Automatic Test Case Generation for Procedural Languages

Stefan Klikovits[1][2], Paul Burkimsher[1], Manuel Gonzalez-Berges[1], and Didier Buchs[2]

[1] CERN, European Organization for Nuclear Research, Geneva, Switzerland
{stefan.klikovits,paul.burkimsher,manuel.gonzalez}@cern.ch
[2] Université de Genève, Centre Universitaire d'Informatique, Carouge, Switzerland
didier.buchs@unige.ch

Abstract

CERN wishes to augment its test code coverage by applying dynamic symbolic execution tools to legacy code. Many of these tools available have been written for object-oriented languages such as Java or C#. This paper describes an application of automated test case generation to a non-object-oriented scripting language.

1 Problem description

CERN uses *Supervisory Controls And Data Acquisition* (SCADA) software to control its accelerators, experiments and installations. The chosen platform, Siemens' *Simatic WinCC Open Architecture* (WinCC OA), provides a platform for two locally written frameworks which facilitate the creation, operation and maintenance of the final applications.

WinCC OA offers a scripting API, programmable with the proprietary language CTRL which has been inspired by ANSI-C. CTRL developers have historically worked without the support of an automated unit test framework. There is currently a backlog of nearly 500,000 lines of framework code having no automated verification/validation procedures.

This not only engenders an uncertainty about code correctness, but also implies that there is a large overhead when changing execution environments, such as operating system or WinCC OA versions or simply installing patches. In any of these situations the code has to be manually tested to ensure unchanged behaviour – a process absorbing significant developer time and effort.

2 Approach

Due to the size of the code base lacking automated unit tests, we decided to look into automatic test case generation (ATCG). Once generated, a significant proportion of the code can then be checked against previous results automatically and easily. The main disadvantage – our tests look for changed behaviour instead of correctness – is deemed acceptable, as the target frameworks have both reached a stable state after 13 years of continuous use and we presume that all of the serious bugs have been already been identified.

Our initial literature research led us to dynamic symbolic execution tools such as Microsoft Research's *Pex*[6], currently regarded as the most powerful in terms of flexibility, coverage and test suite size[2]. Pex stands out when compared with alternative products in that it supports C#, a language that is very flexible, providing features such as operator overloading and custom casting definitions.

To establish a mapping from CTRL to Pex, a dedicated framework has been built around the latter tool. This system parses the original CTRL input and replaces each functions' dependencies (e.g. global variables or function calls) with additional input parameters. This process is known as *semi-purification* [4]. The resulting dependency-free routines are translated

to C# and ATCG is run on the C# version. To fully support this, it was necessary to translate into C# many of the CTRL standard library functions and to implement the novel CTRL data types therein.

The generated values for the additional parameters are used to specify software mocks[5] for the test case execution.

3 Results

Preliminary results show that the above approach is suited to achieve high code (line) coverage. Pex works well in this regard but we identified certain caveats. Firstly, producing "sensible" input data seems to be a difficult task for the generation tool (Pex). In our case we would like to, for example, produce a list of strings with certain format. Although Pex is capable of doing this, the processing time increases dramatically as a result.

We further applied mutation analysis[3, 1], although the results have not been satisfying so far. An initial evaluation showed that Pex produces the smallest set of inputs to cover the largest number of execution paths without taking boundary values or mutation considerations into account.

4 Future

We are currently in the process of evaluating whether writing our own, domain (CTRL language) specific test case generation tool would be beneficial or feasible.

We further aim to generalise our framework to facilitate ATCG for other C-like languages.

References

[1] Hiralal Agrawal, Richard A. DeMillo, Bob Hathaway, William Hsu, Wynne Hsu, E.W. Krauser, R.J. Martin, Aditya P. Mathur, and Eugene Spafford. Design of Mutant Operators for the C Programming Language. Technical report, Purdue University, March 1989.

[2] Lajos Cseppentő. Comparison of Symbolic Execution Based Test Generation Tools. BSc. Thesis, Budapest University of Technology and Economics, Budapest, 2013.

[3] Richard A. DeMillo, Richard J. Lipton, and Frederick G. Sayward. Hints on test data selection: Help for the practicing programmer. *IEEE Computer*, 11(4):34–41, 1978.

[4] Stefan Klikovits, David P. Y. Lawrence, Manuel Gonzalez-Berges, and Didier Buchs. Considering execution environment resilience: A white-box approach. In *Software Engineering for Resilient Systems - 7th International Workshop, SERENE 2015, Paris, France, September 7-8, 2015. Proceedings*, pages 46–61, 2015.

[5] Gerard Meszaros. *XUnit Test Patterns: Refactoring Test Code*, chapter 23. Test Double Patterns, pages 521–590. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2006.

[6] Microsoft Research. Pex, Automated White box Testing for .NET. http://research.microsoft.com/en-us/projects/pex/.