Interface Byte Stream Parsing Algorithm for Black Box Test Cases
Automatic Generation

To cite this article: Cao Jianping et al 2018 IOP Conf. Ser.: Mater. Sci. Eng. 428 012001

View the article online for updates and enhancements.
Interface Byte Stream Parsing Algorithm for Black Box Test Cases Automatic Generation

Cao Jianping*, Sun Wenzhu, Zhou Xiuzhi, Zhu Guotao and Hu Wenting
Naval Aviation University Qingdao Branch, 266041, QingDao, China
*jp_cao2016@126.com

Abstract. Traditional black box test cases are mostly generated by manual methods and are difficult to automate. This is caused by the diversification of software module's input interface. In this paper, an intelligent black box test case automatic generation algorithm referred to as Interface Byte Stream Parsing (IBSP) is proposed. In this algorithm, the input interface of the software module to be tested is firstly abstracted as the number of bytes occupied by the input parameters of the module, and a section of dynamic memory is applied, then a binary sequence is generated in the dynamic memory by using Dynamic Memory Increment (DMI) or Random Sampling (RS) method. Follow that, the type information contained in the interface is inputted through the software module to be tested. The binary sequence is parsed into the input data of the software module to be tested, and the input data is filtered by using the filter conditions. Finally, the test expectations given by the user and the filtered input data constitute the test cases. For improving the performance of the algorithm, a testing experiment is conducted. The experimental result shows that the algorithm facilitates the automatic generation of test cases in software testing, improves the efficiency of test case generation, and has been initially applied.

1. Introduction
Software testing is an important means for ensuring software quality. An automated testing process can increase test efficiency and reduce cost. The black box test is a commonly used software testing method. It regards the program to be measured as a black box that cannot be opened. Without considering the internal structure and characteristics of the program, the test is performed from the correspondence between the input data and the output data[1]. During the black box automated testing process, automatic acquisition of test cases is a key and difficult point. So, it is of great significance to research automatic black box test cases generation.

It is generally considered that the exhaustive testing of input data is a task that cannot be completed. For this reason, traditional black box testing methods including equivalent division method, boundary value analysis method, causality graph analysis method and error guessing method usually select some representative input samples as test cases with the assistance of computer or artificially. These methods cannot be defined as test case automatic generation methods.

The automatic generation of test cases is the research hotspot currently. Most of the researches focus on the white-box testing method and generate test paths for overlay programs, for example genetic algorithms [2], particle swarm optimization algorithm[3] and other intelligent optimization algorithms. But the studies about black-box test cases generation are less.

In terms of applications, Nubian Corporation has published a tree-based black box test case generation method [4]. It first obtains n interface parameters of the software module to be measured...
and constructs them into an inverted tree structure. Finally, a path from the root node to child node forms a test case. Based on this method, valid use cases with higher coverage can be searched and system defects can be located quickly. A patent from Harbin Institute of Technology[5] proposes a method for generating embedded black box test cases based on dynamic models, which focuses more on the management of black box testers and data. But the methods given above are strongly dependent on the interface parameters of the software module to be tested, if the interface parameter types and numbers change, the test cases need to be recreated. From this perspective, these methods have not achieved the goal of universal and automatic test case generation.

This paper proposes a novel black box test case automatic generation algorithm referred to as Interface Byte Stream Parsing (IBSP). In this algorithm, the input interface of the software module to be tested is firstly abstracted as the number of bytes occupied by the module input parameters, and a corresponding byte of dynamic memory is applied. Then, a binary sequence is generated in the dynamic memory. Finally, the binary sequence is parsed into the input data of the software module to be tested through the type information contained in the input interface of the software module, and the input data may be filtered by using the filter condition. The test expectations given by the user and the filtered input data constitute the test cases. The binary sequences mentioned above can be generated by using Dynamic Memory Increment (DMI) or Random Sampling (RS) method which is explained in chapter 3.2 and 3.3. A testing experiment is conducted to assess the performance of IBSP algorithm. The experimental result shows that the algorithm facilitates the automatic generation of test cases in programming and improves the efficiency of test case generation.

2. Theoretical basis
For computer systems, the information can be expressed by "bit + context." All information in a computer system including disk files, programs in memory, user data stored in memory, and data transmitted over the network are represented by a string of bits [6].

The data in the computer memory is expressed by bit sequences composed of 0 and 1. The same data can be interpreted in different meanings for different application backgrounds. For example, for the 8 byte binary sequence 0x43AE044534EE0443 (in hexadecimal notation), if it resolves to \{int p1, int p2\}, its value is \{888013891, 1135477829\}; if it resolves to \{double p3\}, its value is 1.0814649073392767e18. Therefore, according to the corresponding context of the input interface parameter set, the generated binary sequence can be parsed into the input sample.

According to the current requirements, the input interfaces of the module under test are diversified, but the same parameter can be abstracted from the input interface parameter set of the module under test M, that is, the number of bytes occupied by the input interface parameters. For example, if the input interface parameter set of the module M1 to be tested is \{int p1, int p2\}, the number of bytes is 8; the input interface parameter set of the module M2 to be tested is \{double p3\}, and the number of bytes is also 8.

According to the byte number of the module M, a corresponding size of the dynamic memory is applied in the computer, and a certain method is used to generate a binary sequence in the dynamic memory independently. The binary sequence is then parsed using the context contained in the input interface parameter set to obtain test samples. In this process, the difficulty of test cases automatic generation is how to generate a binary sequence on a given dynamic memory. By researching, two generation methods are given in this paper, one is Dynamic Memory Increment (DMI) method and the other is Random Sampling (RS) method. The DMI method is to give a specific size of memory block and initialized to a special value, then increase the value by a fixed value to generate test cases. The RS method is to randomly generate a binary data generation test on a given memory.

3. Algorithm implementation

3.1. Basic process of IBSP
The steps of IBSP algorithm are as follows:
Step1: The input Interface Parameter Set represented as IPS of the software module M to be tested is given by the user.

Step2: According to the IPS, the size of its byte number represented as C is obtained, the Dynamic Memory represented as DM with the size C is requested, and the first address of DM is obtained, the first address of dynamic memory is represented as DA.

Step3: According to DA, we can address to the DM, and then a Binary Sequence represented as BS is generated.

Step4: Use the IPS to parse the BS, and the Instantiated Test Sample Data (ITD) is obtained.

Step5: A filter condition represented as FC is provided by the user to determine whether the ITD is in conformity with the FC, if so, go to Step6, otherwise go to Step3;

Step6: A set of Test Case (TS) is composed of the ITD and the Test Expectation (TE) which is given by user in advance.

The flow chart of IBSP algorithm is shown in Figure 1.

![Flow chart of IBSP algorithm](image)

In step 3, the BS can be generated by DMI method or RS method, which will be discussed separately below.

3.2. DMI method
The DMI (dynamic memory increment) method first apply for a dynamic memory block, next specify the memory block as a data type, and then continue to add a specific value to the memory block to generate a binary sequence. When the initial value is 0 and the step size is 1, the value of dynamic memory can be traversed. This method is equivalent to increasing a large integer until overflow. In this method, recursive algorithm is applied. The DMI method steps are as follows:

Step1: The Dynamic Memory (DM) is divided into a continuous sub dynamic memory space DM<sub>i</sub> which represents the value stored in the memory of the address in DA+i, the size is 1 byte, and the i represents the offset of the sub memory space in bytes relative to the first address, as an unsigned integer, from 0 to C-1;

Step2: Incremental Step represented as IV is given by the user to obtain the DA of DM, set the Current Incremental Step (CIV) to IV, set the current offset i to 0, and IV is an unsigned integer, and its value is no less than 1 and no more than 255;
Step3: The Current Dynamic Memory Address represented as CA is obtained from the equation of CA=DA+i, in which i is mentioned in Step1, and through CA addressing to obtain the Current Value (CV) stored in the byte, and the value CV is assigned to the Register Value (RV);

Step4: Add the CIV value to the CV value and assign it to CV, judge whether the following conditions are satisfied: the CV is less than the CIV and the memory address offset i is less than C-1, if the content is satisfied, it enters the Step5, and if not, it is ended and exited;

Step5: Sets the increment step size CIV = (CIV + RV) /256, the memory address offset i=i+1, and turns to step2.

Obviously, step 2 to step 5 is a recursive process. The flow chart of DMI method is shown in Figure 2.

3.3. RS method
The random sampling (RS) method first divide the dynamic memory into N parts, then fill in the sub Dynamic Memory (DM) with pseudo random number, and finally generate Binary Sequence (BS). According to the theory of multidimensional random variable distribution, the distribution of the generation result based on this method is uniform distribution. The method steps are as follows:

Step1: The dynamic memory DM is divided into a continuous sub dynamic memory space DM(i) which represents the value stored in the memory of the address in DA+i, the size of 1 byte, and the i represents the offset of the sub memory space in bytes relative to the first address, as an unsigned integer, from 0 to C-1;

Step2: Generating a random number seed;

Step3: Define the counter represented as CT and initialize it to 0;

Step4: Judge whether the counter CT is less than C-1, if it satisfies, turn to Step6, otherwise turn to Step5.

Step5: Use program to produce random number and copy the random number to the sub dynamic memory DM(CT), and add the counter then turn to Step4.

Step6: Output the BS in DM, and the program ends.

The flow chart of RS methods is shown in Figure 3.
The test case samples generated by DMI method are concentrated in a certain area, which is suitable for testing interfaces with sensitive boundaries. Under the premise of not knowing, it is preferable to use RS method to generate test cases.

4. Application examples

4.1. Coding of the IBSP algorithm
IBSP algorithm is realized by C++ language programming. First, a DataBlock class is designed to represent the memory block of the test case. There are two member variables, the character pointer which points to the memory data block and the integer variable representing the size of the data block. Secondly, the ++ symbol for the incremental operation is overloaded; finally, the two friendly function “Traverse()” and “RandomSample()” are defined to complete the traversal and random sampling of the data blocks. It is worth noting that there is a callback function type in Traverse() and RandomSample() parameters, which can return the result of each test by using a user-defined callback function.

4.2. Example
The following simple functional module is tested by IBSP algorithm.

```
4.1. Coding of the IBSP algorithm
IBSP algorithm is realized by C++ language programming. First, a DataBlock class is designed to represent the memory block of the test case. There are two member variables, the character pointer which points to the memory data block and the integer variable representing the size of the data block. Secondly, the ++ symbol for the incremental operation is overloaded; finally, the two friendly function “Traverse()” and “RandomSample()” are defined to complete the traversal and random sampling of the data blocks. It is worth noting that there is a callback function type in Traverse() and RandomSample() parameters, which can return the result of each test by using a user-defined callback function.

4.2. Example
The following simple functional module is tested by IBSP algorithm.

```
int Min(int a, int b)
```

The function of the module is to find a smaller one from two integers. First, the callback function is defined in Table 1.

The callback function saves every test result to the local file. The memory data block is then defined in Table 2.

Next, use the memory increment method, call the function Traverse (db, testAdd) and generated 1000 tests results shown in Figure 4:

```
Table 1. The definition of callback function

| Struct minS |
|-------------|
|             |
| {           |
|     unsigned int a; |
|     unsigned int b; |
| }           |
| void testMin(unsigned char *pD, long long times) |
| {           |
|     addS *pV = (addS*)pD; |
|     *ffile<<(int)pV->a<<(int)pV->b<<endl; |
| }           |
```

```
Table 2. The definition of memory data block

| DataBlock db; |
|---------------|
|               |
| addS* pvv = new addS(); |
| db.BlockPtr = (unsigned char *)pvv; |
| db.Len = sizeof(addS); |
```

RS method can also be used to call RandomSample (db, 1000, testAdd) to generate 1000 test cases, and the generated test results are shown in Figure 5:
From the comparison between the two, it can be seen that the test cases generated based on the RS method have a wider coverage, so the application is more extensive; and the test cases generated based on the DMI method are more sensitive to the boundary, so this method is more suitable for generating test cases near a certain value.

5. Conclusion
In this paper, a novel algorithm for automatic generation of black box test cases referred to as interface byte stream parsing is proposed, and two methods for generating binary sequence named dynamic memory increasing method and random sampling method are given. This method abstracts the input interface parameters of the diversified input module into the number of bytes used, so only the number of bytes can be given to generate the related test cases, which is beneficial to the automatic generation of test cases by programming and can improve the generation efficiency of test cases.

References
[1] Ron Patton 2006, Software Testing (2nd Edition)(Paul Boger): p3
[2] Shurong Liu 2015, Research on Automatic Generation of Test Data Based on Hierarchic Genetic Algorithm (Beijing Institute of Technology)
[3] Mao Chengying, Yu Xinxin and Xue Yunzhi 2014, Algorithm design and empirical analysis for particle swarm optimization-based test data generation, Journal of Computer Research and Development. 51(4): p824-837
[4] Nubia Technology Co., Ltd. A black box test case generation method and system based on tree type (CN201610244017.0)
[5] Harbin Institute of Technology Single Parameter Event Sequence Test Data Generation Method of Embedded Software(CN201610663751.0)
[6] Randal E.Bryant, David R.O’Hallaron 2010, Computer System A Programmer’s Perspective (Second Edition) Beijing (China Machine Press): p1-3