Function Extraction (FX) Technology: Automated Calculation of Program Behavior for High-Assurance Systems

Richard C. Linger
CERT Research Center
Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA
rlinger@sei.cmu.edu

Mark G. Pleszkoch
IBM Global Services &
CERT Research Center
Software Engineering Institute
Carnegie Mellon University
markp@us.ibm.com

Abstract

The function-theoretic view of programs suggests the possibility of automated calculation of program behavior. While significant theoretical challenges exist, the value of behavior calculation for high-assurance systems could be substantial.

1. Program Behavior Realities

Short of an impractical expenditure of time and effort, programmers have no means to determine the full functional behavior of programs. Even then, human fallibility in reasoning about myriad program details can cast doubt on the analysis. Despite best efforts, programs are routinely fielded with unknown behavior that may embody errors, vulnerabilities, or malicious code.

The totality of large program behavior is difficult to understand because it is distributed across a virtually infinite number of possible execution paths. Testing selects paths from this set and so cannot reveal full behavior. However, large programs are at the same time composed of a finite number of control structures, each of which makes a finite contribution to overall behavior. This observation motivates a closer look at the possibility of behavior calculation for high-assurance systems, and the theoretical and engineering challenges it presents.

2. Treating Programs Like Equations

The function-theoretic view focuses not on program paths, but rather on control structures and mathematical foundations for their refinement, abstraction, and verification [2]. In this view, control structures are treated as rules for mathematical functions or relations, that is, mappings from domains to ranges, no matter what subject matter they may address. Function-theoretic foundations prescribe procedure-free equations that define the net effects on data of common control structures and provide a starting point for behavior extraction [1,3]. These equations are expressed in terms of function composition, case analysis, and, for iteration structures, a recursive expression based on an equivalence of iteration and alternation structures. Representative equations are given below for control structures labeled P, data operations g and h, predicate p, and program function f.

The program function of a sequence control structure (P: g; h) can be given by

$$f = [P] = [g; h] = [h] \circ [g]$$

where square brackets denote the program function and "o" denotes the composition operator. That is, the program function of a sequence can be calculated by ordinary function composition of its constituent parts. The program function of an alternation control structure (P: if p then g else h endif) can be given by

$$\begin{aligned} f &= [P] = [\text{if } p \text{ then } g \text{ else } h \text{ endif}] \\ &= ([p] = \text{true} \rightarrow [g] \mid [p] = \text{false} \rightarrow [h]) \end{aligned}$$

where | is the "or" symbol. That is, the program function is given by a case analysis of the true and false branches, and the possibility of abstracting them to a single case. The program function of a terminating iteration control structure (P: while p do g enddo) can be expressed as

$$\begin{aligned} f &= [P] = [\text{while } p \text{ do } g \text{ enddo}] \\ &= [\text{if } p \text{ then } g; \text{ while } p \text{ do } g \text{ enddo endif}] \\ &= [\text{if } p \text{ then } g; f \text{ endif}] \end{aligned}$$

and f must therefore satisfy

$$f = ([p] = \text{true} \rightarrow [f] \circ [g] \mid [p] = \text{false} \rightarrow I)$$

These equations define an algebra of functions that can be applied bottom up to the control structure hierarchy of a program in a stepwise manner. The process begins by extracting behavior of the lowest level control structures, thereby revealing the next level for extraction, etc., continuing until the behavior of the entire program has been extracted and expressed in non-procedural form. The process propagates and preserves the net effect of

Proceedings of the Eighth IEEE International Symposium on High Assurance Systems Engineering (HASE'04)
1530-2059/04 $20.00 © 2004 IEEE

control structures through successive levels of abstraction while leaving behind complexities of local computations and data not required to express behavior at higher levels.

3. Strategies for Behavior Computation

If behavior calculation was easy, automated function extractors would be commonplace. Theoretical and engineering challenges include the following areas.

Loop behavior: No general mathematical theory for loop behavior extraction can exist. Because loops can compute arbitrary partial recursive functions, many results from computability theory stand in the way. For example, the undecidability of the Halting Problem [4] means some terminating loops cannot be detected as terminating. The solution strategy includes use of recursive expressions to represent loop operations as a starting point for behavior extraction, and application of canonical patterns and behavior templates for recognition and classification. Initial analysis suggests a surprisingly small number of patterns can cover a significant set of loop structures. Limitations at the mathematical level can often be effectively dealt with at the engineering level, and that appears to be the case for calculating loop behavior.

Indirect data references: Many languages permit indirect references to data through pointers and pointer manipulation, thereby adding complexity to the semantics of behavior calculation. For example, because objects are allocated on the heap in Java, the problem of aliasing can be significant. The solution strategy involves representing pointer-referenced data in generalized storage maps to systematize data access in a common framework as a starting point for analysis. Substantial effort will be needed to define required data semantics and extraction rules, but the task is proportional to the data access facilities defined by the target language.

Scale-up: Rapid human understanding requires that extracted behavior expressions be limited in complexity. As noted above, much complexity reduction is intrinsic to function-theoretic behavior extraction, in the elimination of local processing details while preserving and propagating their net effects. Additional simplification can be achieved through unification and reduction of cases in behavior expressions, and through human factors engineering for effective display of extracted behavior. Introduction of definitions to represent recurring units of behavior is also a key strategy. Such use of definitions has a long history in mathematics to render theorems and proofs more understandable.

Additional challenges include *concurrency* and *real-time* behavior.

It is important to note that the language employed to express extracted behavior need not itself be executable. Instead, its objective is to enable rapid understanding by humans. The language is constrained only by a requirement for mathematical correctness, and can include facilities for first order logic, opportunistic definitions, and other structures supportive of human reasoning. Furthermore, behavior catalogs produced by the extraction process can be expressed in sets of disjoint cases to localize analysis, and can be presented in graphical form for rapid parsing by human eyes and minds.

Automated behavior calculation can thus capitalize on three significant factors:

- the finite population of control structures in a program guarantees termination of the extraction process,
- the elimination of local details during extraction provides intrinsic expression simplification, and,
- the absence of an executability requirement for the behavior expression language permits use of logical structures that facilitate human comprehension.

4. Behavior Extraction Engines

The possibility of behavior calculation could lead to new types of automation with substantial impact on high-assurance system development. Automated tools would be driven by semantics of the target language and rule sets for behavior extraction and simplification. Such tools could reveal unknown program behavior that may harbor errors, vulnerabilities, and malicious code, and help ensure the correctness of intended functionality.

Initial investigations in behavior calculation have produced several papers including [1] that discuss the underlying technology. An executing pre-prototype has been developed that computes behavior for a small subset of Java. Future plans include development of a full prototype that demonstrates solutions to the technical challenges of behavior computation.

5. References

[1] M. Pleszkoch and R. Linger, "Improving Network System Security with Function Extraction Technology for Automated Calculation of Program Behavior," *Proceedings of 37th Hawaii International Conference on System Sciences,* Hawaii, January, 2004, IEEE Computer Society Press, Los Alamitos, CA, 2004.

[2] S. Prowell, C. Trammell, R. Linger, and J. Poore, *Cleanroom Software Engineering: Technology and Practice,* Addison Wesley, Reading, MA, 1999.

[3] P. Hausler, M. Pleszkoch, R. Linger, and A. Hevner, "Using Function Abstraction to Understand Program Behavior." *IEEE Software,* 7, 1, IEEE Computer Society Press, Los Alamitos CA, January, 1990.

[4] M. Davis, *Computability and Unsolvability,* Dover Publications, New York, 1982.

1530-2059/04 $20.00 © 2004 IEEE