Proving and Programming

Cristian S. Calude
www.cs.auckland.ac.nz/~cristian
(joint work with Elena Calude and Solomon Marcus)

Proving vs programming

There is a strong analogy between proving theorems in mathematics and writing programs in computer science. This paper is devoted to an analysis, from the perspective of this analogy, of proof in mathematics.

We will argue that: 1

- Theorems (in mathematics) correspond to algorithms and not programs (in computer science); algorithms are subject to mathematical proofs (for example for correctness).

We will argue that: 1

- Theorems (in mathematics) correspond to algorithms and not programs (in computer science); algorithms are subject to mathematical proofs (for example for correctness).
- The role of proof in mathematical modelling is very little: adequacy is the main issue.

We will argue that: 1

- Theorems (in mathematics) correspond to algorithms and not programs (in computer science); algorithms are subject to mathematical proofs (for example for correctness).
- The role of proof in mathematical modelling is very little: adequacy is the main issue.
- Programs (in computer science) correspond to mathematical models. They are not subject to proofs, but to an adequacy analysis; in this type of analysis, some proofs may appear. Correctness proofs in computer science (if any) are not cost-effective.

We will argue that: 2

- Rigour in programming is superior to rigour in mathematical proofs.

We will argue that: 2

- Rigour in programming is superior to rigour in mathematical proofs.
- Programming gives mathematics a new form of understanding.

We will argue that:

- Rigour in programming is superior to rigour in mathematical proofs.
- Programming gives mathematics a new form of understanding.
- Although the Hilbertian notion of proof has few chances to change, future proofs will be of various types, will play different roles, and their truth will be checked differently.