2012 International Conference on Image, Vision and Computing (ICIVC 2012)
IPCSIT vol. 50 (2012) © (2012) IACSIT Press, Singapore
DOI: 10.7763/IPCSIT.2012.V50.18

Improvement on Solving the Constraint System in Automated Test Data Generation for Database Stored Procedure Testing

Feng Liyun[a,*], Zeng Qiang[a] and Hong Mei[a]

[a] Computer Science college of Sichuan University, Chengdu and 610065, China

Abstract. Stored procedure has occurred independently in database application systems. How to test the stored procedure effectively becomes an urgent problem in automated testing. Test data generation is a vital part in automated test of stored procedure. However, the current approaches of test data generation for stored procedure needs manual intervention and the solution of the constraint system which limits test data can not effectively cover all the situations. Existing approaches can not solve constraint system with character string and strict inequality. This paper improved the approach to generate test data, perfected existing approach to solve constraint system and reduced the limitation on solving the constraint system.

Keywords: Software Automated Testing, Database System Testing, Database Stored Procedure Testing; Automated Generation of Test Data, Solution for Constraint System

1. Introduction

In many leading-edge domains such as aviation, spaceflight and military, the vital mission systems are based on database system. The quality of database is the groundwork for these mission systems and needs to be assured by testing the database system. Stored procedure, as an advanced and significant function in database system, has been widely applied because of its advantages-- it can simplify the high-level application development, reduce the network flow, enhance the security and so on. Therefore, stored procedure should be tested as a separate segment. Since stored procedures are stored in back-end server, testing them is difficult and limited by result visualization problem.

The major difference between stored procedure testing and general software testing is that the database states will influence the test result. That means even if the input parameters are same, the execution result of the stored procedure might be different under different database state. Furthermore, the data amount of one database state is usually large, so it is inefficient to construct testing data manually or even it is impossible to construct manually when massive test data is needed. Stored procedure testing is hence necessary to be automated.

In order to implement automated tests of stored procedure, there are four problems need to be solved:

a. Analysis of stored procedure code and automated generation of test data (include input parameter and database states)
b. Automated generation of expected test results
c. Automated execution of test program (scripts) and record of the test result.
d. Automated analysis of test result and attained conclusion

To solve the first problem---automated generation of test data, Mei Hong[1] has proposed the method aimed at automated generation of test data for stored procedure on the basis of generating test data automatically for the third programming language. The method is divided into three steps: First, establish the constraints of test data. Second, solve the constraints. Finally, generate test data according to solution result of

* Corresponding author.
E-mail address: xiaoyuer198856@163.com.

the constraints. The existing algorithm reverse query processing and multi-reverse query processing have solved the issues of automated generation and combination of database states[2],[3]. These algorithms make it possible to test stored procedure automatically.

However, current approaches on solving the constraint system can not meet the requirements of automated test cases generation for stored procedure testing. Character strings and strict inequality appear in the constraint system inevitably. But general approaches can not solve this kind of constraint system, so it is necessary to refine and improve the existing approaches about solving constraint system.

2. Related Work

Nikolai Tillmann[4] proposed an unit test tool PEX for .NET. Pex can produce a small test suite with high code coverage for a .NET program. It performs a systematic program analysis to determine test inputs for Parameterized Unit Tests and also learns the program behavior by monitoring execution traces. Koushik Sen[5] addressed the problem of automating unit testing with memory graphs as inputs and proposed an approach to exploring all feasible execution paths using a combination of symbolic and concrete execution to generate test inputs. Zhongxing Xu[6] described a prototype tool, called SimC, which automatically generates test data for unit testing of C programs. The tool symbolically simulates the execution of the given program and is capable of generating test data for programs involving pointer and structure operations. J. Zhang[7] described an implemented toolkit whose goal is to find values for input variables such that a terminal state can be reached. J. Zhang[8] described constraint-based tools that can be used for decide the feasibility of paths. Sen, K[9] put forward the use of CUTE（Concolic Unit Testing Engine）in Java. The researches above are primarily focused on how to establish path-driven constraint system and find the solution of them for the third generation programming language. The problem is that they do not relate to the database state

MeiHong [1] studied the issue of automated generation of test case for stored procedure testing and proposed a method to generate constraints for stored procedure which is the basis of this paper. However, the character string variables in the constraint system still remain unsolved.

3. The Improvement of Solution on Constraint System for Stored Procedure Testing

Current general approaches to solve the constraint system can not meet the requirements of automated test cases generation for stored procedure testing. In the constraint system, character strings and strict inequality appear inevitably. General approaches can not solve this kind of constraint system.

Variables can be divided into two types, which are string variable and numerical variable. Numerical value can not have direct relationship with string variable itself, but properties of the string such as the length, the location of a character, which associated with the numerical values. When strings and numeric variables both occur in a judgment condition, we can define one numerical variable to replace one property of the string. After replacing all the string property, there are only numerical variables in the judgment condition. According to the type of variables appeared in the determining condition, we divide constraint system into two types: strings variables constraint system and numeric variables constraint system. The process of improved solution of constraint systems is shown in Figure 1:

3.1. Solution of strings variables constraint system

One string variable judgment condition has two variables and a judgment relationship at most---- E1 and E2 can be either a string variable or a constant. As the transitivity of judgment relationships, string variable judgment condition finally comes down to the judgment relationship between a string variable and a string constant, as shown in (1).

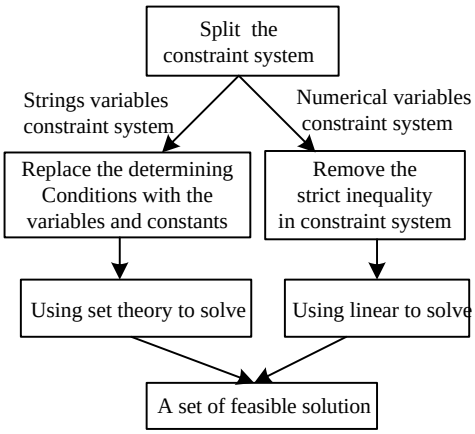

Figure 1 Process of the Improved Solution of Constraint System

The solutions of the two types are described in detail as follow.

$$R:\begin{cases} r_1: E_1^{r_1}\ OP\ E_2^{r_1} \\ r_2: E_1^{r_2}\ OP\ E_2^{r_2} \\ \quad \square \\ r_n: E_1^{r_n}\ OP\ E_2^{r_n} \end{cases} \Leftrightarrow Var\ OP\ Con \tag{1}$$

Var represents string variable, and Con represents string constants.

The string is an ordered set of characters, and it can be expressed as <A, ≦ >, where A represents all the characters in a string, ≦ represents the preorder relationship between the characters. There is preorder relationship between any two characters, so <A, ≦ > also is a line ordered set. Thus, solving a string variable judgment conditions could be converted to solve the line ordered sets <A, ≦ >, as shown in (2).

$$Var\ OP\ Con \ \Leftrightarrow\ \text{<X, ≦ > OP <C, ≦ >} \tag{2}$$

X represents sets of string variable, C represents string constants, and OP represents judgment relationship.

We will discuss the solution of string determining condition whit different OP separately:

a) <X, ≦ > = < C, ≦ >

According to the extension theorem, two sets are equivalent only if they have the same members ---- X = C. And <A, ≦ > is a line ordered set, so there is only one unique solution Con for Var.

b) <X, ≦ > ≠ < C, ≦ >

According to the extension theorem, any two sets are not equivalent when they contain different elements, or different number of elements, so two strings is not equal when the length of Var is not the same as Con or the characters is different at the same location.

c) <X, ≦ > ⊆ < C, ≦ >

<X, ≦ > is a subset of < C, ≦ >. As < A, ≦ > is a line ordered set, there are two line ordered sets <con1, ≦ > and <con2, ≦ > making the following equation established:

<C, ≦ > = <con1, ≦ > + < X, ≦ > + <con2, ≦ >

d) <X, ≦ > ⊄ < C, ≦ >

<X, ≦ > is not a subset of < C, ≦ >. So any two ordered sets <con1, ≦ > or <con2, ≦ > will make the following inequation established:

<C, ≦ > ≠ <con1, ≦ > + < X, ≦ > + <con2, ≦ >

e) <X, ≦ > ⊇ < C, ≦ >

<C, ≦ > is a subset of <X, ≦ >. As < A, ≦ > is a line ordered set, there are two line ordered sets <con1, ≦ > and <con2, ≦ > making the following equation established:

<X, ≦ > = <con1, ≦ > + <C, ≦ > + <con2, ≦ >

Adding a string of any length (including zero) before the first character or after the last character of Con could be the value of Var.

f) <X, ≦ > !⊇ < C, ≦ >

<C, ≦ > is not a subset of < X, ≦ >. So any two ordered sets <con1, ≦ > or <con2, ≦ > will make the following inequation established:

<X, ≦ > ≠ <con1, ≦ > + < C, ≦ > + <con2, ≦ >

When X contains less elements than C , C contains some elements not in X or < C, ≦ >contains preorder relationships not in <X, ≦ >, the above inequation is established.

We can successful get the value of variables in the judgment condition which contains string variables and constants using the method above. But there are a prerequisite that there are constants in the constraint system, and string variables have direct or indirect relationship with the constants. Otherwise, all the variables in the constraint system are unsolvable.

The algorithm of solving string variable constraint system is shown in Figure 2.

Solution of numerical variables constraint system

Previous studies often use linear programming, nonlinear programming and genetic algorithms to solving numerical variables constraint system.

In the linear programming, the linear programming model is defined as shown in (3) ,(4)and (5):

$$\max Z = CX;$$
$$s.t\ AX = b,$$
$$X \geq 0. \tag{3}$$
$$A = (a_{ij})_{m\times n}, n > m \tag{4}$$
$$r(A) = m, X \in R^n, b \in R^m, b \geq 0, C = (c_1, c_2, \cdots, c_n) \tag{5}$$

The result of linear programming solution is often the point on the border. So it makes the linear programming method is only applicable to loose constraint condition such as X≥0 but not applicable to strict inequality such as X>0 [10] . But in stored procedure, strict inequality will inevitably occur in those constraint systems for each path which is because of the complementarities of the paths.

The idea of nonlinear programming solution is using approximate method to convert the nonlinear constraints system into similar linear constraints system, and then using the linear programming. Therefore, the nonlinear programming also demands the judgment condition to be non-strict inequality [10].

Although genetic algorithms avoid the constraints of strict inequality, it brings in the defects of genetic algorithm. One drawback is that the genetic algorithm depends on the initial population, but initial population is often generated randomly which makes the speed and the results of solution can not be guaranteed.

The improved solution of numerical variables constraint system is described as follows.

During testing stored procedure, we only need to execute stored procedure path once to test whether the data is processed correct or not in the certain path. So we don not need the complete solution set of the constraint system. We can make appropriate adjustment on the numerical variables constraint system as long as the solution results of the adjusted constraint system do not expand. Based on this idea, we can convert the strict inequality to non- strict inequality.

The improved method is as follows. First, transform the inequality to make sure that there are only constants at the right of inequality. If the inequality does not contain a constant, we can add 0. Then convert strict greater than to non-strict greater than, and increase the constant a unit; convert strict less than to non-strict less than, and decrease the constant a unit. For instance x <1 can be converted to x <= 0.

After such adjusted, constraint system does not contain strict inequality. The new constrain system can meet the requirements of linear programming. Then we use linear programming to solve the adjusted constraint system. Linear programming solve method is quite mature; the paper will not elaborate on it. We use LP_solve tool to solve constraint system.

4. Experimental Verification XPERIMENTAL VERIFICATION

Example: testing path constraint system of stored procedure is shown in (6):

$$\begin{cases} name \neq \text{"bush"} \\ list \supseteq name \\ sum - base > 10000 \\ (sum - base) / base > 10\% \\ sum = count \times price \\ base = count \times 30 \\ price < 60 \end{cases} \tag{6}$$

After splitting and adjusting the constraint system above, we can get the string variable constraint system and numerical variable constraint system as shown in (7)and (8).

$$\begin{cases} name \neq \text{"bush"} \\ list \supseteq name \end{cases} \tag{7}$$

$$\begin{cases} sum - base \geq 10001 \\ (sum - base) / base \geq 11\% \\ sum = count \times price \\ base = count \times 30 \\ price \leq 59 \end{cases} \tag{8}$$

Solve the string and the numerical variable constraint system respectively and get a group of solution as shown in(9):

$$\begin{cases} name = \text{"white"} \\ list = \text{"morewhite"} \\ price = 50 \\ count = 600 \\ sum = 30000 \\ base = 18000 \end{cases} \tag{9}$$

According to the results of constraint system mentioned above, it is evident that the method of solving constraint systems introduced in this paper can efficiently solve constraint systems containing character strings or strict inequality.

5. Conclusion and Future Work

With the expanding use of stored procedure, the automated stored procedures test is an invertible trend. In this paper, we extract the requirements on test data and the database state and refer to reverse query processing algorithm, which effectively reduces the manual intervention in test data generation process. Meanwhile, the solving algorithm for string variable constraint system and the improvement of solving algorithm for numerical variables constraint system, effectively reduce the limitation of the string and strict inequality on the restraint system.

Due to the limitation of time and capability, the data in the solution of the constraint system which generated by the approach introduced in our paper has no semantic meaning. How to solve the constraint system and generate test data with semantic meaning is the future research.

6. References

[1] Mei Hong, An Approach of Automated Test Cases Generation in Database Stored Procedure Testing[C], The 2nd International Workshop on Education Technology and Computer Science,2010

[2] C. Binnig, D. Kossmann, and E. Lo. Reverse query processing. In ICDE '07: Proceedings of the International Conference on Data Engineering, 2007:pages 506—515

[3] Carsten Binnig, Donald Kossmann, Eric Lo. Multi RQP Generating Test Databases for the Functional Testing of OLTP Applications. Technical report, ETH Zurich, 2008

[4] N. Tillmann and J. de Halleux. Pex - white box test generation for .NET. In Proc. Second International Conference on Tests and Proofs (TAP). 2008.

[5] Koushik Sen , Darko Marinov , Gul Agha, CUTE: a concolic unit testing engine for C, Proceedings of the 10th

European software engineering conference held jointly with 13th ACM SIGSOFT international symposium on Foundations of software engineering, 2005

[6] Zhongxing Xu , Jian Zhang, A Test Data Generation Tool for Unit Testing of C Programs, Proceedings of the Sixth International Conference on Quality Software, 2006,p.107-116

[7] J. Zhang, C. Xu, and X.Wang. Path-oriented test data generation using symbolic execution and constraint solving techniques.In Proceedings of the Second International Conference on Software Engineering and Formal Methods, 2004.

[8] J. Zhang and X. Wang. A constraint solver and its applicationto path feasibility analysis. International Journalof Software Engineering and Knowledge Engineering, 2001,11(2):139–156,

[9] Sen, K., Agha, G.: CUTE and jCUTE: Concolic unit testing and explicit path model-checking tools. In: Ball, T., Jones, R.B. (eds.) CAV 2006. LNCS, vol. 4144, pp. 419–423.

[10] Zhang Guang-mei，Li Xiao-wei. Automatic Generation of Basis Path Set in Path Test [J]. Computer Engineering,2007(22)