2 and step 3, and we assume that program result has been given using test case. What's more, only one test case is not enough, so we will generate a set of test case-test set.

Specification
Process Test(x: int) y: int
Pre x>=0
Post (x>=5 = >y=x+3) and x<5=>y=x-3

Fig II. Formal Specification

Input	Output	Test Condition	Defining Condition	Test Oralce
x	y	$S_{pre}^{\sim} \wedge C$	D	$S_{pre}^{\sim} \wedge C \wedge \neg D$
6	9	TRUE	TRUE	FALSE
7	9	TRUE	FALSE	TRUE
4	9	TRUE	FALSE	TRUE

Table IV. Test Result Analysis Tabel

We can obviously find bug in the analysis table above, when the test oracle's value is true, it indicates that a bug is found in the program cause the value of the program and the value of the specification are not the same.

In this paper, we focus on step 2 and step 3, and we assume that program result has been given using test case. What's more, only one test case is not enough, so we will generate a set of test case-test set.

2) For Set and Sequence Types

Set and sequence are types dealing with collection.

A set is an unordered collection of distinct objects where each object is known as an element of the set. Since computers can deal with only finite sets, we require that any set of a set type be finite. A set type is declared by applying the set type constructor to an element type.

A sequence is an ordered collection of objects that allows duplications of objects .As with sets, the objects are known as elements of the sequence

The difference between them is set is an unordered collection of distinct objects while sequence is an ordered collection of objects that allows duplications of objects.

Set contains the following operators: inset, notin, union, inter, diff, subset, psubset.

Sequence contains the following operators: len, S(i), conc, elems, inds.

The details of these operators are in Liu's book about SOFL.[7]

Let us consider a simple operation as an example. Suppose an operation *Borrow* in a Book register system is defined as follows:

```
process     Borrow(borrow-req:
            BorrowRequest)warning:string|Confirmation
            :string;
ext         rd :userlist;
            wr :booklist;
pre         exists[b:elems(booklist)]|b.name=borrow-
            req.bookname and b.ID  =borrow-req.ID
            and exists [u inset(userlist)] |
            u.username=borrow-req.username and
            u.userID=borrow-req.userID;
post        if borrow-req.HoldQuantity>=5
               return warning;
            else confirmationmes="success";
end-process
```

Where BorrowRequest is type defined in the section of the type declarations of the specification(omitted for brevity),string is the string type.
In specification, operator "exists[b:elems(booklist)]" is implemented in C# as the following method:

```
public Boolean elems(string b,string booklist,string testcase)
    {
      string[] sLine = File.ReadAllLines(testcase,
      Encoding.Default);
      string[] sep = new string[] { "=", "[", "]" };
      string[] booklist = sLine[0].Split(sep,
      StringSplitOptions.RemoveEmptyEntries);
      List<string> list = new List<string>();
      int count = 0;
      for (int j = 0; j < booklist[1].Length; j++)
      {
        string book = booklist[1].Substring(j, 1);
        if (book == "{")
        { count = j;
        }
        if (book == "}")
        {
list.Add(booklist[1].Substring(count, j - count + 1));
        }
      }
      for(int u=0;u<list.Count;u++)
      {
       for(int t=1;t<list.Count-1;t++)
       {
       if(list[u]==list[t])
        { list.Remove(list[t]);
        }
       }
      }
      string[] str = new string[list.Count];
      for (int k = 0; k < list.Count; k++)
      { str[k] = list[k];
      }
      if(str.contains(b))
        return true;
     else
        return false;
      }
```

In the program implementing the specification, as there may exists the situation like this: test case may contains embedded collection, for example:

[1,2,{3,4},5].According to set and sequence definition, element {3,4}is a single element, so we firstly divide the collection into "1","2","{3","4}","5" format, then recombine them based on "{" and "}",the elements between "{" and the first "}"after "{" are put together to form one element.

3) For compound types

Specification-based test result analysis for compound types is hard to handle because it is the combination of all the types mentioned above. Since the complexity depends on embedding among different data types, we divide the discussion into 3 kinds.

(a)	(b)	(c)
collection based	numeric based	logic based
diff(s1,union(s1,s2))	len(s)+q>2	x inset union(s1,s2)

Table V. Compound Type Classification

The following are details of the 3 types:
(a).expression contains only set and sequence, the result type of it is set or sequence type, where the operator contains are all belongs to set or sequence.
(b).expression contains "len" and numeric operators, the result type of it is numeric value or boolean value, where the operator contains are all belongs to numeric, set and sequence.
(c).expression contains "inset" operator, the result type of it is boolean value, where the operator contains are all belongs to set and sequence. Specifically, algorithms for dealing with each kind of situation are described below.

Alogrithm1: We apply recognizing while assigning method to deal with situation **(a)**.

Below the major step for testing situation (a) in compound type.

NO. of algorithms	Operators	Algorithms for situation (a)
(1)	Union, inter, diff	Step 1:Split expression into atomic expression. Step 2:Assign the atomic expression, replace the atomic expression with newly calculated value. Step 3: Rearrange expression ,if there exists no operator in expression, then output the final value. Otherwise, go back to step 2.
(2)	S(i), conc, elems, inds	Omitted ,similar to above

Table VI. Algorithms For Situation (a)

Algorithm 2:In this situation, we apply the algorithm mentioned previously. Details are in table 8.

No. Of algorithms	Operators	Algorithms for situation (b)
(1)	len,+,- ,*,/,<,>,<=,>=, <>,union,diff,inter,inds,conc,s(i),elems.	Step1:Calculate len operator using algorithm I. Step2:Transform the format into numeric format, using numeric algorithm

Table VII. Algorithms For Situation (b)

Algorithm 3:Situation (c) deals with "inset" operator. Cause expression is divided by "inset" into two parts .Algorithm is in table 10

No. Of algorithms	Operators	Algorithms for situation (c)
(1)	len,+,-,*,/,<,>,<=,>=, <>,union,diff,inter,inds ,conc,s(i),elems.	Step 1: separate expression into two parts by "inset" Step 2:according to each part, apply algorithm1 and algorithm2.

Table VIII. .Algorithm For Situation (c)

IV. Design of the Tool

The automatic specification-based test result analysis supporting tool is designed to support the process of reviews using the component-based package on the basis of SOFL specification language[4]. Specifically, the tool offers the following functions:

Automatically judge the consistency between the specification and the refined program. The way to do this is to automatically recognize the specification and call the related package which is developed in C#. Then compare the program result with the specification result after assigning the test case to specification.

V. Related Work

The automatic specification-based test result analysis is introduced in Liu's paper. In this thesis, we describe a tool that supports most kinds of the testing objects. In addition, there have existed some tools to support test result analysis using other testing methods. In this section, we overview the existing test result analysis tools and the approaches underlying the tools.

O.B.Bellal and his team present the analysis of test results in the context of OSD communication protocols. They use principles involved in the comparison of test results with respect to a reference specification which may be non-deterministic. Their analysis tool TETRA performs the analysis for specifications written in the formal description technique LOTOS[9], which is a formal specification language developed by ISO for the description of OSI communication protocols and services.

VI. Conclusion and Future Work

In this paper, we present an approach to applying the specification-based test result analysis method into a tool support. We described the component-based package to solve the specification-based testing. And the tool realized automatic test case generation and automatic test result analysis.

In the future, we are interested in extending the test objective to more data types, like map, composite, conjunction and disjunction, etc.

References

[1] Fumiko Nagoya, Shaoying Liu, Yuting Chen, "A Tool and Case Study for Specification-Based Program Review", The 29th Annual International Computer Software and Applications Conference (COMPSAC2005), Edinburgh, Sotland, July 25-28, 2005, IEEE Computer Society Press, pp. 375-380

[2] Shaoying Liu, ``An Approach to Applying SOFL for Agile Process and Its Application in Developing a Test Support Tool", Innovations in Systems and Software Engineering, Springer London, 11334_6(1), 22 December, DOI 10.1007/s11334-0114-3.

[3] Shaoying Liu, "Automatic Specification-Based Testing: Challenges and Possibilities", 5th Intl. Conf. on Theoretical Aspects of Software Engineering, IEEE CS Press, Xi'an, China, Aug. 29-31, 2011, pp. 5-8.

[4] Fumiko Nagoya, Shaoying Liu, Yuting Chen, "Design of a Tool for Specification-Based Program Review", Workshop on SOFL in the 10th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS2005), Shanghai, China, 16-20 June 2005, IEEE Computer Society Press, pp. 10-11.

[5] Shaoying Liu and Hao Wang `` Shaoying Liu, "Pre-Post Notation is Questionable in Effectively Specifying Operations of Object-Oriented Systems", Frontier of Computer Science in China, DOI 10.1007/s11704-011-0130-y, 2011, pp. 1-12.

[6] Shaoying Liu, "Formal Engineering for Industrial Software Development using the SOFL Method", Springer-Verlag, March 2004, 428 pages, ISBN 3-540-20602-7.

[7] Shaoying Liu, "Developing Quality Software Systems Using the SOFL Formal Engineering Method", Proceedings of 4th International Conference on Formal Engineering Methods (ICFEM2002), LNCS

[8] Shaoying Liu, ``Verifying Consistency and Validity of Formal Specifications by Testing", Proceedings of World Congress on Formal Methods in the Development of Computing Systems, FM'99 - Formal Methods, Lecture Notes in Computer Science, No. 1708, Springer-Verlag, Toulouse, France, September 20-24, 1999, pp. 896-914.

[9] O.B. Bellal, G.v. Bochmann, M. Dubuc, F. Saba, " Automatic Test Result Analysis for High-Level Specifications" — 1991