A PATH-ORIENTED AUTOMATIC RANDOM TESTING BASED ON DOUBLE CONSTRAINT PROPAGATION

Ruilian Zhao[1] Yuandong Huang[2]

[1]Dept. of Computer Science, Beijing University of Chemical Technology, China

rlzhao@mail.buct.edu.cn

2 Beijing Zhongke Fulong Computer Technology Co.,Ltd

yuandong.h@gmail.com

ABSTRACT

A key issue in software testing is the actual generation of test data from program input domain. Obviously, more accurate input domain is, more efficient test generation is. This paper presents a path-oriented automatic random testing method based on double constraint propagation. For a given path, its domain can be reduced by splitting an input variable domain and executing a double constraint propagation algorithm. Moreover, a random test data generator is developed according to the reduced path domain and the test experiments are conducted on a number of programs. Experimental results show that the method gets more accurate path domain than PRT (path-oriented random testing) approach, and random testing efficiency can thus be enhanced by using the proposed method.

KEYWORDS

Random testing; Path condition; Double constraint propagation; Automatic testing generation

1. INTRODUCTION

Software testing is one of the most important and practical techniques to ensure software quality. One challenging task of software testing is to selecting test cases that effectively detect faults at a minimum cost. Many testing approaches have been developed to guide the test data generation [1-5]. One simple and common method is Random Testing (RT), in which test data are selected in a random manner from the program's input domain. Although some researchers criticized RT for no information about the software under test (SUT) to guide its test case selection, many studies show that RT is effective in detecting faults not found by other methods [6-9]. For example, in a recent study researchers at NASA applied a RT tool to a file system used aboard the Mars rover. The random testing tool created hundreds of failing tests that revealed previously unknown errors, despite the fact that many other manual and automated testing techniques had already been applied to the system [8]. As a result, RT has been shown to be a very useful tool in the hands of software tester as it is simple, unbiased and the cost is lower than others [9]. Among these advantages, one key advantage of RT over other techniques is that it selects objectively the test data by ignoring the specification or the structure of SUT. Therefore, RT has been widely used in much industrial software testing [10-19].

DOI : 10.5121/ijsea.2012.3201

Path testing is a well-known software testing technique. The basic idea in path testing is to find at least one test data to activate each selected path. In [10], A. Gotlieb et al. introduced an approach, called path-oriented random testing (PRT), to perform RT at the path level. This approach used backward symbolic execution to derive path conditions corresponding to a selected path, and computed an approximation of the input subdomain by using constraint propagation and constraint refutation over finite domains. Then, a uniform random generator was applied to the approximated subdomain to generate test data. However, it can be observed that some invalid inputs still exist after applying the PRT approach.

So, in this paper, we present a path-oriented automatic random testing method based on Double Constraint Propagation. The approach gets the constraint set of the input variables by source code analysis techniques, and a double constraint propagation algorithm is used to compute the domain of input variables along a chosen path. Then, a random test generator is invoked to generate test data for the selected path. The experiment results show that the domain gotten by our method is more accurate than the PRT, and random testing efficiency can thus be enhanced by using the proposed method.

The remainder of this paper is organized as follows. Section 2 introduces some basic terminologies and random testing technology. Section 3 describes path-oriented random testing strategy. Section 4 present our random test data generation method based on double constraint propagation. Section 5 reports experimental results to show that the method is effective and practicable. Section 6 overviews related work. Finally, the conclusions are presented in Section 7.

2. BASIC TERMINOLOGY AND RANDOM TESTING

2.1 Basic Terminology

A program structure can be represented by a *control flow graph (CFG).* A *path* is a sequence of nodes from entry to exit node. An edge (v_i,v_j) is called a *branch* if node v_i corresponds to a decision statement at which the control flow has two or more alternative execution routes, such as **if–then–else**, **switch**, **for** or **while** statements in C programs. Each branch in a control flow graph can be labeled with a *predicate* that describes the conditions under which the branch will be traversed. A predicate is usually connected with a *predicate interpretation* that is obtained by replacing each variable appearing in the predicate with its symbolic value in terms of input variables. Each path is associated with a *path condition* that is the conjunction of all the predicate interpretations that are taken along the path. The path condition represents the constraints that have to be satisfied for inputs in order to execute the path. If path condition corresponding to a path has no solution, meaning the path is non-feasible. So, solving the path conditions yield either to find a test datum on which the path is traversed or to show that the corresponding path is non-feasible.

2.2 Random Testing

Random testing is a basic and simple software testing technique, which selects test cases at random from the set of all possible program inputs. When used to detect software failures, RT often selects test cases according to a uniform distribution strategy, that is, all program inputs have the same probability to be chosen as test cases. For example, for a program with 2 input variable x and y, its input domain D can be represented as $D=Dx \cup Dy$, where *Dx/Dy,* called variable domain*,* is a set of all values that input variable *x/y* can hold. RT can be implemented

just by selecting x and y from its domain at random, respectively, meaning when selecting y without paying attention on the value obtained for x. In other word, the two variable values are independently determined. Obviously, if the domain of x or that of y can be reduced, the test generation on the invalid domain can be avoided. Therefore, a key question of RT is how to get a precise input domain. If it is difficult to obtain a precise input domain, we hope to get the most approximate solution to the one.

3. PATH-ORIENTED RANDOM TESTING

PRT was first proposed by Gotlieb et al [5] in 2006. PRT works like random testing with the different that selected test data at random to cover a given subset of paths according to a uniform probability distribution over the program's input domain. More specifically, PRT applied constraint propagation to get input domain along a given path, and then a path-oriented random test data generation was performed. The goal of constraint propagation was to shrink the finite variation domain of each variable in order to get an approximation of the solutions with respect to a set of constraints. The PRT algorithm took as inputs a set of variables, a constraint set corresponding to the path conditions of the selected path, and a division parameter k (a given parameter). The algorithm separated each variable domain into k equal sub-domains. If the size of a variable domain could not be divided by k, the domain was enlarged until its size could be divided by k. By iterating this process over all the n input variables, the input domain would be partitioned into k^n sub-domains. The sub-domains that could not satisfy the path constraints would be omitted. As a result, some invalid inputs were removed, so the test generation efficiency could be increased. For instance, showed by figure 1, the obtained input domain along a path was $d_2 \cup d_3 \cup d_4$, and the sub-domain d_1 was removed by PRT approach.

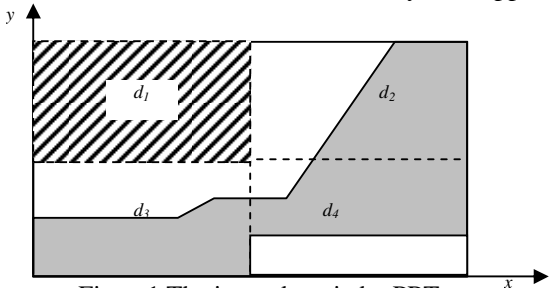

Figure1 The input domain by PRT

4. PATH-ORIENTED RANDOM TESTING BASED ON DOUBLE CONSTRAINT PROPAGATION

It can be found from Figure 1 that some invalid inputs still exist after applying the PRT approach. The main reason is the PRT approach enlarges the domain until it can be divided by k if it cannot. In this way, some invalid inputs may be introduced. In order to reduce the input domain further, we propose a path-oriented automatic random testing method based on double constraint propagation technique, called DCPRT.

In what follows we will describe in detail how to apply the DCPRT algorithm to automatically generate random test data with respect to each path of the program under test.

4.1 Overview of DCPRT

Firstly, the DCPRT method analyzes the program under test, and gets a path constraints set with respect to each path of tested program by using program analysis technology. Secondly, a path domain is computed and reduced with the help of the double constraint propagation strategy. In particular, a constraint propagation algorithm is applied to obtain the input domain D_1 along a selected path. Then, a variable domain, for example D_1x, is divided into two parts, and the constraint propagation algorithm is employed again to each of the sub-domains. As a result, a more accurate input domain D_2 is obtained, which contains less invalid domain. Finally, the random test data are generated according to a uniform probability distribution over the domain D_2.

For example, for the input domain D_1 by PRT, as shown in Figure 1, the result of domain D_2 by using DCPRT method is displayed in Figure 2. After the constraint propagation algorithm is applied firstly, we get the input domain $D_1=D_1x \cup D_1y=d_2 \cup d_3 \cup d_4 =\{(x,y)|x \in (0,x_2),\ y \in (0,y_3)\}$. Then x variable domain D_1x is portioned into two part, and the constraint propagation algorithm is used secondly. As a result, the input domain $D_2=d_2 \cup d_{32} \cup d_{41}=\{(x,y)|x \in (0,x_1),\ y \in (0,y_2)\} \cup \{(x,y)|x \in (x_1,x_2),\ y \in (y_1,y_3)\}$ is obtained. It can be seen that D_2 is more accurate than D_1, and the DCPRT method not only remove d_1, but also eliminate the invalid domains d_{31} and d_{42}.

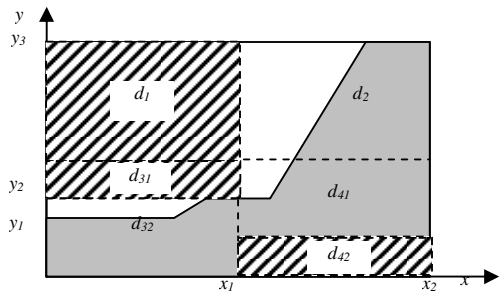

Figure2 The input domain by DCPRT

4.2 Constraint Set Collection

As mentioned above, a path has a *path condition* which represents the constraints that have to be satisfied for inputs in order to execute the path. This is to say, an input variable is associated with a set of possible values. A constraint is a relation defined on subsets of these variables and denoted valid combination of their values. Constraints restrict possible values that the variables can take. In fact, constraints refer to logistic or arithmetical expressions with respect to input variables along a chosen path, which imply that input variables satisfy some special conditions, namely some restrictions about input variables. A constraint set is the set of the restrictions along the selected path. For example, the path 1-2-3-4-5 of program *foo,* showed in table 1, corresponding constraint set is $R=\{x<=100 \ \&\& \ y<=100;\ y > x + 50;\ x * y < 60\}$. In this program code segment, the symbol *ush* is short for unsigned short int.

Table1 Example Program *foo*

```
    ush foo(ush x, ush y){
1.  if (x<=100 && y<=100){
2.  if (y > x + 50)
3.  ...
4.  if (x * y < 60)
5.  ...
```

In the research reported in the paper, we construct a mapping relation between special functions as well as special expressions (EF/EO) and constraint formulas, shown in table 2, on the basis of analysis on many C-language codes. A special function (expression) refers to the function that has some special requirements to input variables. Table 2 lists some special functions (expression) and their corresponding constraint formulas. If some functions or expressions in the left list are encountered along a path, then the corresponding constraint formulas are added into its constraint set.

Table 2 The relation between EF/EO and constraint formula

EF/EO	Constraint Formula
log(x)	$x > 0$
sqrt(x)	$x \geq 0$
x+=C	$x_{min} \leq x + C \leq x_{max}$
C/x	$x \neq 0$
…	…

The constraint set collection is completed by using program analysis technology. In detail, firstly, the program is analyzed, its control flow graph is gained, and paths are produced by Breadth-First search strategy. Secondly, for each path, corresponding branch conditions, special functions, special expressions and so on, are collected with the help of GCC [20]. As a result, we get the constraint set for each path.

4.3 Double Constraint Propagation

In order to obtain over-approximated input domain, we use a double constraint propagation algorithm on the constraint set to compute its input domains along a selected path. The double constraint propagation algorithm does not simply employ the constraint propagation two times, but is a process of approaching to the real input domain by reducing input variable domains step by step according to the appearance order of input variables. For example, for the program showed in Table 1, the constraint set along the path 1-2-3-4-5 is as follows:

$$R = \begin{cases} x <= 100 \&\& y <= 100 \cdots\cdots (1) \\ y > x + 50 \cdots\cdots\cdots\cdots\cdots\cdots (2) \\ x * y < 60 \cdots\cdots\cdots\cdots\cdots\cdots (3) \end{cases}$$

The variable *x* and *y* are all unsigned short *int* type. The initial input domain is $D_0=\{x \in (0,+\infty), y \in (0,+\infty)\}$, The approximate solution gotten from constraint (1) is $D_1=\{x \in (0,100), y \in (0,100)\}$, the solution gotten from constraint (1) and (2) is $D_2=\{x \in (0,49), y \in (51,100)\}$, and the solution gotten from constraint (1) to (3) is $D_3=\{x \in (0,1), y \in (51,100)\}$. D_3 is the result of using the constraint propagation firstly, we can see that when $x=1$ and $y>60$, it can not satisfy the constraint. So we apply the constraint propagation again, the result is $D_4=\{x=0, y \in (51,100)\} \cup \{x=1, y \in (52,60)\}$.

The algorithm of the double constraint propagation is as follows:

Algorithm: Double constraint propagation
Input: *R* — Constraint set along a given path（*the number of the constraint is n*）
I — The set of input variables
Output: *D* — The input domain along the given path
S1. Initial the input domain, get the domain D_1
S2. Apply the constraint propagation to R
  S2.1. Let *i=1*
  S2.2. Compute the i^{th} constraint, get the new domain D_i
  S2.3. *i = i + 1*
  S2.4. If *i <= n*, repeat to S2.2, Else S3.
S3. Find a variable from set *I* whose value is more than 1 until there is no variable can be selected, then S5.
  Divide the variable domain into two parts, get the domain D_{n+1} and $D_{n+2.}$
S4. Apply the constraint propagation to the two sub-domain.
  Goto S2.1
S5. Output the final domain *D*

Moreover, we develop a prototype based on the double constraint propagation algorithm for random testing. It uses above algorithm to get reduced input domain, then generates random test data automatically according to a uniform probability distribution, and tests the program under test.

5. EXPERIMENTAL AND EVALUATION

To evaluate the double constraint propagation based path-oriented random testing (DCPRT), we compare it with path-oriented Random Testing (PRT) by computing the reduced input domain on program *foo* and *wage.*

5.1 Experiments on Program *foo*

For program *foo* in table 1, we compute the input domain with respect to the path 1-2-3-4-5 by using RT, PRT and our DCPRT method, respectively. The results of input domain as well as corresponding test points are showed in Table 3. The input domain by RT contains 10201 possible test points, whereas the input domain by PRT consists of 68 which much are fewer than RT. It can be further observed that there are still some invalid points, such as (x,y)=(0,100) in the domain of PRT. The invalid points can be omitted since no enlargement is introduced when a variable domain is divided by using the DCPRT. So, there is no invalid domain and points added. The input domain by DCPRT is made of 59 possible points in *foo* program, less than PRT. Hence, obtained input domain along a given path is more accurate by using the DCPRT method.

Table 3 The input domain of the *foo* program

Approach	Input domain	Test points
RT	$\{x \in (0,100), y \in (0, 100)\}$	10201
PRT(k=2)	$\{x=0, y \in (51,100)\} \cup \{x=1, y \in (51,67)\}$	68
DCPRT	$\{x=0, y \in (51,100)\} \cup \{x=1, y \in (52,60)\}$	59

In addition, the DCPRT method considers the special functions and special expressions, while is not involved in PRT. For example, if there is a special function *log(x)* before the first *if* statement of the program showed in Table 1, x=0 must be removed from its input domain. As a result, the final input domain should be $D=\{x=1, y \in (52,60)\}$.

5.2 Experiments on Program *wage*

Table 4 gives a program of computing real wage of the retired officials, shorted for *wage*. For this program, we compute the input domain with respect to the path 1-2-3-4-5-6-7-8-9 by using RT, PRT and DCPRT, respectively. The input domain as well as corresponding test points are given in Table 5. It is clear that the DCPRT method can get more reduced input domain than PRT.

Table 4 The *wage* program

```
// compute real wage of the retired officials (tax rate is below 20%)
//x: the proportion according to the number of the retired years, multiply 100
//y: the wage of one month just before retiring
ush calc_wage(ush x, ush y){
1.  int r = 0;        //test variable, is used to record the executed path.
2.  int wage = 0;  //the retired wage after tax
 //judge the proportion (50%-90%) is correct or not
3. if(x>=50 && x<=90){
4.   r += 0x1;
 //judge the wage scope is valid or not
5.   if(y>=500 && y<=20000){
6.        r += 0x2;
//suppose when income is less than 5000, the tax rate is 15%, more is 20%
7.        if(x*y<=500000){
8.              r += 0x4;
9.              wage = x * y * (100 - 15) / 100; //compute the wage
                 //print the wage list
                … } } }
return r;
}
```

Table 5 The input domain of the *wage* program

Approach	Input domain	Test points
RT	$x \in (50,90), y \in (500,10000)$	380000
PRT(k=2)	$\{x \in (50,70), y \in (5251,10001)\} \cup \{x \in (71,91), y \in (5251,10001)\}$ $\cup \{x \in (50,70), y \in (500,5250)\} \cup \{x \in (71,91), y \in (500,5250)\}$ $\cup \{x=91, y \in (5251,10001)\} \cup \{x \in (50,90), y=100001\}$	389543
DCPRT	$\{x \in (50,70), y \in (500,10000)\} \cup \{x \in (71,90), y \in (500,7402)\}$ $-\{x \in (71,91), y \in (7403,10000)\}$	262338

More detailed, the original result of input domain is $D_1=(x \in (50,90), y \in (500,10000))$, by using

program analysis technology, showed in Figure 3 with shadow. In order words, the test data are generated randomly from D_1, i.e. RT.

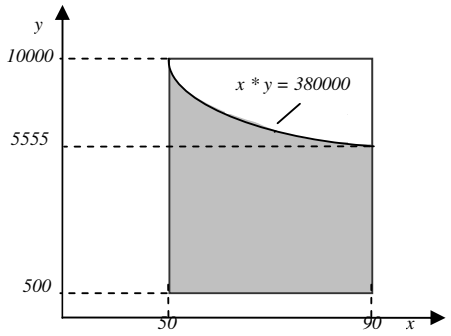

Figure 3 The input domain of *wage* by first step

And then, the PRT approach divides the domain of all input variables into 2 same parts. As a result, the domain D_1 is divided into four equal sub-domains, i.e., d_1~d_4 showed in Figure 4. More detail, $d_1=\{x \in (50,70), y \in (5251,10001)\}$, $d_2=\{x \in (71,91), y \in (5251,10001)\}$, $d_3=\{x \in (50,70), y \in (500,5250)\}$ and $d_4=\{x \in (71,91), y \in (500,5250)\}$.

In addition, $d_5=\{x=91, y \in (5251,10001)\} \cup \{x \in (50,90), y=100001\}$ is the enlarged part by using PRT. The final reduced result is $D_{PRT}=\{x \in (50,91), y \in (500,10001)\}$. For the program *wage*, PRT fails to reduce the input domain; on the contrary, it introduced (10001-500+1)+(90-50+1) =9543 invalid points.

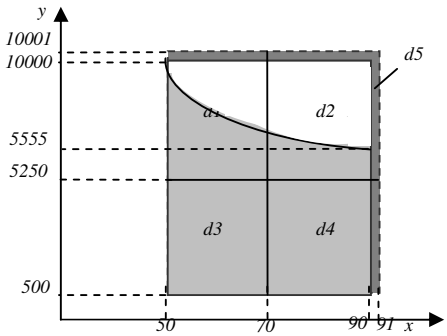

Figure 4 The input domain of *wage* by PRT(k=2)

The input domain gotten by using DCPRAT is $D_{DCPRT}= \{x \in (50,70), y \in (500,10000)\} \cup \{x \in (71,90), y \in (500,7402)\}$, and it removed $d_2=\{x \in (71,91), y \in (7403,10000)\}$. Hence, (10000-7402)*(90-70)=51960 invalid points are eliminated, showed in Figure 5.

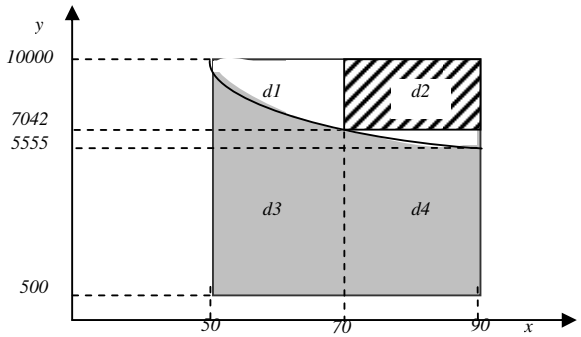

Figure 5 The input domain of *wage* by DCPRT

So, we can see that PRT introduce some invalid points, but the DCPRT method never introduces new invalid points. And at the same time, it eliminates some invalid points. As a result, the input domain can be reduced effectively.

6. RELATED WORK

Random testing is a widely used software testing technology. The fundamental idea behind random testing is that it generates test inputs at random from the input domain of SUT. Researchers and practitioners have implemented random test tools for various areas, including Unix utilities, Windows GUI applications, Haskell programs, and object-oriented programs [15]. But the key of the random test tools is the same, this is to say, when a choice is to be made in constructing a test input, a random generator makes the choice randomly, not deterministically [16-18].

PRT is a path-oriented random testing technique that aims at generating randomly test data that execute a path within a program. A. Groce et al employed constraint propagation and constraint refutation to get a uniform sequence of test data that triggers a selected path [5, 10]. Godefroid et al proposed a randomized algorithm that generates test suites to activate a path by using symbolic execution and constraint propagation over finite domains in the tools DART (Directed Automated Random Testing) [19]. They got very good experimental results on C programs extracted from real-world applications.

In contrast to PRT, our DCPRT method employs a double constraint propagation strategy to obtain a more accurate path domain.

7. CONCLUSION

In this paper, we propose a path-oriented automatic random testing method based on double constraint propagation strategy. The constraint set is obtained along the selected path by program analysis techniques, the reduced input domain is computed by using double constraint propagation strategy, and random test data are generated on the reduced input domain. Moreover, our DCPRT is compared with the PRT method. It can be showed that our method can get more precise input domain than PRT. As a result, the random testing effectiveness can thus be remarkably enhanced by using the proposed method.

The input domain acquired by using double constraint propagation can be not only used in test data generation, but also can be applied in many other areas for software testing automation. The further work is to extend the DCPRT method to deal with more types of test data, achieve the test

data generation for the boundary of the input domain, and enlarge the scope of application.

ACKNOWLEDGEMENT

This work was supported in part by National Natural Science Fund of China under Grant No.61073035 and No.60903002.

REFERENCES

[1] R. Zhao, Lyu, Michael R. and Y. Min. Automatic string test data generation for detecting domain errors, Software. Testing, Verification and Reliability, 20(3):209-236, 2010.
[2] B. Korel. Automated Software Test Data Generation, IEEE Transactions on Software Engineering, 16(8): 870-879, 1990.
[3] Ali, Shaukat and Briand, Lionel C. and Hemmati, Hadi and Panesar-Walawege, Rajwinder Kaur, A Systematic Review of the Application and Empirical Investigation of Search-Based Test Case Generation, IEEE Transactions on Software Engineering, 36(6):742-762, 2010.
[4] H.Tahbildar and B.Kalita. Heuristic Approach of Automated Test Data Generation for Programs Having Array of Different Dimensions and Loops with Variable Number of Iteration. International Journal of Software Engineering & Applications, 1(4):75-93, October 2010.
[5] A. Gotlieb and M. Petit. Path-oriented random testing. In 1st ACM Int. Workshop on Random Testing (RT'06), Portland, Maine, July 2006.
[6] J. Duran and S. Ntafos. An Evaluation of Random Testing. IEEE Transactions on Software Engineering, 10(4):438–444, Jul. 1984.
[7] D. Hamlet and R. Taylor. Partition Testing Does Not Inspire Confidence. IEEE Transactions on Software Engineering, 16(12):1402–1411, Dec. 1990.
[8] A. D. Groce, G. Holzmann, and R. Joshi. Randomized differential testing as a prelude to formal verification. In ICSE '07: Proceedings of the 29th International Conference on Software Engineering, Minneapolis, MN, USA, 2007.
[9] A. Arcuri, M. Z. Iqbal, and L. Briand. Formal analysis of the effectiveness and predictability of random testing. In ACM International Symposium on Software Testing and Analysis (ISSTA), 219–229, 2010.
[10] A. Gotlieb and M. Petit. A Uniform Random Test Data Generator for Path Testing, Journal of Systems and Software, 83, 12: 2618-2626,2010.
[11] A. Arcuri and L. Briand. Adaptive Random Testing: An Illusion of Effectiveness? In ACM International Symposium on Software Testing and Analysis (ISSTA), 265-275, 2011.
[12] T. Y. Chen, F. Kuo, and Z. Zhou. On favourable conditions for adaptive random testing. International Journal of Software Engineering and Knowledge Engineering, 17(6):805–825, 2007.
[13] T. Y. Chen, F. C. Kuo, and H. Liu. Distributing test cases more evenly in adaptive random testing. Journal of Systems and Software (JSS), 81(12):2146–2162, 2008.
[14] T. Y. Chen, F. C. Kuo, and H. Liu. Application of a failure driven test profile in random testing. IEEE Transactions on Reliability, 58(1):179–192, 2009.
[15] C. Pacheco. Directed Random Testing, Doctor Thesis, Massachusetts Institute of Technology 2009.
[16] T. Y. Chen, F. Kuoa, R. G. Merkela, and T. Tseb. Adaptive random testing: The art of test case diversity. Journal of Systems and Software (JSS), 83(1): 60-66, 2010.
[17] F. C. Kuo, T. Y. Chen, H. Liu, and W. K. Chan. Enhancing adaptive random testing for programs with high dimensional input domains or failure-unrelated parameters. Software Quality Journal, 16(3):303–327, 2008.
[18] A. F. Tappenden and J. Miller. A Novel Evolutionary Approach for Adaptive Random Testing. IEEE Transactions on Reliability, 58(4): 619-633, 2009.
[19] P. Godefroid, N. Klarlund, & K. Sen. Dart: Directed automated random testing. In Proceedings of ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'05) 213–223, 2005.
[20] M. S. Richard and the GCC Developer community. Using the GNU Compiler Collection (GCC) – For GCC 4.1.1[Z/OL]. Free Software Foundation, Inc. 9-10, 2005.

Author

Ruilian Zhao received her B.S. and M.S. degree in computer science from North China Industry University in 1985 and 1990, and Ph.D. degree in computer science from Institute of Computing Technology, Chinese Academy of Sciences in 2001. She is now a professor at Department of Computer Science, Beijing University of Chemical Technology. Her current research interests include software testing and fault-tolerant computing.

Yuandong Huang received his Computer Science and Technology Bachelor degree from Beijing University of Chemical Technology, Beijing, China, in 2006. Since 2006, he has been working towards the Master degree of Computer Applications Technology at Beijing University of Chemical Technology. His research interests test data generation for dynamic data structure. Now, he has worked in Beijing Zhongke Fulong Computer Technology Co.,Ltd.