The Path to Computer Mathematics

Venanzio Capretta
University of Nijmegen, The Netherlands

Tallin, 19 April 2007

Using Computers in Mathematics:

- Computer Algebra Systems
  (Mathematica, Maple)
  Symbolic Computation
  Inherently Unreliable

- Proof-Assistants (Coq, HOL, Mizar)
  Formalization of Proofs
  Absolute Certainty
  Hard Work (deBruijn factor = 4)

- First-Order Provers (Otter, Vampire)
  Automatically prove logical formulas
  Restricted domain

Ideal Computer Mathematics System:
Develop proofs interactively, using symbolic computation and some automation.

Proof-assistants (Proof-checkers)

Based on different foundations:

- HOL (Higher Order Logic)
- Mizar (Axiomatic Set Theory)
- Coq (Type Theory)

Proof-checkers:
The user inputs a proof
The system checks if it is correct

Proof-assistants:
Help the user to develop the proof
Interactive development
Some automation

The seventeen provers of the world
Freek Wiedijk (ed.), LNAI 3600

Four Color Theorem

Every planar map can be colored with at most 4 colors

- Alfred Kempe (1879):
  Elegant proof
  Considered correct for 10 years
  Incorrect

- Kenneth Appel and Wolfgang Haken (1976)
  Correct proof
  Relied heavily on computer computations
  Should we trust the computations?

- Benjamin Werner and Georges Gonthier (2004)
  Formal proof in Coq
  Computations + Rigorous proofs

Kepler Conjecture

The cubic close packing is the densest way to pack spheres

Thomas Hales (1998) prooved it
Complex computer computations

Panel of 20 referees: 99% sure of correctness
No errors but no certainty of correctness

Hales started Project FlysPecK: formally verify the proof by computer.

Programming with dependent types

Example: List Sorting
Progressive refinements of the type of the program:

$\mathsf{sort} : [\mathbb{N}] \to [\mathbb{N}]$
$\mathsf{sort} : (n : \mathbb{N}, l : \mathbb{N}^n) \to \mathbb{N}^n$
$\mathsf{sort} : (n : \mathbb{N}, l : \mathbb{N}^n) \to \mathsf{Permutation}(l)$
$\mathsf{sort} : (n : \mathbb{N}, l : \mathbb{N}^n) \to \{l' : \mathsf{Permutation}(l) \mid \mathsf{Ordered}(l')\}$

A program with the last type is automatically correct.
This is a way to guarantee correctness.

Relation with logic and mathematics:

$$\forall n : \mathbb{N}.\forall l : \mathbb{N}^n.\exists l' : \mathsf{Permutation}(l).\mathsf{Ordered}(l')$$

From constructive mathematics to type theory

Heyting's explanation of intuitionistic logic (first-order arithmetics):

- Proof of $A \wedge B$:
  pair $\langle a, b \rangle$, a proof of A, b proof of B;

- Proof of $A \vee B$:
  either a proof of A or a proof of B,
  with the information of which one it is;

- Proof of $A \Rightarrow B$:
  function: proofs of A to proofs of B;

- Proof of $\forall x.P(x)$:
  function: maps $n : \mathbb{N}$ to a proof of $P(n)$;

- Proof of $\exists x.P(x)$:
  pair $\langle n, h \rangle$: $n : \mathbb{N}$, h proof of $P(n)$.

Kleene's realizability interpretation
(recursion theory: proof = natural number)

- Proof of $A \wedge B$:
  pair $\#\langle a, b\rangle$, a proof of A, b proof of B;

- Proof of $A \vee B$:
  either $\langle 0, a\rangle$ with a proof of A
  or $\#\langle 1, b\rangle$ with b proof of B

- Proof of $A \Rightarrow B$:
  a code $e : \mathbb{N}$, such that,
  for a proof of A, $\{e\}(a)$ is a proof of B;

- Proof of $\forall x.P(x)$:
  a code $e : \mathbb{N}$, such that,
  for $n : \mathbb{N}$, $\{e\}(a)$ is a proof of $P(n)$;

- Proof of $\exists x.P(x)$:
  pair $\#\langle n, h\rangle$: $n : \mathbb{N}$, h proof of $P(n)$.

Problem: How do we know if a certain natural number e is a correct proof?

How do we know if e is a proof of $\forall x.P(x)$?
Check: $\{e\}(n)$ is a proof $P(n)$ for all n.
Undecidable.

A proof must itself contain all the information needed to verify its correctness. If we cannot decide whether it is correct, then it is an incomplete proof.

Realizability is undecidable.
Not good for Computer Mathematics.

Type Theory
(Curry-Howard Isomorphism)

Correspondence between propositions and types:

$A \wedge B$	$A \times B$
$A \vee B$	$A + B$
$A \Rightarrow B$	$A \to B$
$\forall x : A.P(x)$	$\prod x : A.P(x)$
$\exists x : A.P(x)$	$\sum x : A.P(x)$

Martin-Löf Type Theory:
Base Mathematics on this correspondence.

No such isomorphism between data types and mathematical structures.

Problems still remain:

Extensionality

$$f_1, f_2 : A \to B$$

Computer Science: $f_1 = f_2$ if they are *the same algorithm*
Mathematics: $f_1 = f_2$ if $\forall x : A. f_1(x) = f_2(x)$.

Example: *insertion sort* and *quicksort* are the same function for a mathematician, but very different functions for a computer scientist.

Present situation: no choice, f_1 and f_2 cannot be distinguished but cannot be proven equal.

Subsets

In Mathematics we can do:

$$\frac{A \ \mathsf{Set} \quad P : A \to \mathsf{Prop}}{\{x : A \mid P(x)\}}$$

Such construction is not available in Type Theory.
Problem with decidability:

$$\frac{a : A \quad h : P(a)}{a : \{x : A \mid P(a)\}}$$

This is undecidable: we threw away the proof h, so we cannot check anymore wheter the judgement $a : \{x : A \mid P(a)\}$ is correct.

Quotients

Similar problem:

$$\frac{A \text{ Set} \quad R : A \to A \to \text{Prop}}{A/\equiv \text{ Set}}$$

$$\frac{a_1, a_2 : A \quad h : R(a_1, a_2)}{[a_1]_R = [a_2]_R}$$

We threw away the proof h,
the equality $[a_1]_\equiv = [a_2]_\equiv$ is undecidable.

Partiality

Computer Science: partial recursive functions:
$f : A \to B$ doesn't need to be total,
it may not terminate on some inputs.

If we allow functions to be partial,
the Curry-Howard isomorphism collapses:
$A \to B$ cannot be the type of proofs of $A \Rightarrow B$.

There are solutions to all these problems:

- *Extensionality:* Extensional Type Theory, Setoids, Observational Type Theory;
- *Subsets and Quotients:* Setoids;
- Partiality: Inductive Domain Predicates, Delay Monad.

The Ideal Computer Mathematics System:

- All Standard Mathematical Constructions;
- Set Theory;
- Rich Programming Language;
- Turing Completeness;
- Reflection;
- Implemented in Itself;