Using the Computer to Prove the Correctness of Programs

Bengt Nordström

bengt@cs.chalmers.se

ChungAng University

on leave from Chalmers University, Göteborg, Sweden

Important properties of software:

- Number of features
- Performance
- Correctness

History of Program Correctness

- Turing 1949: Checking a Large Routine
- Robert Floyd, 1967: Assigning meanings to programs
- E.W.D. Dijkstra 1968: A Constructive Approach to the Problem of Program Correctness
- Tony Hoare, 1969: An Axiomatic Basis for Computer Programming

Different kinds of languages

- **Tacit:** To communicate with yourself
- **Informal:** To communicate with a human being
- **Formal:** To communicate with a computer

Languages involved in programming

- **Programming Language:** How to compute something?

  Begin with the first page and then continue with the next.

- **Specification Language:** What should the program compute?

  The phone number of a person.

- **Logic:** Why is the program computing what it should?

  All pages are gone through.

Degrees of precision

- **Programming Languages:** formal in academia around 1930, in industry around 1955.
- **Specification languages:** formal in academia around 1967. In industry today: mainly tacit/informal.
- **Logic:** formal in academia around 1967. Now in industry: mainly tacit.

Testing vs Proving

Testing: In testing you check whether the program is correct for a finite numer of inputs. This relies on the assumption that it is easy to check whether a given output is correct.

Proving: In proving we prove that the program is correct for **all** inputs.

Two approaches to proving

- **Automatic Theorem Proving:** Give the program and the property it should have to the computer and wait for an answer:
  - yes, the program is correct
  - no, the program is not correct
  - no reply at all
- **Interactive Proof Checking** The programmer builds interactively a formal proof and the computer checks all steps.

Type Theory

was developed 1970–1980 by Per Martin-Löf as a language for mathematics. His interest was foundational, he wanted to describe the language mathematicians are using in such a detail that nothing is left informal.
In it, words like **proof**, **proposition**, **true**, **equal** have a precise meaning.

a	$\in$	A
element		set
proof		proposition
program		specification

Example of a progamming problem

Let's look at the following example which is a specification of a sorting algorithm.
The problem is to find a function which outputs a sorted permutation of its input.

$$Sort \equiv \Pi x \in List(N).\Sigma y \in List(N)\ .\ Perm(x, y) \wedge Sorted(y)$$

where

$$Perm(x, y) \equiv \forall z \in N.\ \#z in x = \#z in y$$

Proof editors based on type theories:

- **Alf, Alfa, Agda** Chalmers University, Sweden
- **Coq** INRIA Rocquencourt, INRIA Futurs, France
- **Lego** Edinburgh University, UK
- **LF** Edinburgh University, UK
- **ELF** Carnegie Mellon, USA
- **NuPRL** Cornell University, USA

Other proof editors:

- **Isabelle** Cambridge
- **HOL** Cambridge

Important research issues:

- **Correctness of Computer Systems** execution platform for JavaCard, IEEE 754 standard for floatin point operations, correctness of computer algebra algorithms.
- **Proof Technology:** proof libraries, unification, user interfaces, tactic languages, proofs on the web.
- **Formal mathematics:** Extension of computer algebra systems.
- **Mathematics education:** Logic courses, analysis.
- **Foundational research:** extensionality, correctness of languages.