[11] Paul Feautrier, *Parametric Integer Programming*, Laboratoire MASI, Institut Blaise Pascal, Universite de Versailles St-Quentin, 1988.
[12] A.M. Sergienko, *VHDL for calculation devices design*, "Korneichuk", "TID DS" ltd, 2003.
[13] http://llvm.org/
[14] http://gcc.gnu.org/
[15] G. Aigner, A. Diwan, D. Heine, M. Lam, D. Moorey, B. Murphy, C. Sapuntzakis, *The SUIF Program Representation*.
[16] D. Boulytchev, *Processor Architecture Description Language*, Institute for Information Technologies, 2002. http://oops.tepkom.ru/papers.html
[17] Pioneer Chooses Mentor Graphics Catapult C Synthesis Tool for R&D of Digital Signal Processing Applications http://www.embedded-computing.com/news/Contracts/2908

The System for Automated Program Testing

Steinberg B., Alimova E., Baglij A., Morilev R., Nis Z., Petrenko V., Steinberg R.
Southern Federal University, 105, B. Sadovaya st., Rostov-on-Don, 344006, Russia,
mob. phone: +79185090514
E-mail: steinb@ns.math.rsu.ru

Abstract

The paper focuses on automated white-box and black-box testing of programs. The black-box testing tool has automated input data generator and the subsystem of output data comparison. The described tool can be applied to the testing of portability and scalability of parallel programs. The attention is given to aspect of syntax analyzer testing as a part of developed tool for white-box testing. The syntax analyzer testing process, proposed in this paper, is based on all significant language sequences deduction from formal grammar specification.

1. Introduction

The system for automated program testing is presented in this paper.

At present, graphical interface and web-applications [1] automatic testing tools are popular. Paper [2] is dedicated to the problems of complex software systems. Strategies, planning, resources needs for testing complex software are considered also.

The testing tool includes random input data generator, running programs using random input data subsystem, checking results equivalence subsystem and testing results output subsystem. This tool allows testing calculation modules using "black box" approach [3]. Special tools are used to test program using "white box" approach [3]. Those tools allow controlling the coverage of program branches by set of tests. This tools software system could be used if

- testing program has test presented as the pair <input data files set, corresponding output data files set>;
- template program is given (for example, to test parallel program we have template sequential program);
- it is needed to test the program on a different computing systems (for example, 4 nodes cluster and 16 nodes cluster and we need to test scalability and compatibility).

2. Testing using «Black box» approach

This software tool is designed to automate the black box testing of the computational modules that are part of the high-performance software complex. The «black box testing» term means a lack of access to a source code of the tested program, and the lack of understanding of its internal structure.

The test system sequentially executes the tested program with different input data and on different configurations of the computing system. At the end of each run the test system automatically evaluates the success of the test. The report file is generated as a result of the execution of all required tests.

The input data files for the tested programs are generated automatically by their description. The random number generator from the standard C library is used to generate the data.

In this system the following restrictions on the testing program were adopted:

- The program is not interactive, i.e. it does not contain a graphical user interface and does not read data from the console.
- The program works with predefined input and output files in a fixed format.

Test system is configured using a configuration file, which describes the following information:

- Execution configurations of tested programs.
- Input and output data formats of tested programs.
- The test cases.

Execution configuration means a configuration of computational systems, which will run the test program. The test system is able to run the testing program on a multiprocessor system (using OpenMP), and on a cluster of multiprocessor nodes (using MPI + OpenMP).

Input and output data of the tested programs are described in a special format called IODataDescription. This XML-description can be created using any text editor as well as specially developed for this system IODataDesriptionEditor editor. IODataDescription format is used in the IODataTuner library. With the help of this library the test system gets the following features:

- Generating of input files with random data by a format description.
- Checking the correctness (consistency with description) of the input/output data files.
- Checking the equivalence of the two files with the output data in terms of description. For example: if one file contains the real number 3.1415, and the second one - 3.1414, while the description specifies the accuracy of 0.001, these data are considered equal, and the files they contain - as equivalent.

Test case is determined by a combination of the execution configuration, the input data set and the success conditions. The test is considered successful if and only if all its success conditions are met. The test system supports the following conditions:

- The program must return a zero return code.
- After the completion of test program an output files must be saved to disk.
- The output data must be consistent with their description and must be equivalent while running on different configurations.

Automation of testing by the "white box" principle.

Program system, described above, which is intended for automation of program modules testing by the "black box" principle, is extended to the system for testing by the "white box" principle. This extension is based on the internal representation of programs [6], which is used in Open parallelizing system (OPS) [4], [5] and on a work with a program control flow graph. This extension is developed for programs written in C and FORTRAN languages.

Coverage of control flow is used as a criterion of completeness of testing in this work. Registration of program passing through the edges of control flow graph is needed for use of this criterion [7, p258]. Special operators (registrators) may be placed into program locations, which correspond to control flow graph edges, for automation of such registration. Addition of registrators into program text increases program execution time. It is enough to place this registrators not on all edges, but on some subset. Algorithm for finding such minimal subset of edges is described in [8]. This algorithm is implemented as a program in the context of the given project.

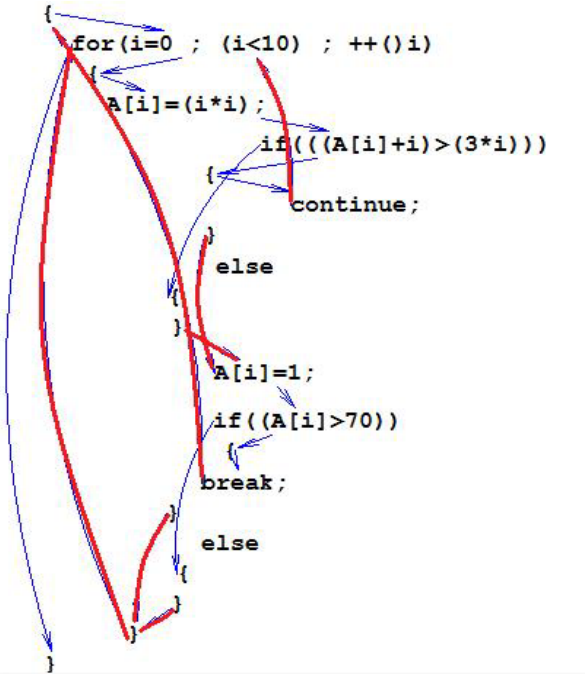

Figure 1. A set of control edges on the Flow Control Graph for the program fragment is highlighted

3. Automatic test generation for syntax analyzer

The automatic tests generator for the parser [9] has been developed. Tests are aimed to prove implementation correctness of following aspects:

- the parser under test accepts all strings of target language (completeness);

- syntax error detection capability (recognition correctness).

The language, which needs to be parsed, represented as a formal grammar description [10] in EBNF (Extended Backus–Naur Form). This description is given to the generator as an input data. Dependency graph of nonterminal symbols (fig.1) is built by formal grammar description automatically. The graph nodes represent nonterminal symbols. The graph edges depict the dependencies of the symbols according the input grammar rules.

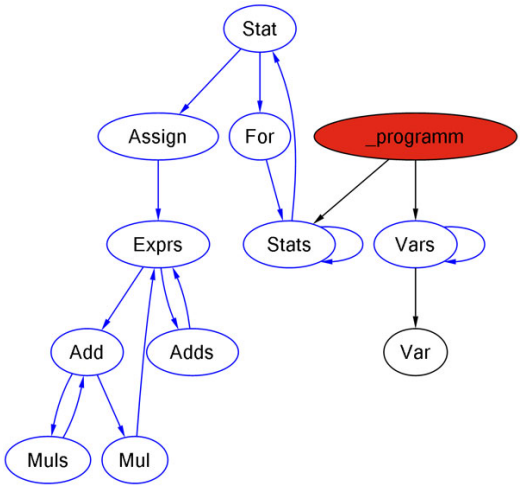

Figure 2. The nonterminal symbols dependency graph (the figure illustrates the simplified formal grammar of C-language)

The set of non-terminal pairs is built as the result of full enumeration process:

$$C = \{(N_i, N_j)\}, i, j = \overline{1,n}, \text{ where } n \text{ is the total amount of nonterminal symbols.} \quad (1)$$

By using the non-terminal dependency graph it is possible to determine whether nonterminal N_j can be deducted from N_i. The deducibility is determined sequential for each pair from set C. If the pair is deductable, the string $s = S \to \ldots \to N_i \to \ldots \to N_j \to \ldots T$, where S – the start symbol, T – terminal symbols, is produced. It is guaranteed that each string s belongs to the target language by using of that algorithm and each nonterminal from set C presents in the set of generated strings.

The test set completeness criteria is occurrence of string derived for each pair of nonterminals in set C.

4. References

[1] P. Mozhaev, "Automatic testing tools", *Open Systems*, 2009, №3.

[2] V. V. Lipaev, *Testing of complex software systems for demands suitability*, IPZ "Globus", Moscow, 2008.

[3] G. Mayers, *Art of software testing*, Finances and Statistics, Moscow, 1982.

[4] Open Paralellizing System. www.ops.rsu.ru

[5] B. J. Steinberg, "Open Parallelizing System 2007 and Interactive Program Parallelizing", *Proceedings of the IEEE EAST-WEST DESIGN & TEST INTERNATIONAL SYMPOSIUM (EWDTS'07)*, Yerevan, Armenia on 7-10 September 2007.

[6] V.V. Petrenko, "Internal representation Reprise for parallelizing system", *IV international conference "Parallel computations and control problems" PACO'2008*, Moscow: October, 2008.

[7] V. A. Evstigneev, V. N. Kasiyanov, *Using graphs in programming: processing, visualization and applying*, "BHV-Petersburg", 2003.

[8] B. J. Steinberg, M.V. Naprasnikova, "Minimal set of control edges while testing program modules", *Isvestia of high-schools. North-Caucasian region. Natural sciences*, №4, 2003, p.15-18.

[9] Alfred V. Aho, Monica S. Lam, Ravi Sethi, Jeffrey D. Ullman, *Compilers: Principles, Techniques, & Tools with Gradiance*, Addison Wesley, USA.

[10] Michael A. Harrison. *Introduction to Formal Language Theory*, Addison-Wesley, USA.