Marelle
Mathematics, reasoning, software

Yves Bertot

March 2017

Objectives

- Explore type-theory based theorem proving applications for:
  - mathematics
  - software
- Mathematics needed for software correctness
- Software for advances in mathematics

Elements of context

- The Coq system: a leader in its domain
  - http://coq.inria.fr
  - ACM Software System Award 2013 (https://www.youtube.com/watch?v=vwA4JZ-5qMU)
  - A central tool for prestigious research achievements
  - CompCert (very high quality compiler), DeepSpec (US big project)
  - Feit-Thompson (Major mathematical proof in Algebra)

Methodology

- Write programs in a very safe programming language
- Write expected properties of the data and the programs
- Prove that the properties are satisfied
- Compile into more conventional languages
  - When doing proofs, mathematics may be required
- Computer science with a strong flavor of logic, proof theory, and mathematics

Formalizing mathematics

- Mathematics at the limit of human capabilities
  - *4-color theorem, Kepler conjecture*
  - Odd order theorem (finite groups: Feit-Thompson)
- Reflection: use proved decision procedures
- Algebra and Geometry
  - Linear algebra and related software
  - Polynomial systems
  - Algebraic topology
  - Precise computation of mathematical functions

The main context: The Coq system

- A dedicated modelling language for software and mathematics
  - A functional style,
  - Types used for specifications,
  - Mix data, programs, specifications, and proofs,
  - High-level of abstraction: write once, apply in many domains.
- A collection of tools to derive executable software from the models
  - Generation of executable Caml, Haskell, or Scheme code,
  - Connections to verification on C or Java.
- An international success
  - 2013 ACM Sigplan Software System award
    (Huet, Coquand, Paulin, Barras, Filliatre, Herbelin, Murthy, Bertot, Castéran)
- Our expertise on this system is well-established
  - Coq'Art book by Y. Bertot and P. Castéran,
    translated in Chinese and published by Tsinghua U. P.

The team's contributions

- Extensions to the Coq system:
  - Packages for reasoning on recursive definitions,
  - Libraries for large number computations,
  - Libraries for algebra,
  - Powerful decision procedures, e.g. in algebraic reasoning.
- Study of specific domains:
  - Geometry: figures and algorithms,
  - Polynoms: root isolation, decision procedures,
  - Cryptography robustness,
  - Fundamental mathematics: Finite group theory, linear algebra,
  - Exact real computation, multi-dimensional real analysis.

Technical insight: computing numbers to high precision

- $\pi_0 = 2 + \sqrt{2}$
- $y_0 = \sqrt{2} \qquad y_{n+1} = \dfrac{1 + y_n}{2\sqrt{y_n}}$
- $z_1 = \sqrt{\sqrt{2}} \qquad z_{n+1} = \dfrac{1 + z_n y_n}{(1 + z_n)\sqrt{y_n}}$
- $\pi_{n+1} = \pi_n \dfrac{1 + y_{n+1}}{1 + z_{n+1}}$
- π_n converges quadratically to π

$$0 \leq \pi_n - \pi \leq \frac{4\pi_0}{500^{2^n}}$$

Abstract description

Definitions of `y_` and `z_` given elsewhere

```
Fixpoint agmpi n :=
  match n with
    0%nat => (2 + sqrt 2)
  | S p => agmpi p * (1 + y_ n (/sqrt 2))
              / (1 + z_ n (/sqrt 2))
  end.
```

Fixed precision computation

```
Definition hp1 :=
  (*some large integer*) (2 ^ magnifier)%bigZ.
Definition hp2 := 2 * hp1.
Definition invhp x := (hp1 * hp1 / x)%bigZ.
Definition sqrthp x := BigZ.sqrt (x * hp1).
Definition mulhp x y := ((x * y) / hp1)%bigZ.
Definition addhp x y := (x + y)%bigZ.

Notation "x + y" := (addhp x y) : hp_scope.
Notation "x * y" := (mulhp x y) : hp_scope.
Notation "x / y" := (mulhp x (invhp y)) : hp_scope.

Delimit Scope hp_scope with H.
```

Properties of the operations

- All operations return integers, but they represent rational numbers
- multiplication, division, and square are only approxmations
- the rounding error is bounded by $2^{-\texttt{magnifier}}$
- addition and multiplication by 2 incur no rounding error

Concrete implementation of algorithm

```
Fixpoint agmpi n :=
  match n with
    0%nat => ((hp2 + (sqrthp hp2))%H, y1, z1)
  | S p =>
    let '(pip, yn, zn) := agmpi p in
    let sy := sqrthp yn in
    let zn1 := (hp1 + zn)%H in
      ((pip * ((hp1 + yn)%H / zn1)%H)%H,
       ((hp1 + yn)%H / (hp2 * sy)%H)%H,
       ((hp1 + (yn * zn)%H)%H / (zn1 * sy)%H)%H)
end.
```

Using Coq as a symbolic computation engine

- Computation of one million digits of π in less than 2 hours
- Less efficient than dedicated C or C++ code, but all steps logically verified
- Mathematical proofs of algorithm and of rounding accuracy

Future orientation

- Improvements of the language for proofs and library development
- Research on cryptographic algorithms
- Research on algorithms for robotics
  - Algorithmic geometry and motion planning
  - Ordinary differential equations and control