

Formal Methods Outreach

# Formal Methods Outreach: Challenges and Opportunities

N. Shankar

shankar@csl.sri.com

URL: <http://www.csl.sri.com/~shankar/>

Computer Science Laboratory

SRI International

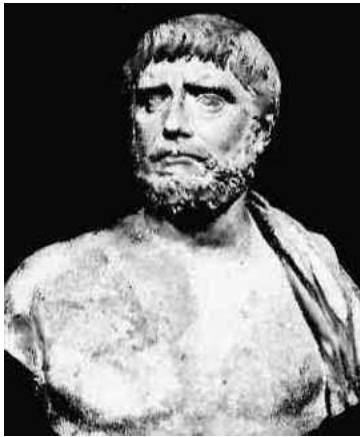
Menlo Park, CA

## Overview

- Formal methods for modelling and analysis are at a crossroads
- We have the core technologies needed to contemplate a wide range of exciting applications.
- These applications should foster computational thinking.
- Potential application areas include education and industry.
- However, we need to take a long-term, strategic view of achieving wider adoption.

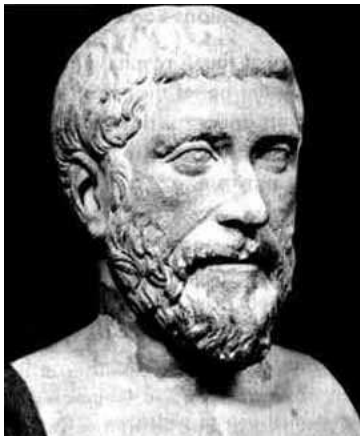
## In the Beginning

Thales of Miletus (624 – 547 B.C.): Earliest known person to be credited with theorems and proofs.



*It was Thales who first conceived the principle of explaining the multitude of phenomena by a small number of hypotheses for all the various manifestations of matter.*

Pythagoras of Samos (569–475 B.C.): Systematic study of mathematics for its own sake.



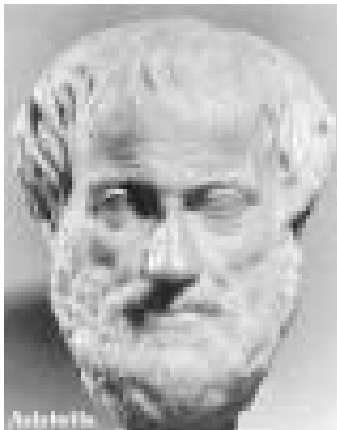
*... he tried to use his symbolic method of teaching which was similar in all respects to the lessons he had learnt in Egypt. The Samians were not very keen on this method and treated him in a rude and improper manner.*

## The Axiomatic Method



Plato (427–347 B.C.): Suggested the idea of a single axiom system for all knowledge.

*the reality which scientific thought is seeking must be expressible in mathematical terms, mathematics being the most precise and definite kind of thinking of which we are capable.*



Aristotle (384–322 B.C.) of Stagira: Laid the foundation for scientific thought by proposing that all theoretical disciplines must be based on axiomatic principles.

# The Elements

Euclid of Alexandria (325–265 B.C.): Systematic compilation and exposition of geometry and number theory.



1. A straight line segment can be drawn joining any two points.
2. Any straight line segment can be extended indefinitely in a straight line.
3. Given any straight line segment, a circle can be drawn having the segment as radius and one endpoint as center.
4. All right angles are congruent.
5. If two lines are drawn which intersect a third in such a way that the sum of the inner angles on one side is less than two right angles, then the two lines inevitably must intersect each other on that side if extended far enough. This postulate is equivalent to what is known as the parallel postulate.

## A Glimmer of Rationality



**Ramon Llull (1235–1316)**: Talked of *reducing all knowledge to first principles*. Developed a symbolic notation (Ars Magna) and conceived of a reasoning machine.

*When he attempted to apply rational thinking to religion, Pope Gregor XI "accused him of confusing faith with reason and condemned his teachings."*

**Gottfried Leibniz (1646–1716)** The idea of a formal language (*characteristica universalis*) for expressing scholarly knowledge and a mechanical method for making deductions (*calculus ratiocinator*).



*What must be achieved is in fact this: that every paralogism be recognized as an error of calculation, and every sophism when expressed in this new kind of notation, appear as a solecism or barbarism, to be corrected easily by the laws of this philosophical grammar.*

*Once this is done, then when a controversy arises, disputation will no more be needed between two philosophers than between two computers. It will suffice that, pen in hand, they sit down to their abacus and (calling in a friend, if they so wish) say to each other: let us calculate.*

## Why Automate Proof?

Vannevar Bush (<http://www.theatlantic.com/unbound/flashbks/computer/bushf.htm>) in a penetratingly prescient 1945 article entitled *As We May Think* wrote:



*Logic can become enormously difficult, and it would undoubtedly be well to produce more assurance in its use. . . . We may some day click off arguments on a machine with the same assurance that we now enter sales on a cash register.*

## Peirce Against Applied Logic



Charles Sanders Peirce (1839–1914), a founder of modern formal logic railed against applications of logic and automated reasoning long before even the possibility existed.

*The aggregate of all applications of logic will not compare with the treasure of the pure theory itself. For when one has surveyed the whole subject, one will see that the theory of logic insofar as we attain to it, is the vision and the attainment of that Reasonableness for the sake of which the Heavens and the Earth have been created.*

## As We May Think

Vannevar Bush, (<http://www.theatlantic.com/unbound/flashbks/computer/bushf.htm>) in a penetratingly prescient 1945 article entitled *As We May Think*, wrote:



*Logic can become enormously difficult, and it would undoubtedly be well to produce more assurance in its use. . . . We may some day click off arguments on a machine with the same assurance that we now enter sales on a cash register.*

## **Are We There Yet?**

Not quite, but within ten to fifteen years, we will be at a point where we can enjoyably, if not effortlessly, click off proofs on a machine.

After decades of intensive work, we now have powerful special-purpose inference algorithms and well-engineered general-purpose theorem provers and proof assistants.

This is supplemented with advances in standardized data formats, program semantics, databases and data mining, machine learning, algorithms, and Moore's law.

This combination of technologies can be used to design and build complex systems, plan and schedule activities, and help extract useful information from volumes of data.

## **Are We Jumping the Gun?**

We do still have a long way to go in developing the science of formalization and inference, and the associated technologies.

But we have crossed a threshold — we can now imagine a future in which information and inference are crucial.

As with the 1956 Dartmouth meeting on Artificial Intelligence, we have an opportunity to gaze deeply into the crystal ball and convene an ambitious but realistic agenda.

## What is Computing?

Computing, like mathematics, is the construction and study of pure abstractions.

Mathematics deals with the laws underlying abstract entities like number, length, area, volume, surfaces, graphs, sets, functions, relations, algebras, and homomorphisms.

Computing, on the other hand, deals with the more prosaic abstractions such as bits, bit-vectors, arrays, tables, stacks, queues, trees, and grammars.

Both Mathematics and Computing are used for modeling, simulating, and analyzing physical systems.

Computers can also model mathematics, and are ultimately realized using the laws of Physics.

## What is the Matter with Computing?

The anticipated 2006–2016 growth rate for computing-related jobs is 30 to 40% (<http://www.bls.gov/oco/ocos042.htm>).

Yet, undergraduate enrollment dropped from 13,900 in 1998 to 7915 in 2007 (<http://www.insidehighered.com/news/2008/03/05/compsci>).

*Between 1984 and 2004, the share of CS bachelors degrees awarded to women fell from 37% to 25%. Taulbee data from 2005 and 2006 suggest that upcoming NSF studies will report that the share of degrees granted to women continued to fall in those years.* (<http://www.cra.org>)

Women are 31% of the Masters degree-winners in CS, and 20% of the PhDs.

## What is a Formal Method?

Formal methods operate at the symbolic level.

Abstract models are used to represent real systems in the form of logical assertions and state machines.

Techniques such as rewriting, constraint solving, model checking, and proof search can be used to find witnesses and proofs.

These tools are used for modeling, constraint solving, state exploration, and verification.

Many of these methods are fast enough that we could put them in toasters and wrist watches.

## **Challenges: Near and Far**

The critical challenge for formal methods include:

1. The specification and verification of complex, software systems.
2. Tools for formalizing and exploring mathematical and scientific knowledge.
3. The design and analysis of complex cyber-physical systems that operate within time, space, and societies.
4. Ultimately, these ideas could also be applied even in poorly formalized domains.
5. Formal modeling, problem solving, and analysis methods should be an integral part of the training of computer scientists.

## What is the Matter with Software?

Software is the ether of the 21st century but it is also symptomatic of a lot of things unreliable.

Nothing you can say about software is universally valid.

Does software really have requirements and design, or do we make these up along the way?

Should software be viewed as a disposable product with planned obsolescence, or should it be *ready on day one*?

Is software a product, or a medium (for more software and services)?

Does open-source software development lead to better, more reliable software?

## Formal Software Verification

Inference tools (model checkers, constraint solvers, static analyzers, and theorem provers) are being used for

1. Building models and formulating requirements
2. Checking requirements for consistency
3. Generating test cases
4. Exploring the state space for properties
5. Verifying hardware and software designs
6. Automatically inferring system properties

## Other Uses of Formal Methods

1. Building formalized libraries of definitions, theorems, and proofs (Mizar, Flyspeck).
2. Generating plans, activities, and schedules.
3. Diagnosing faults.
4. Synthesizing programs.
5. Playing games.

## Some Representative Outreach Projects

*TeachScheme!* from Felliesen and colleagues for teaching program design to high school students and teachers. Experimenting with ACL2 for proofs.

*POPLMark* challenge for verifying programming language metatheory proofs. Landmark C compiler proof by Xavier Leroy and colleagues.

*Team for Research in Ubiquitous Secure Technology* (TRUST) consortium for cyber-security research and outreach.

*ChicTech* at UIUC

*Squeak!* at <http://www.squeakland.org>.

## Formal Methods and Education

Education and technology do not always mix well.

Learning is a social activity, whereas technology can be easily subverted and it fuels isolation.

Still, when we prove theorems or find counterexamples or verify programs with a proof assistant, the gain in knowledge is palpable.

Through such interaction we also gain meta-level knowledge that can be fed back into the mechanization.

Technology can be integrated into a social process so that it bootstraps learning, reinforces knowledge, drills in skills, offers multiple perspectives, and supports collaboration.

We can move from learning by being told to learning from doing.

## Opportunities

There are many grand challenges in building and managing abstractions.

The Verified Software grand challenge: Can we specify and verify software as it is being designed and developed with a small ( $\times 2$ ) overhead?

The QED Challenge: Can we formalize the core of human mathematical knowledge in mechanized form?

The Systems X Challenge: Can we model and analyze scientific phenomena as computational systems?

An Abstraction Library: A personal abstraction library (PAL) that maintains the abstractions and data acquired by a person in a reusable and explainable form.

## **This Meeting**

We have an opportunity for laying out an ambitious agenda to bring the benefits of abstraction to the masses.

Formal methods have come a long way in the development of semantics as well as tools.

The world of computing is threatened by the greater complexity of systems and by threats from sophisticated attacks.

We need a workforce that can make productive use of abstraction in designing and understanding complex systems.

We have speakers who will look at industrial applications, research, and education.

## Conclusions

The great challenge of the twenty-first century is information complexity.

Abstraction is a critical tool for managing complexity.

We have managed quite well for millennia with fairly simple abstractions, but most systems have complexities that cannot be captured by systems of linear equations.

Formal methods offers technologies for developing and manipulating abstract knowledge.

*How can we get more people using these techniques and learning from them?*

## Schedule

Time	June 9	June 10
9AM	Breakfast	Breakfast
9.30	Shankar: Introduction	John Harrison
10	Jay Misra	Pat Lincoln
10.30	Gary Leavens	Cesar Munoz
11	Coffee	Coffee
11.30	Mike Gordon	Dave Naumann
12	Tom Ball	Ashish Tiwari
12.30 – 1.30	Lunch	Lunch
1.30	Kevin Sullivan	
2	Dana Scott	
2.30	Carolyn Talcott	
3	Bob Constable	
3.30	Coffee	
4	Charles Patton	
4.30	Nora Sabelli	
5PM	Michael Beeson	
7PM	Dinner at Sultana's	

## Michael Beeson Statement

I have been involved in two projects that might be considered applications of automated deduction. The first was my software MathXpert, intended to help students of algebra, trigonometry, and calculus. I will report briefly on my experiences since the completion of MathXpert. The second is a project I am currently engaged in, called "Computer Support for Process Engineering". I will describe what process engineering is and how automated deduction can be used, for example, to help build cargo aircraft. I will say a few words about the other players involved in this project and speculate on the future of this type of application.

## Gilles Dowek's Statement

About industrial outreach. We have seen a lot of progress in the last decade. But still the formal methods are limited to very specific niches: transportation (by train, by car or by air, including space), smart-cards and energy production (such as nuclear power-plants). To continue development we could find more niches, such as health (remote surgery, ...). But we could also aim a much larger community by designing less powerful and more accessible tools, for instance we have seen that model checking and static analysis have been more used than formal proofs, because they are more easier to use, although they are less powerful. In this direction, the use of programming languages with dependent types, that allow to express more properties than simple types (such as the absence of array overflow) without requiring too much proofs seem to be in the good direction.

For education, one problem is that teaching formal methods requires to have students that master other topics first such as programming and logic. A good question is that of the possibility to teach formal methods to students who have a low level in logic. Benjamin Werner has taught Coq to engineering students who had no previous exposition to logic and this works (at least for a introductory course, how far these students can go without a better understanding of logic is another question). Benjamin is going to start a similar course in École polytechnique next year. As we already have a course on static analysis and abstract interpretation (by Laurent Mauborgne) and we had a course on model checking (by Jean-Pierre Jouannaud) in the past, all the branches of formal methods can be taught to engineering students (who have no intention to go to graduate school). Yet a fundamental problem we have is that, because of the lack of computer science teaching in high school and because the market

requires a lot of computer scientists, many computer scientists in industry are undereducated, not only in formal methods, but in all areas of computer science. This is probably one reason for the difficulty of advanced computer science (such as formal methods) to have a larger diffusion. If we cannot influence markets a lot we have more possibilities to influence high school curricula.

## Allen van Gelder's Statement

Software verification should start to be introduced into the college computer programming curriculum, not as a topic to study, but as a tool for getting their assigned programs to work.

My model is something like escjava2. Students can learn how to put in the assertions and it can be a grading issue that the assertions check out. I tried a pilot on some graduate students before springing it on undergraduates. We ran into too many problems with escjava2, but this was several years ago. Many problems were related to recursion. As I remember, we could not even state what we wanted to state in some cases, and in other cases it went into a loop until stack space was used up. So I did not take it further.

I think a lot of software companies have the theory that what the students get used to in college transfers to a demand from industry as they graduate into the work force. So they give big discounts to colleges or maybe comp them. I've heard Cadence does this. Anyway, if a good tool is usable at the junior undergraduate level for verification of specifications, pre- and post-conditions, this would evolve to become an industry standard.

## **Gary Leavens' Statement**

### **A Viral, Infectious Approach to Outreach in Computing Education by Gary T. Leavens (UCF)**

Tools that use formal methods (FM) should be part of the experience of all students who graduate with a degree in computing. The main justifications for such an educational objective are the prevalence and cost of bugs and the increasing spread of critical software in society. Or increasing reliance on computer software means that even results in the theory of computing need to be more trustworthy.

However, giving all students experience with formal method tools is difficult, since tools can and should be applied in all areas, and since there are few professors who are formal method specialists. Thus I advocate a viral, infectious approach to educational outreach. In this approach, we build symbiotic relationships with researchers in other areas. The tools and teaching materials we develop for these other areas become the vectors for us to infect other areas with FM ideas, which will realize our objective of giving FM tool experience to all graduates.