The Verifying Compiler: a Grand Challenge for Computing Research

Tony Hoare

Cambridge 15 July, 2003

Typical Grand Challenges

Prove Fermat's last theorem	(accomplished)
Put a man on the moon	(accomplished)
Cure cancer within ten years	(failed in 1970s)
Map the Human Genome	(accomplished)
Map the Human Proteome	(too difficult now)
Find the Higgs boson	(in progress)
Find Gravity waves	(in progress)
Unify the four forces of Physics	(in progress)
Hilbert's program for math foundations	(abandoned 1930s)

In Computing Science

Prove that P is not equal to NP	(open)
The Turing test	(outstanding)
The verifying compiler	(abandoned in 1970s)
A championship chess program	(completed1997)
A GO program at professional standard	(too hard)
Machine translation English to Russian	(failed in 1960s)

A Grand Challenge needs

- General support from the international scientific community
- Long-term commitment from the teams who 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. The criterion of correctness is specified by types, assertions, and other redundant annotations that are associated with the code of the program.

Its Evaluation

The capabilities of the verifying compiler will be assessed by application to a broad selection of legacy code, chiefly from open sources. This will give confidence that the engineering compromises that are necessary in such an ambitious project have not damaged its ability to deal with real programs written by real programmers.

GC Properties

- Fundamental
- Astonishing
- Testable
- Inspiring
- Understandable
- Useful
- Historical
- International
- Revolutionary
- Research-directed
- Challenging
- Incremental
- Co-operative
- Competitive
- Effective
- Risky

Fundamental

- It arises from scientific curiosity about the foundation, the nature, and the limits of an entire scientific discipline, or a significant branch of it.
- How does it work? Why does it work? Program verification technology gives accurate answers these questions.

Astonishing

- It gives scope for engineering ambition to build something useful that was earlier thought impractical, thus turning science fiction to science fact.
- It is amazing that computers can check the correctness of their own programs, using logical proof just as in mathematics.

Testable

- The project has a clear measure of success or failure at the end; and ideally, at intermediate stages too.
- A verifying compiler will be available for standard software engineering tool-sets, extensively tested on millions of lines of code.

Incremental.

- The project decomposes into identified intermediate research goals, which can be shared among many separate teams over a long time-scale.
- Application of tools to increasing volumes of code will establish correctness at increasing levels of security, integrity and functionality.

Understandable

- The goals are generally comprehensible, and capture the imagination of the general public, as well as the esteem of scientists in other disciplines
- The general public is well aware of the problem of program incorrectness, and should welcome an attempt by scientists to solve a problem of their own creation.

Useful

- The understanding and knowledge gained on completion of the project bring scientific and economic or social benefits.
- Reduction in cost of testing throughout the product life has been estimated at $60bn (US Dept. Commerce).

Historical.

- The prestigious challenges are those which were formulated long ago; without concerted effort, they would be likely to stand for many years to come.
- The challenge goes back to Turing (1948), McCarthy (1962), Floyd (1967).

Feasible.

- The reasons for previous failure to meet the challenge are well understood and there are good reasons to believe that they can now be overcome.
- Everything is better since 1972

Feasible

- Hardware speed and capacity (1000-fold)
- Safety-critical experience
- Open Source ideals
- Many beneficiaries
- Advances in global program analysis
- And in theorem proving technology
- And in formal semantics (object orientation, concurrency).

International.

- It has international scope, including the best research groups in the world. The cost and the prestige of the project is shared among many nations; the benefits are shared among all.
- International collaborations, as in Astronomy and Physics, are now needed in Computer Science.

Research, not Development.

- The project is forwarded by known methods of academic research. It does not duplicate commercially motivated evolution of existing products.
- Commercial software tools discover faults; only academic research pursues ideals of purity, accuracy, correctness.

Effective.

- Its promulgation changes the attitudes and activities of those engaged in it. They promise to do what they would not otherwise be doing.
- Long-term large-scale collaborative projects are rare in computing, and without an idealistic goal will probably remain so.

Toolset Integration.

- Assertion generators (DAIKON)
- Program analysers (PREfix, SPLINT)
- Program optimisers
- Test harnesses (with fault injection)
- Test case generators (from assertions)
- Verification Condition generators (ESC)
- Theorem provers (simplify, HOL)
- Decision procedures (SAT, PVS)
- Model checkers (SPIN, FDR)

Reverse Engineering

- Of libraries
- Of network services
- With guarantees of security
- Elimination of crashes
- Documentation of internal interfaces
- Specification of functional correctness
- Run time assertion testing (by users)
- Projects for Masters' theses, etc.

Risks

- Poor quality of legacy code/languages.
- Errors are just missing preconditions.
- Or needed for functionality/compatibility.
- External interfaces not formalisable.
- Build/configuration files not provable.
- Multiple languages in single application.

A Grand Challenge needs

- General support from the international scientific community
- Long-term commitment from those who engage in it
- Understanding from funding agencies