Journal of Applied Sciences

ISSN 1812-5654

Journal of Applied Sciences 13 (13): 2372-2377, 2013
ISSN 1812-5654 / DOI: 10.3923/jas.2013.2372.2377

Algorithm Based on Data Flow Criteria for Automatic Test Data Generation

Jifeng Chen and Hao Peng
School of information science and engineering, Hunan International Economics University,
Changsha, 410205, China

Abstract: By analyzing the theory of the Definition-Use (DU) test for data flow testing, an algorithm based on data flow criteria for automatic test data generation was presented in this study. The ALL-DU-PATHS data flow criterion was introduced and the Warshall method was used to check the feasibility and testability of DU pair in the algorithm. An automatic test data generation approach based on test sequence (path) was given after selecting test sequence (path) on DU pair coverage. Finally, the algorithm was checked by example and experiment. Theoretical analysis and test results show that the algorithm can effectively check the DU pair's feasibility and testability of the program and automatically generate test data for the test sequence (path) to cover the DU pairs, or get the conclusion which could not found the effective program's input to cover the DU pairs.

Key words: Data flow criteria, DU pair, predicate function, test data

INTRODUCTION

Software testing is a process for finding errors of program. During the process, the program is executed with a group of test data which is detailedly designed as input according to the software specification and the internal structure of program. Software testing can be divided into structural testing (Alshraideh *et al.*, 2010; Diaz *et al.*, 2008) and functional testing (Lijun *et al.*, 2011; Nie and Leung, 2011) on the base of test data design. The adequacy of the software structural test is judged by the criterion based on whether some parts of software have been tested. In recent years, a lot of structural testing criteria are presented, such as statement coverage (Tikir and Hollingsworth, 2005; Tao *et al.*, 2006), branch coverage (Alshraideh *et al.*, 2011; Kiran *et al.*, 2010) and path coverage (Junko *et al.*, 2009; Gong *et al.*, 2011). Rapps-Weyuker defined a set of data flow test guidelines for an ideal programming language (Rapps and Weyuker, 1985). Data flow testing is a structural test method which uses the relationship among the flow data in the program to guide the tester selected test cases. The basic idea is: the definition of a variable can affect the value of another variable or the choice of the path through the further using and defining. Therefore, some test data can be chosen to make an execution for the program in accordance with a definition-use path of certain variables. The result is verified whether it is consistent with expectations. Thus, the errors of program could be discovered.

The generation of test sequences based on data flow was analyzed in (Liu *et al.*, 2005). Based on it, the test sequence generation method was improved in this study and a new method of automatic generation of test data was proposed and verified by examples.

BASIC KNOWLEDGE

In order to better illustrate the algorithm's design idea and to maintain the integrity of the article, related knowledge is repeated as follows:

The theory of the definition-use test based on data flow: Data flow testing is the form of structural testing which concerns the variable value of the receiving points and the use(or reference) points . Most of the formal works for definition- use test theory of data flow are completed in the early 80's of the 20th century (Liu *et al.*, 2005). Suppose process P followed a structured program design specifications, V is its set of program variables, P is the set of all paths in the PATH (P), P picture shows the program G (P), according to the definition of data flow testing/use test theory, has the following definition (Jorgensen, 2003).

Definition 1: Node $n \in G$ (P) is the definition of the variable node $v \in V$, denoted DEF (v, n), if and only if the value of v is defined by the statement fragment of the corresponding node n.

Corresponding Author: Jifeng Chen, School of information science and engineering, Hunan International Economics University, Changsha, 410205, China

Enter the statement, assignment statements, loop control statements and program calls, are the examples of the definition of node statement. If this statement is the implementation of the corresponding node, then the variable associated with the contents of the storage unit will change.

Definition 2: Node n∈G (P) is the use of the variable node v∈V, denoted by USE (v, n), if and only if the value of v is used by the statement fragment of the corresponding node n.

Output statements, assignment statements, conditional statements, loop control statements and program calls, are the examples of using the nodes in the statement. If this statement is the implementation of the corresponding node, then the variable associated with the contents of the storage unit will remain unchanged.

Definition 3: If a variable v∈V is defined in the statement m (DEF (v, m)) and used in the statement n(USE (v, n)), then the statement said the statement m and n is called as a definition-use pairs of the variable v, referred to DU pairs (denoted as <v, m, n>).

Definition 4: For the definition-use path of variable v (denoted as DU-PATH) is the path in PATH (P), there is the definition node DEF (v, m) and the use node USE (v, n) for some v∈V, making m and n is the initial and final nodes in the path.

Definition 5: The definition-clear path of variable v (denoted as DC-PATH) is the path in PATH (P) which has the initial node DEF (v, m) and the final node USE (v, n) to make that no other nodes are the definition nodes of v in the path.

Definition-use path and definition-clear path give a description of the source statements date flow on the cross-path value from the defined point to the used point.

Definition 6: If there is a DC-PATH in a DU pair, then the DU pair is to be tested, otherwise, it is not.

The test coverage standard on the definition-use path of data flow: The standard of Rapps-Weyuker based on data flow analysis (Liu *et al*., 2005) mainly includes ALL-DU-PATHS,ALL-USES,ALL-C-USES,ALL-P-USES, ALL-DEFS, ALL-EDGES, ALL-NODES. The containment relationship is shown in Fig. 1.

In this case, it is possible for a more detailed structure testing between the full path indicators (impossible to achieve) and all sides indicators which are generally considered the lowest. The test of definition-use can provide a rigorous and systematic method on checking points that defect may occur.

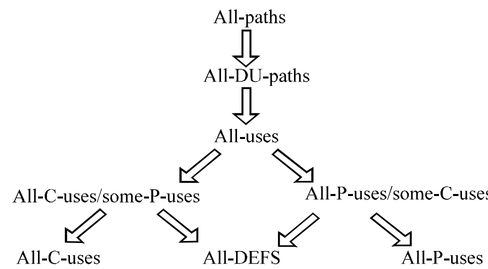

Fig. 1: The containment relationship among the data flow testing criteria

When choosing a test standard, there must be some kind of trade-offs. The stronger selecting standard is, the easier detecting bug in the program is. At the same time, the cost of testing will also be higher. On the other hand, the test cases will be decreased if the weaker standards are used and the generating costs are lower. We use the second strongest standard ALL-DU-PATHS as the criterion for algorithm designing, i.e., the test data that is generated should be able to cover the DU pair owned by each variable. As the variable value is optional, it is not guaranteed that these DU pairs can detect all the existent bugs (Jianguo, 2001).

ALGORITHM DESIGN

The main algorithm's idea is as follows:

Analyzing the program to determine DU pairs: For the programs written by the imperative programming languages, the program flow graph is a directed graph, where the nodes in it are either the entire statement or a part of the statement and the edges express the control flow. Therefore, we can use graph theory to analyze the program to determine the DU pair.

Step 1: By analyzing the program under test, we determine the nodes and variables and the relationship between each node, the type (definition node or use node) that each node relative to each variable. According to the analysis of the relationship between the nodes information, construct program flow graph, find the current adjacency matrix and determine the DU pairs owned by each variable in the program according to the current adjacency matrix and the node type that each node relative to each variable

Checking the feasibility and testability on DU pairs:

Step 2: Determine the accessibility matrix of program graph. Using the graph theory (Mingzhe, 2010) and the Warshall algorithm, the accessibility matrix P can be directly obtained from the adjacency matrix A. As follows (where j, i, respectively stand for the row and column of matrix, n is the number of nodes in the program graph):

- Set a new matrix P: = A
- Set i: = 1
- For all j, if P (j, i) = 1 while k = 1, 2,..., n then P (j, k): = P (j, k)∈P (i, k)
- i: = i +1
- If i = n, go to step 3, otherwise stop

Step 3: Check the feasibility of all DU pairs(the first and last nodes of DU pairs are accessible) according to the accessibility matrix, remove the infeasible DU pairs (the first and last nodes of DU pairs are inaccessible)

Step 4: In accordance with definition 6, check the testability of all DU pairs, remove the DU pairs which are not testable (i.e., the DU pairs that do not contain the DC-PATH): For one of the DU pairs, after removing the definition nodes of the same variables except corresponding to the DU pairs, we can get a new adjacency matrix and calculate the accessibility matrix of the adjacency matrix. If the DU pair is still viable in the new adjacency matrix, the DU pair can be tested, otherwise deleted

Step 5: Not considering the variable name, delete the DU pair that the definition-use node is the same and repeated and get the final DU pair's set $\{DU_i \mid i = 1,2 ... n\}$

Optimizing and selecting the test sequences (paths) which cover the DU pairs: Suppose that all DU pairs are the set {DUi|i = 1,2..., n} and the selected test sequence (path) set is P = {Pj | j = 1,2 ... m}:

Step 6: Choose a path P_1 which is through the DU_1 and check P_1 whether it is also through DU_2. If it is true, then $P = \{P_1\}$; otherwise, elect a path P_2 through DU_2 and check P_2 whether it is also through DU_1. If it is true, then $P = \{P_2\}$; otherwise, $P = \{P_1, P_2\}$

Step 7: When i = k, j = l (k = 1, l = 1), check the path set $P = \{P_j|j = 1, 2..., l\}$ whether there is a path through the DU_k. If true, $P = \{P_j|j = 1, 2,..., l\}$; otherwise, elect another path P_{l+1} through the DU_k and check whether P_{l+1} is also through $\{DU_i|i = 1, 2,..., k-1\}$. If true, $P = \{P_{l+1}\}$; otherwise, $P = \{P_j|j = 1, 2...l, l+1\}$

Since, all the DU pairs in set $\{DU_i|i = 1, 2,..., n\}$ are feasible and testable, so there must be a path set which covers all the DU pairs. Repeat step 6, we can find the test sequence (path) $P = \{P_j|j = 1, 2,..., m\}$ which covers the set $\{DU_i|i = 1, 2,..., n\}$.

Generating test data for the selected test sequence: Inspect the various branch predicates on chosen path, if the various branch predicates are all the linear expression, then carry out step 8, otherwise carry out step 9:

Step 8: Using branch predicate function on the path to construct linear constraint system on the input variables directly and set up the input variable linear equations, solve the input variables value I. The value of input variables shall be the test data. If the restraint system has no solution, then the path is not accessible

Step 9: Choosing a set of input variable values in given domain to check each branch predicate on the path, the linear arithmetic representation of nonlinear predicate functions on current input is computed. The linear constraint system on the input variables is constructed with the linear predicate functions on the path and the linear arithmetic representations obtained previously. Further, the linear equation system on the input variables is established and solved to get the values of input variables. Hence, a set of new input is obtained. If the set of new input can't traverse the given path yet, then, the process above is repeated till the desired outcome is obtained or the number of iterative upper limit is achieved(the path is infeasible to a large extent)

If there are some infeasible (or infeasible in large extent) paths in path set $P = \{P_j|j = 1, 2,..., m\}$, generate a new set of DU pairs using the DU pairs which are not covered by the feasible paths in set P . Choosing another path except for the set $P = \{P_j|j = 1, 2,..., m\}$, repeat 3.3 and 3.4. If the DU pairs which are not passed still exist and there is no new path that can be selected, then it is true that there is not any test data which can be generate for covering the DU pairs. The algorithm ends.

EXPERIMENT

In order to compare with (Liu *et al*., 2005), we use the same program shown in Fig. 2 to verify the validity of the algorithm.

Search for all DU pairs: By analyzing the program in Fig. 2, the definition and use of variables for each nodes in the program's graph can be obtained and the same to all the feasible and testable DU pairs. It was shown in Table 1.

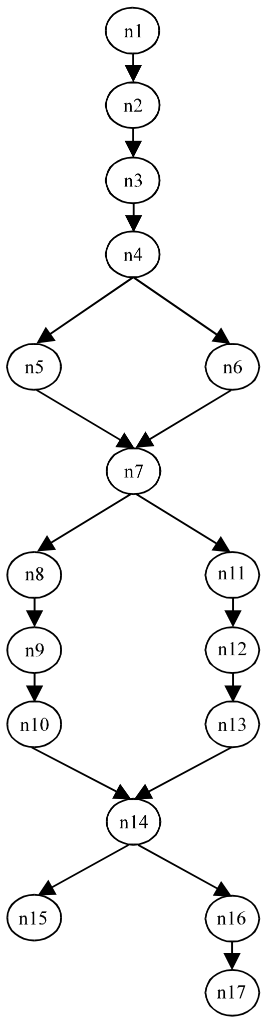

Fig. 2: An example program with linear function, quadratic function and the sin function. (1) Float X, Y, Z, U, W, (2) Scanf (“%f %f %f”, &X, &Y, &Z), (3) U = (X-Y)×2, (4) if(X>Y), (5) W = U, (6) Else W = Y, (7) if((W+Z)>100) (8) X = X-2, (9) Y = Y+W, (10) Print f(“Linear”);}, (11) Else if(X*X-Z*Z> = 100){ (12) Y = X×Z+1, (13) Printf(“NonLinear: Quadratic”);}, (14) If (U>0), (15) Printf(“%f”,U), (16) Else if((Y-Sin(Z))>0) and (17) Printf(“Nonlinear: Sine”)

Checking the feasibility and testability on DU pairs: Using graphic theory and Washall algorithm, find out the accessibility matrix of the program flow graph. According to the accessibility matrix and Definition 6, check the feasibility and testability on DU pairs and all the feasible and testable DU pairs can be obtained. It was shown in Table 2.

Without considering the variable name, remove the DU pairs which have the same definition and use nodes, we can get the following DU pair's set:

$$DU = \{\langle 2,3\rangle; \langle 2,4\rangle; \langle 2,6\rangle; \langle 2,7\rangle; \langle 2,8\rangle; \langle 2,9\rangle; \langle 2,11\rangle; \langle 2,12\rangle; \langle 2,16\rangle; \langle 3,5\rangle; \langle 3,14\rangle; \langle 3,15\rangle; \langle 5,7\rangle; \langle 5,9\rangle; \langle 6,7\rangle; \langle 6,9\rangle; \langle 9,16\rangle; \langle 12,16\rangle\}$$

Optimizing and selecting the paths of covering the DU pairs: In accordance with 3.3, the following three paths are easy to find for covering the DU pair's set:

$$P_1 = \{1, 2, 3, 4, 5, 7, 8, 9, 10, 14, 15\}$$

$$P_2 = \{1, 2, 3, 4, 6, 7, 8, 9, 10, 14, 16, 17\}$$

$$P_3 = \{1, 2, 3, 4, 6, 7, 11, 12, 13, 14, 16, 17\}$$

Seeking the test data of P1, P2, P3: First seeking the test data of P_1. Step 7 is executed due to all the predicate functions on path P_1 are linear. Construct the linear constraint system of predicate functions on the input variables directly:

$$\begin{cases} X - Y > 0 \\ 2X - 2Y + Z - 100 > 0 \\ 2X - 2Y > 0 \end{cases}$$

Table 1: All of the DU pairs

Variables	The DU pairs
X	<X,2,3>; <X,2,4>; <X,2,8>; <X,2,11>; <X,2,12>; <X,8,11>; <X,8,12>
Y	<Y,2,3>; <Y,2,4>; <Y,2,6>; <Y,2,9>; <Y,2,16>; <Y,9,12>; <Y,9,16>; <Y,12,16>
Z	<Z,2,7>;<Z,2,11>;<Z,2,12>;<Z,2,16>
U	<U,3,5>;<U,3,14>;<U,3,15>
W	<W,5,7>;<W,5,9>;<W,6,7>;<W,6,9>

Table 2: All the feasible and testable DU pairs

Variable	The feasible and testable DU pairs
X	<X,2,3>; <X,2,4>; <X,2,8>; <X,2,11>; <X,2,12>
Y	<Y,2,3>; <Y,2,4>; <Y,2,6>; <Y,2,9>; <Y,2,16>; <Y,12,16>; <Y,9,16>
Z	<Z,2,7>;<Z,2,11>;<Z,2,12>;<Z,2,16>
U	<U,3,5>;<U,3,14>;<U,3,15>
W	<W,5,7>;<W,5,9>;<W,6,7>;<W,6,9>

Using the solving method of linear constrain system in (Shan *et al*., 2002; Edvardsson and Kamkar, 2001), the solution is X = 2, Y = 1, Z = 100. Therefore, the new input I_1 = (2, 1, 100). As P_1 can be passed by I_1, so I_1 is the sought test data.

Seek the test data for P_2: Checking path P_2 in Fig. 2, step 8 is executed as the predicate function on the 16th node is nonlinear. Given any arbitrarily chosen input I_0 in the program domain and iterative increments of input variables ΔX, ΔY and ΔZ. e.g., $I_0 = (X_0, Y_0, Z_0) = (1, 2, 3)$, ΔX = ΔY = ΔZ = 1.

The path P_2 is not traversed on I_0, so the steps for iterative refinement of I_0 are executed.

Since, the predicate function on the 16th node is F = Y-Sin (Z), the linear arithmetic representation of it can be represented as follows:

$$L(BP5, I_k, P) = bY+cZ+d$$

Approximate the derivatives of a predicate function by its divided differences, then b = 1, c = 0.89792:

$$bY_0+cZ_0+d = Y_0-\mathrm{Sin}\,(Z_0)$$

Solving the equation, d = -2.83488. The linear arithmetic representation of predicate function F = Y-Sin (Z) on I_0 is:

$$L(BP16, I_0,P_2) = Y+0.89792Z-2.83488$$

Construct the linear constraint system of predicate function on I_0 using linear predicate functions on P_2 and the linear arithmetic representation $L(BP16, I_0,P_2)$:

$$\begin{cases} X-Y\le 0 \\ Y+Z-100>0 \\ 2X-2Y\le 0 \\ Y+0.89792Z\text{-}2.83488>0 \end{cases}$$

Using the solving method of linear constrain system in P_1, the solution is X = -80.15, Y = -79.15, Z = 180.5. Therefore, the new input I_1 = (3.235, 3.835, 51.196).

The iterative refinement I_1 is needed to executed repeatedly since the path P_2 is not traversed on I_1 yet. Finally, the path P_2 is traversed on I_3 = (101.125, 101.725, -0.398) obtained in the third iterative. Therefore, the algorithm is determined. I_3 is the desired test data.

The test data of P_3 can be obtained as X = 1, Y = 1, Z = 0 according to the method described above. Thus, when the test data are (2, 1, 100), (101.125, 101.725, -0.398) and (1, 1,0), respectively, all the DU pairs can be covered.

The instance has been verified in the computer with the system CPU P4 1.6G, 512M DDR, Linux OS (Red Flag 4.1).

CONCLUSION

The main idea of the new algorithm is theoretically analyzed in this study. The proposed method was verified with an example. From the experimental results, the algorithm can correctly generate test data to cover all of the DU pairs. Though four paths are sought to cover all of the DU pairs in (Liu *et al*., 2005), the available test data are not generated. Furthermore, of which 2 paths are infeasible after calculating and the DU pair's coverage only reaches 69% in fact. Compared with (Liu *et al*., 2005), the algorithm in this study only generates three feasible paths and the DU pair's coverage reaches 100%

ACKNOWLEDGMENT

This project supported by Hunan Provincial Natural Science Foundation of China (No. 10JJ6092).

REFRENCES

Alshraideh, M., B.A. Mahafzah and S. Al-Sharaeh, 2011. A multiple-population genetic algorithm for branch coverage test data generation. Software Qual. Control, 19: 489-513.

Alshraideh, M., L. Bottaci and B.A. Mahafzah, 2010. Using program data-state scarcity to guide automatic test data generation. Software Qual. Control, 18: 109-144.

Diaz, E., J. Tuya, R. Blanco and J. Javierdolado, 2008. A tabu search algorithm for structural software testing. J. Comput. Operat. Res., 35: 3052-3072.

Edvardsson, J. and M. Kamkar, 2001. Analysis of the constraint solver in UNA based test data generation. ACM SIGSOFT Software Eng. Notes, 26: 237-245.

Gong, D., W. Zhang and X. Yao, 2011. Evolutionary generation of test data for many paths coverage based on grouping. J. Syst. Software, 84: 2222-2233.

Jianguo, W., 2001. Generation of protocol test set based on extended finite state machine. J. Software, 12: 1197-1204.

Jorgensen, P.C., 2003. Software testing: A Craftsman's Approach. 2nd Edn., Mechanical Industry Press, China, pp: 143-159.

Junko, H., S. Shigeo and S. Hiroshi, 2009. Path coverage properties of randomly deployed sensors with finite data-transmission ranges. Comput. Networking, 53: 1014-1026.

Kiran, L., M. Phil and H. Mark, 2010. An empirical investigation into branch coverage for C programs using CUTE and AUSTIN. J. Syst. Software, 83: 2379-2391.

Lijun, M., W.K. Chan, T.H. Tse and R.G. Merkel, 2011. XML-manipulating test case prioritization for XML-manipulating services. J. Syst. Software, 84: 603-619.

Liu, Y., M. Zeng, L. Zhu, J.F. Chen and J.W. Yan, 2005. Automatic test sequences generation technology based on data flow rules. Microelectronics Comput., 22: 131-135.

Mingzhe, L., 2010. Graph Theory and Algorithm. Mechanical Industry Press, China.

Nie, C. and H. Leung, 2011. The minimal Failure-causing schema of combinatorial testing. ACM Trans. Software Eng. Methodol., 20: 1-38.

Rapps, S. and E.J. Weyuker, 1985. Selecting software test data using data flow information. IEEE Transac. Software Eng., SE-11: 367-375.

Shan, J.H., J. Wang, Z.C. Qi and J.P. Wu, 2002. Improvement of the gupta method. Chinese J. Comput., 25: 1378-1386.

Tao, L., F. Jian-Ping, L. Xiao-Wei and L. Ling-Yi, 2006. Observability statement coverage based on dynamic factored Use-definition chains for functional verification. J. Electronic Testing, 22: 273-285.

Tikir, M.M. and J.K. Hollingsworth, 2005. Efficient online computation of statement coverage. J. Syst. Software, 78: 146-165.