WE-13 (Panel): Formal Methods: Fact vs Fiction

Chair

C. Michael Holloway
NASA Langley Research Center

Panelists

Ben Di Vito, *ViGYAN, Inc.*
David Guaspari, *Odyssey Research Associates*
Michael Smith, *Computational Logic*

Panel: Formal Methods Fact vs. Fiction

Panel Chair:
Michael Holloway, *NASA Langley Research Center*

Panelists:
Ben Di Vito, *ViGYAN, Inc.*
David Guaspari, *Odyssey Research Associates*
Michael Smith, *Computational Logic, Inc.*

Summary

Few people seem to be ambivalent about formal methods. Proponents are ardent in their support; opponents are equally ardent in their disdain. Over-zealous supporters make unsupportable claims about the potential of formal methods, while over-zealous detractors make unsupportable claims about its shortcomings. A primary cause of this polarization is that it is often difficult to separate fact from fiction in formal methods. This panel plans to make that separation.

In particular, the panel will address each of the following fictitious statements about formal methods:

1. Formal methods requires lots of education and training
2. Formal methods is equivalent to theorem proving
3. Formal methods is too expensive to use in practice
4. Formal methods does not scale to real problems
5. Formal methods guarantees perfection
6. Formal methods is a straight jacket
7. Formal methods is an all-or-nothing approach
8. Formal methods is simply code-level proofs
9. Ada is too complex and ambiguous semantically to be compatible with formal methods.

Each panelist will make a 15-minute opening presentation in which he will address several of the above statements. The remainder of the session will be used for interaction between the audience and the panelists.

The Panelists

Ben Di Vito is a Senior Research Scientist for ViGYAN, Inc. He currently conducts research at NASA Langley Research Center on applications of formal methods to fault-tolerant computing and flight control system reliability. He is also contributing to the analysis and design of Langley's Reliable Computing Platform (RCP), and is participating in a pilot project to evaluate formal methods for NASA space applications as part of a multi-center effort involving Langley Research Center, Johnson Space Center, and the Jet Propulsion Lab. Prior to joining ViGYAN, he worked for TRW Inc., in a variety of roles involving computer security and formal methods technologies, including a key role in the Army Secure Operating System (ASOS), an Ada-based operating system designed to meet A1 security criteria. He has a Ph.D. in Computer Science from the University of Texas at Austin, and M.S.E. and B.S.E. degrees in Computer Engineering from the University of Michigan. Mr. Di Vito may be contacted through electronic mail at the following address: b.l.divito@larc.nasa.gov.

David Guaspari is employed by Odyssey Research Associates (ORA) in Ithaca, New York. His areas of expertise include: mathematical logic, formal specification and verification of software, and Ada. He has been involved in the design and implementation of the Penelope programming environment for specification and formal verification of Ada programs and of the Larch/Ada specification language, and he was the original leader of the Ada 9X Language Precision Project (sponsored by the Ada 9X Project Office), providing advice to the Mapping/Revision Team based on mathematical analysis of proposed language changes. He has a Ph.D. in Mathematics from the University of Cambridge, and a B.S. in Mathematics from Rensselaer Polytechnic Institute. Mr. Guaspari's electronic mail address is davidg@oracorp.com.

Michael Smith is the Executive Vice-President of Computational Logic, Inc. in Austin, Texas. He was one of the five founders of CLI. He directed CLI's efforts as part of the Ada 9X Language Precision Team (ORA was prime), which analyzed selected Ada 9X mapping proposals and attempted to provide a basis for future mathematical work on the semantics of Ada. He is currently engaged in an effort to prove properties of Ada programs by direct application of an operational semantic definition of a subset called AVA (A Verifiable Ada) that has been specified in the ACL2 Logic. The result of this work should be a library of theorems and an assortment of tools to assist in the proof process. He holds a Ph.D. in Computer Science from the University of Texas at Austin, and a B.S.E. in Electrical Engineering from Princeton University. Mr. Smith's electronic mail address is mksmith@cli.com.

The Chairman

Michael Holloway is a research engineer at the NASA Langley Research Center. He is a member of the formal methods team, a participant in the design and analysis of the RCP, and the designer and maintainer of Langley's World Wide Web homepage for formal methods (URL: http://shemesh.larc.nasa.gov/fm-top.html). His electronic mail address is c.m.holloway@larc.nasa.gov.

Panel: Formal Methods Fact vs. Fiction

Position Statements

Ben Di Vito, ViGYAN, Inc.

In my presentation, I will address directly the following three myths about formal methods:

1. Formal methods requires lots of education and training. (myth 1 in the panel summary list)
2. Formal methods is too expensive to use. (3)
3. Formal methods does not scale to real problems. (4)

Although many researchers in the field of formal methods have extensive backgrounds in logic and mathematics, the expertise needed to practice formal methods is considerably more modest. The core concepts of elementary logic and discrete mathematics needed by practitioners, along with an introduction to a formal specification language and its use, have been taught routinely in one-week training courses. Broader exposure can be obtained through one-semester college courses or their equivalents. Repeated industry experience has shown that people with conventional software backgrounds and skills can be taught basic formal methods quite readily. What has been lacking is enough corporate commitment to allow these people to apply their newly acquired skills.

The attitude that formal methods is too expensive to use stems from the narrow interpretation of formal methods as a complete application of code-level verification. When viewed as a more flexible analysis technique, usable at different points in the life cycle and with varying degrees of coverage, the cost issue ceases to be forebidding. Experience with trusted system development efforts yielded costs in the range of 10-20% for "A1" systems. The NIST survey of applications by Craigen, Gerhart, and Ralston likewise indicates a range of positive findings. While everyone would like to have better cost data, the evidence we have so far does not support the feared unaffordability of formal methods. Moreover, as progress is made and we develop a sizable body of reusable "deductive assets," costs will drop considerably.

The attitude that formal methods does not scale also stems from the interpretation of formal methods as merely code-level verification. The types of real systems to which formal methods have been applied, in one form or another, include: operating systems, networks and their supporting software, real-time embedded controllers, microprocessors and other hardware subsystems, and special-purpose security devices. Many of these applications are chronicled in the NIST survey. Pilot projects for aerospace applications now underway include requirements analysis for the Space Shuttle and other space programs as well as critical subsystems for commercial aviation. What is clear from these enterprises is that by taking an open-minded and adaptable approach to formal methods, many real problems can benefit from the increased modeling and analysis capabilities.

In addition to addressing these two false impressions of formal methods, I would also like to address one or more of the following issues:

- The airborne software guideline DO-178B includes a provision for the use of formal methods in systems requiring FAA certification.
- The Ada 9X Safety and Security Annex may be the crossroads where the Ada, formal methods, and application communities (such as DO-178B) have their "harmonic convergence."
- The best opportunity for synergy is the problem of software reuse. Formal methods can enhance the value of reusable software, and reuse technology can be used by formal methods people in their own work.

David Guaspari, Odyssey Research Associates

In my presentation, I will mainly address the following three misunderstandings:

1. Formal methods is equivalent to theorem proving. (2)
2. Formal methods is an all-or-nothing approach. (7)
3. Ada is too complex and ambiguous semantically to be compatible with formal methods. (9)

"Using formal methods" is not equivalent to theorem proving. "Using formal methods" means no more or less than thinking about a problem in a mathematical way (that's the "formal" part) and expressing the results in a way that's readily communicable to and usable by other people (that's the "methods" part). Shift-reduce parsing, rate-monotonic scheduling, etc., are formal methods, but aren't called that because the "formal methods" label is usually removed from the mathematics once it is put into common use.

Formal methods is not an all-or-nothing matter. Formal techniques can be used for stating requirements, analyzing requirements, specifying code, analyzing the conformance of source code to specs, analyzing translation of source to object, analyzing the hardware on which the object code runs, etc. Any of these can be worthwhile even in the absence of the others. Our own experience is that writing formal specifications is an effective way of finding immediate problems and forestalling long-term problems with

maintenance, etc. We believe that we are more successful at finding coding errors by walking through implementations with the formal specifications in hand.

Finally, a large and useful subset of Ada can be specified and reasoned about formally. The most complex parts of Ada are static semantics, tasking, and optimization (section 11.6). We can claim to have proved that the canonical dynamic semantics (i.e., no 11.6) of sequential Ada is clean -- namely, by producing a sophisticated, but relatively short, semantic definition that can actually be used to reason about programs. Furthermore, such complexities as **are** introduced by 11.6 are not intrinsic to Ada -- they're present, and swept under the rug, in all its obvious competitors.

Michael Smith, Computational Logic, Inc.

In my presentation, I will address directly the following three myths about formal methods:

1. Formal methods guarantees perfection. (5)
2. Formal methods is a straight jacket. (6)
3. Formal methods is simply code-level proofs. (8)

The assertion that formal methods claims to guarantee perfection has been used repeatedly in an attempt to discredit formal methods. See in particular James Fetzer's article "Program Verification: the Very Idea," (*Communications of the ACM*, September 1988, pp. 1048-1063) and the responses to it. The problem with this assertion is that it is clearly false and has been loudly asserted to be false by the community developing formal techniques. Consider a statement that I consider completely synonymous to the above:

Mathematical modeling guarantees perfection.

As engineers we use mathematical techniques to provide **increased** assurance, not **absolute** assurance. Logic is just another branch of mathematics that can be applied in a variety of ways to the development of both software and hardware systems. It enhances our ability to predict the behavior of systems before they are fielded. But ultimately these systems have to face the real world for which our mathematics is only an approximation.

The claims that formal methods is a straight jacket and that formal methods is simply code-level proofs are related. Formal methods provides a smorgasbord of techniques applicable in various ways to the entire software life-cycle. Formal methods can be used for stating requirements, analyzing requirements, specifying code, checking that code specs conform to requirements, analyzing the conformance of source code to specs, analyzing translation of source to object, analyzing the compiled object code, analyzing the hardware on which the object code runs, and more. This broad array of possibilities is just the opposite of a straight jacket.

Mathematical logics necessarily constrain descriptions. If what you need to assert is very difficult to express in a particular logic then you need to consider the possibility that (a) you should shop around for a formalism more appropriate to your task or (b) the mathematical underpinnings for your application are not well enough developed for logical reasoning to be applicable.

If formal methods meant only code-level proofs then its adoption might be considered a straight jacket. Because of the informal mathematical reasoning and testing that good programmers apply to code development, they produce correct software modules on a regular basis. The additional overhead of code proofs could add substantial cost to such modules (using current technology). It might also be very difficult, for example in the case of a program that used floating point. The justification for code proofs on small modules is identical to the justification for any technique: does the assurance required for this application justify the cost of the analysis?