The Verifying Compiler: a Grand Challenge for Computing Research

Tony Hoare

Redmond Faculty Summit August 2, 2004

Typical Grand Challenges

Prove Fermat's last theorem	(accomplished)
Put a man on the moon	(accomplished)
Cure cancer within ten years	(failed in 1970s)
Find the Higgs boson	(in progress)
Map the Human Genome	(accomplished)

In Computing Science

Prove that P is not equal to NP	(open)
The Turing test	(outstanding)
A championship chess program	(completed 1997)
The verifying compiler	(in progress)

A Grand Challenge

- May take fifteen years or more to complete
- Attracts world-wide participation
- Has a clear test of success or failure
- Pursues scientific ideals
- Enlarges fundamental understanding
  - pursues scientific ideals
  - in an area of significance to mankind

A Grand Challenge needs

- Maturity of the state of the art
- Support from the scientific community
- Commitment from the teams that engage in it
- Understanding from funding agencies

The Verifying Compiler

A verifying compiler uses automated mathematical and logical reasoning to check the correctness of the programs that it compiles. Correctness is specified by types, assertions, specifications, and other redundant annotations that accompany the code of the program.

Test of success

- Significant software products are analysed mechanically and formally verified,
  - ranging from safety-critical and embedded codes to open source and legacy applications
  - verified at an appropriate level of safety/soundness/security/service.
- Verified programs replace existing versions in use
  - subsequent evolution will maintain correctness.
- Verification is integrated into commercial toolsets

Fundamental understanding

- What is this program for?
  - Its specification tells you its function
- How does it work?
  - Annotation at interfaces explains how.
- Why does it work?
  - The theory of programming explains why.
- Are the answers accurate?
  - A verifying compiler provides a reliable check

Scientific ideals

- The project complements commercially motivated evolution of existing products
  - which follow market demand
  - to discover more faults in existing programs.
  - appeal to current educational level of programmers
  - with many pictures

- But academic research pursues ideals of purity, accuracy, completeness -- and correctness
  - far beyond the current needs of the market place

Beneficial

- The understanding and knowledge gained on completion of the project promises benefit to mankind.

- Reduction in program errors could even now save $22 to $60 billion per year in US (US Dept. Commerce Planning Report 02-03, May 2002).

The team must include …

- Programming theorists
- Programming tool-set builders
- Compiler writers and optimisers
- Design pattern architects
- Sympathetic users to test the assertions
- Open source code contributors
- Proof-tool builders, model checkers,…
- Teachers and students to do the work

Maturity

- The reasons for previous failure to meet the challenge are well understood.
  - And they can be overcome.

- Compared with 1967
  - Gigabytes and Gigacycles are cheap
  - Beneficiaries number in millions
  - The state of the art is much advanced …

State of the art

- Smart-card applications have been manually proved (eg. Logica).
- Safety-critical systems have been developed from specification (eg. Praxis).
- Commodity software already includes many assertions (eg. Microsoft Office)
- Open Source software is freely available for research, as well as for use (eg. Apache).
- Programming theory covers O-O, concurrency (eg. Separation Logic, Process algebra ,…)

Some Available Tools

- Assertion generators
- Program analysers
- Abstract Syntax Tree compilers
- Verification Condition Generators
- Optimisers
- Program Development Environments
- Theorem provers
- Constraint solvers
- Model checkers

An open verification repository

- Collects a range of ‘challenge’ codes
  - ranging from embedded and safety-critical to commodity services and legacy software
- together with its documentation
  - specifications, assertions, design trajectories, test harnesses, test suites, etc.

Tools

- Collects applicable program analysis tools
  - Ensures their full inter-working
  - Advises and assists in their use
  - Facilitates their continuous improvement
  - Integrates them for convenient application
- Transfers technology to commercial vendors.

Standards

- formulates standards for compatibility of analytical programs and the challenge data to which they are applied
  - Negotiates their acceptance by the scientific community
  - Advises and assists in the achievement of the standards
  - Converts data where necessary

IFIP Working Conference?

- Proposal for submission to IFIP TC2
- Verified Software: Theories, Tools and Experiments
- Zurich, Fall 2005
- Chairmen: Jay Misra and Tony Hoare
- Organisers: Bertrand Meyer, Natarajan Shankar, David Gries, Jim Woodcock

A Program Verifier

One can dream of routinely using a verifying compiler as an everyday tool. In the context of this idea our work has been extremely modest and must be considered as a small first step. We only hope that, indeed, this has been a first step of a progression which will allow this dream to come to fruition.

A Program Verifier
Thesis by James C. King
Carnegie Institute of Technology
September 1969