This article is part of the IEEE Reliability Society 2009 Annual Technology Report.

Function Extraction (FX) Technology: Computing the Behavior of Software

Richard C. Linger
Carnegie Mellon University
Email: rlinger@sei.cmu.edu

The Idea of Software Behavior Computation

Modern society is dependent on software systems of ever-increasing scope and complexity. These systems continue to experience errors and vulnerabilities despite best efforts in their development. Programmers today have no practical means to determine the full functional behavior of software. This technology gap has been the source of problems and frustrations in software for decades.

Software engineers must understand all of the behavior of software, intended or unintended, benign or malicious. While current software development and testing tools can help analyze specific properties and cases of behavior, what is needed is an "*all* cases of behavior" understanding of what software does.

The CERT (Computer Emergency Response Team) organization of the Software Engineering Institute at CMU is developing the emerging technology of function extraction (FX) to automate the calculation of software behavior with mathematical precision to the maximum extent possible [1]. FX is a new type of engineering automation. The objective is to reduce dependence on slow, fallible, manual methods of software analysis by substituting fast, correct computation of behavior. Computing the precise behavior of software requires deriving its net functional effect; that is, how it transforms inputs into outputs in all circumstances of use, without heuristics or approximations. That information can be presented to analysts in non-procedural behavior displays that define all the possible effects a program can have, essentially, the "*all* cases of behavior" view. The ultimate objective is to move from an uncertain understanding of software derived in human time scale to a precise understanding computed in machine time scale. Theory-based function extraction operates on the deep functional semantics of software, and is not subject to the limitations of traditional syntactic methods [2], [3], [4], [5]. Controlled experimentation showed that users of an FX prototype were orders of magnitude faster than users of manual methods in determining software functionality, and were much more productive when computed behavior was available [6].

A Miniature Example of Behavior Computation

In notional illustration of behavior computation, consider the following sequence structure of three instructions (":=" is the assignment operator) operating on integer variables x and y (machine precision aside), and the question of what it does:

```
do
  x := x – y
  y := y + x
  x := y – x
enddo
```

The function extraction process computes a procedure-free expression of what this structure does from beginning to end for all possible values of x and y. For a sequence structure, behavior computation requires composing the statements to determine their net, sequence-free effect. A simple trace table can be used for this purpose, with a row for every assignment, and a column for every data variable assigned. Cells in the table record the effect of the row assignments on the variables. Subscripts are attached to variables to index effects from row to row, with 0 denoting initial values, shown in Table I.

Table I. A Trace Table for Function Composition

Instruction	**x**	**y**
x := x – y	x1 = x0 – y0	y1 = y0
y := y + x	x2 = x1	y2 = y1 + x1
x := y – x	x3 = y2 – x2	y3 = y2

The derivations below express final values in the table in terms of initial values through algebraic substitution.

x3 = y2 – x2
= y1 + x1 – x1
= y1
= y0

y3 = y2
= y1 + x1
= y0 + x0 – y0
= x0

Thus, the computed final values reveal that the control structure exchanges the initial values of x, and y. This is its behavior signature, which can be written as a concurrent assignment where initial values on the right are simultaneously assigned to final values on the left.

x := y
y := x

To envision what is involved in computing behavior for real programs, scale this example up several orders of magnitude, and add substantial mathematical capabilities for function composition and analysis developed over several years of research and development.

FX Application

As a first application, CERT is developing a function extraction system that computes the behavior of programs written in or compiled into Intel assembly language, with a principal focus on malware analysis [7]. This system contains no heuristics or approximations, and accounts for machine precision in its behavior computations. In its current state of development, the system transforms input spaghetti-logic code into structured form, computes its behavior, and displays it to users. Methods for loop behavior computation have been created that can make the effect of theoretical limitations arbitrarily small. The system can demonstrate computation of the behavior of a virus in native, obfuscated, and hidden forms. FX technology has been used to crack virtualized malware that had resisted previous attempts. The technology can be applied to other languages, and can provide automated support for software development and testing [8], correctness verification, embedded system validation [9], and analysis of security properties [10]. FX behavior databases can provide input for other value-add applications. FX can have potentially transformational impact on software development, use, and management. Development of the technology is well along, but is not completed. Sponsors are welcome to participate in targeting FX development to applications of interest.

References

[1] R. Linger, M. Pleszkoch, L. Burns, A. Hevner, and G. Walton, "Next-Generation Software Engineering: Function Extraction for Computation of Software Behavior," *Proceedings of the 40th Annual Hawaii International Conference on System Sciences (HICSS40)*, Hawaii, IEEE Computer Society Press, Los Alamitos, CA, 2007

[2] Lloyd Allison, *A Practical Introduction to Denotational Semantics*, Cambridge Computer Science Texts 23, Cambridge University Press, 1986.
[3] Raymond Smullyan, *Recursion Theory for Metamathematics*, Oxford Logic Guides 22, Oxford University Press, 1993.
[4] S. Prowell, C. Trammell, R. Linger, and J. Poore, *Cleanroom Software Engineering: Technology and Practice.* Addison Wesley, 1999.
[5] H. Mills, and R. Linger, "Cleanroom Software Engineering," *Encyclopedia of Software Engineering, 2nd ed.* (J. Marciniak, ed.). John Wiley & Sons, 2002.
[6] R. Collins, G. Walton, A. Hevner, and R. Linger, *The CERT Function Extraction Experiment: Quantifying FX Impact on Software Comprehension and Verification* (CMU/SEI-2005-TN-047). Software Engineering Institute, Carnegie Mellon University, 2005.
[7] M. Pleszkoch, and R. Linger, "Improving Network System Security with Function Extraction Technology for Automated Calculation of Program Behavior," *Proceedings of the 37th Hawaii International Conference on System Sciences* (HICSS-37), Waikoloa, HI, Jan. 5-8, 2004. IEEE Computer Society Press, 2004.
[8] R. Linger, M. Pleszkoch, and R. Hevner, "Introducing Function Extraction into Software Testing," *The Data Base for Advances in Information Systems: Special Issue on Software Systems Testing,* ACM SIGMIS, New York, NY, 2008.
[9] R. Bartholomew, L. Burns, T. Daly, R. Linger, and S. Prowell, "Function Extraction: Automated Behavior Computation for Aerospace Software Verification and Certification," *Proceedings of 2007 AIAA Aerospace Conference*, Monterey, CA, Vol. 3, May, 2007.
[10] G. Walton, T. Longstaff, and R. Linger, *Technology Foundations for Computational Evaluation of Security Attributes,* Technical Report CMU/SEI-2006-TR-021, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 2006.