Formal methods in education and industry

John Harrison
Intel Corporation

FM Outreach Meeting
SRI, Menlo Park

Tue 10th June 2008 (09:30 – 10:00)

Two different goals

Here is the stated goal of this meeting:

> "The goal of the meeting is to develop strategies for the creative use of formal modeling and analysis tools in education and industry."

Should we consider the *education* and *industry* parts separately?

Two different goals: conflict?

Maybe the *education* and *industry* parts are antagonistic:

- If something is common practice in industry, why should universities and funding agencies subsidize it? And if it's a solved problem, why should students want to research it?
- If formal methods are part of people's training but not of much immediate use, won't they be considered as impractical ivory tower stuff?

I hope this isn't true, but it could explain the paradox noted by Bob Constable.

Two different goals: synergy?

Maybe the *education* and *industry* parts reinforce each other

- If industry is interested in formal methods, students and universities have financial incentives to focus on the area
- If typical computer scientists / engineers have formal methods in their toolkit, they may be inspired to apply them in their jobs

This is what I hope/believe.

Industry needs formal methods

Industry can certainly provide plenty of motivation for the use of formal methods:

- Famous disasters like FDIV and Ariane 5 have cost companies and governments huge amounts of money
- Costs of validation and quality control starting to exceed costs of design, and hold back progress
- Increasing interest in parallel programming makes traditional debugging even more unsatisfactory

Ideas for education

- Emphasize logical structure of arguments at an early stage
- Teach and use logical notation where beneficial
- Emphasize computational methods
- Use more tools, especially theorem provers, and improve them

Ideas for industry

- Consider gradual or partial use of formal methods
- Strive for more precise specifications of basic interfaces
- Improve tools, especially theorem provers

Emphasize logical structure of arguments

This is often considered “too difficult” and teachers are encouraged to de-emphasize proof. But I believe many students find mathematics difficult precisely *because* the underlying logical principles are never made clear, and resent not being told the whole story.

$$3x + 1 = x + 5$$

$$2x + 1 = 5$$

$$2x = 5 - 1$$

$$2x = 4$$

$$x = 2$$

Mathematics students can get confused over what steps are permissible, nature of implicit quantification, what is a definition, axiom, etc. (“Suppose x is . . . ”).

Teach and use logical notation

Dijkstra has said that ‘as far as the mathematical community is concerned George Boole has lived in vain’.

Symbolism in mathematics is used incessantly, but logical symbolism is only used in an incomplete ad hoc way. But sometimes it would be clarifying.

Mathematicians sometimes express in a tortuous way using words or arithmetization things that could be said very directly and briefly using logical symbolism.

Emphasize computational methods

From Knuth:

> For three years I taught a sophomore course in abstract algebra for mathematics majors at Caltech, and the most difficult topic was always the study of “Jordan canonical forms” for matrices. The third year I tried a new approach, by looking at the subject algorithmically, and suddenly it became quite clear. The same thing happened with the discussion of finite groups defined by generators and relations, and in another course with the reduction theory of binary quadratic forms. By presenting the subject in terms of algorithms, the purpose and meaning of the mathematical theorems became transparent.

Emphasizes problem-solving roots of subjects.

Use and improve tools

CASs and theorem proving tools can help students experiment and explore to deepen their understanding. Moreover (Szczerba):

> There occurred a substantial change in my role as a teacher. Earlier, when assigning homework tests, I was treated as an enemy who has to be forced to accept the solution, sometimes in not an exactly honest way. Now the enemy to be defeated was the computer and I was turned into an ally helping to fight this horrible device. This small fact has seriously influenced my contacts with the students. They were much more eager to approach me with their problems, to report on their difficulties, and ask for help.

However, theorem-proving tools need to improve substantially before they can be routinely used in mathematics educations.

Gradualism: a spectrum of formal methods

There are various possible levels of rigor in correctness proofs:

- Programming language typechecking
- Lint-like static checks (uninitialized variables . . .)
- Checking of loop invariants and other annotations
- Complete functional verification

Exclusive emphasis on the last is a worthy goal, but may be better to focus on ‘weak’ results on large practical systems, e.g.

- Analysis of device drivers at Microsoft
- Static analysis of Airbus A380 flight control software

Strive for more precise interfaces

Computing should be essentially a formal science. But key interfaces at all levels are so imprecise that it's hard even to say what is a bug and whose fault it is:

- The Verilog hardware description language has a "Standard" that is often very hard to understand and leaves many quite basic points unclear.
- The semantics of the basic ISAs (instruction set architectures) are usually not specified precisely, and programmers don't know what they can rely on.
- Most programming languages and libraries lack a remotely precise semantics, and even those with a "formal semantics" leave basic things unsaid (e.g. $1 + 1$ in SML).

Improve tools, especially theorem provers

Even model checkers are considered hard to use for many programmers and engineers, and theorem provers and similar verification tools completely out of range.

In order to make the use of such tools more practical, we have to make them more powerful and more user-friendly.

For full functional verificiation, we also need more integrated verification tools and more formalization of background theories.

I believe that a lack of such general frameworks held back program verification in the 1970s and is still a handicap today for otherwise powerful systems like KIV.

But how to improve theorem provers?

- Ideas from computer algebra, links to actual CASs
- New decision procedures, better combined decision procedures
- Better interoperability, ability to import and export proofs/theories
- Better input syntax, but perhaps not *too* much like NL

Conclusions

It’s possible that progress in education and industry may be antagonistic, but I expect that for the most part they will be mutually reinforcing.

Many natural prescriptions may improve the situation in both cases.

Some common themes, particularly the development of improved theorem provers.