The Future of Formalised Mathematics

Lawrence C Paulson

UNIVERSITY OF CAMBRIDGE

1. Formalised Mathematics Today

Formalised Mathematics has Arrived!

- *Flyspeck project*: verifying the proof of the Kepler Conjecture by Ferguson and Hales (1998).
  - The proof was *too complicated for referees* and also relied on computer code.
  - The proof text and code were subsequently verified in a collaborative effort involving HOL Light and Isabelle.
- *Four Colour Theorem*: the 1976 proof relied on code, which was finally verified in Coq by Georges Gonthier.

Other Milestones in Formalised Mathematics

- A formalisation of geometry and *nonstandard analysis* to check infinitesimal proofs in Newton's *Principia* (Fleuriot, 1998) [using Isabelle]

- *Prime number theorem* (Avigad; Harrison) [Isabelle and HOL Light]

- *Odd order theorem* (Gonthier et al.) [Coq]

- Gödel's *constructible universe* and (**both**) *incompleteness theorems* (Paulson)

What is the Point of Doing Maths by Machine?

To validate gigantic proofs

To reveal hidden assumptions

To create vast libraries of mathematical knowledge

Ultimately: to augment human intelligence

But isn't Formalised Mathematics Impossible?

Whitehead and Russell needed 362 pages to prove 1+1=2!

We have better formal systems than theirs.

Gödel proved that all "good" formal systems must be incomplete!

We don't need a universal formal system.

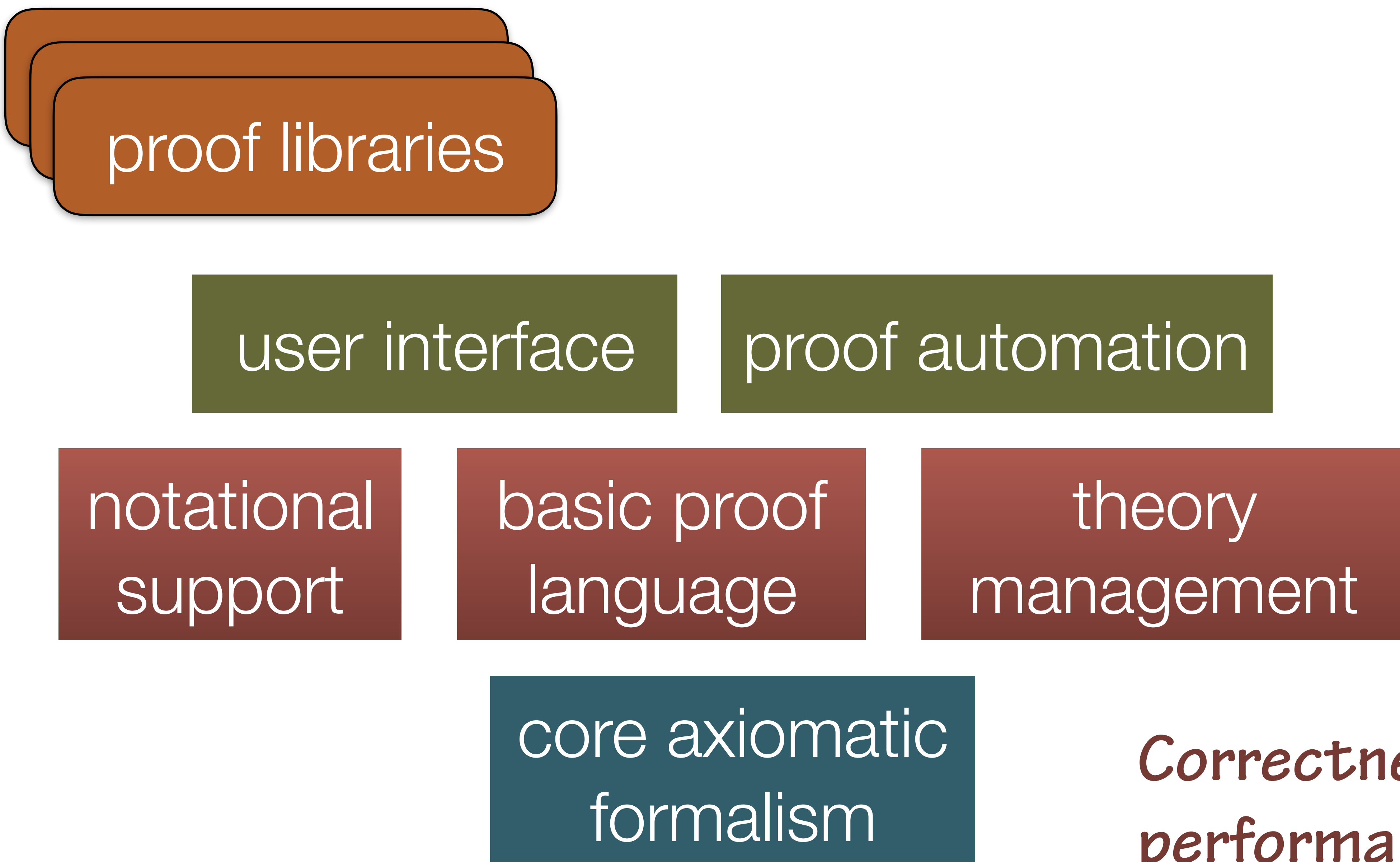
Church proved that first-order logic is undecidable!

We use automation to assist intuition.

Some History; Some Systems

- NG de Bruijn's **Automath** (1968): pioneering; based on a novel type theory; formalised the construction of the reals
- **Coq** by Coquand and Huet (1984) and many others: the most advanced type theory proof assistant
- A Trybulec's **Mizar** (1973): based on set theory with "soft typing" and a readable *structured language*
- John Harrison: real analysis (1992); floating point verification of sqrt, ln, exp; multivariate analysis, etc., using higher-order logic: **HOL** [**Light**]

Components of a Proof Assistant



Correctness, not performance, is key!

2. Formalised Mathematics: Our Choices

The Dimensions of Formalised Mathematics

Types — or not?

What is 1/0?

Search and automation

Syntax of terms and proofs

Type Theory or Set Theory

simple types

polymorphism — *type classes*

HOL **Isabelle/HOL**

predicate subtypes

PVS

dependent types **Coq**

AUTOMATH **Agda**

classical sets

Mizar **Isabelle/ZF**

Type Class Polymorphism

axiomatically define groups, rings, topological spaces, metric spaces,…

prove that something is a metric spaces (say) and *inherit* all proved properties

… supporting uniform mathematical *notation*

But less flexible than dependent types — or classical sets!

…exchanging some flexibility for abstract reasoning

Definedness, or What is 1/0?

- *Don't care*: all terms denote *something*, and1/0 = 1/0. [HOL, Isabelle]

- *Dependent types*: to use x/y, must prove y≠0 (but does the value of x/y depend on this proof?) [Coq,PVS]

- *Free logic*: adopt a formalism where defined[x/y] can be expressed. But if x/0 = x/0 fails, is x/0 ≠ x/0 true? [IMPS]

- *Three-valued logics??*

Search and Automation

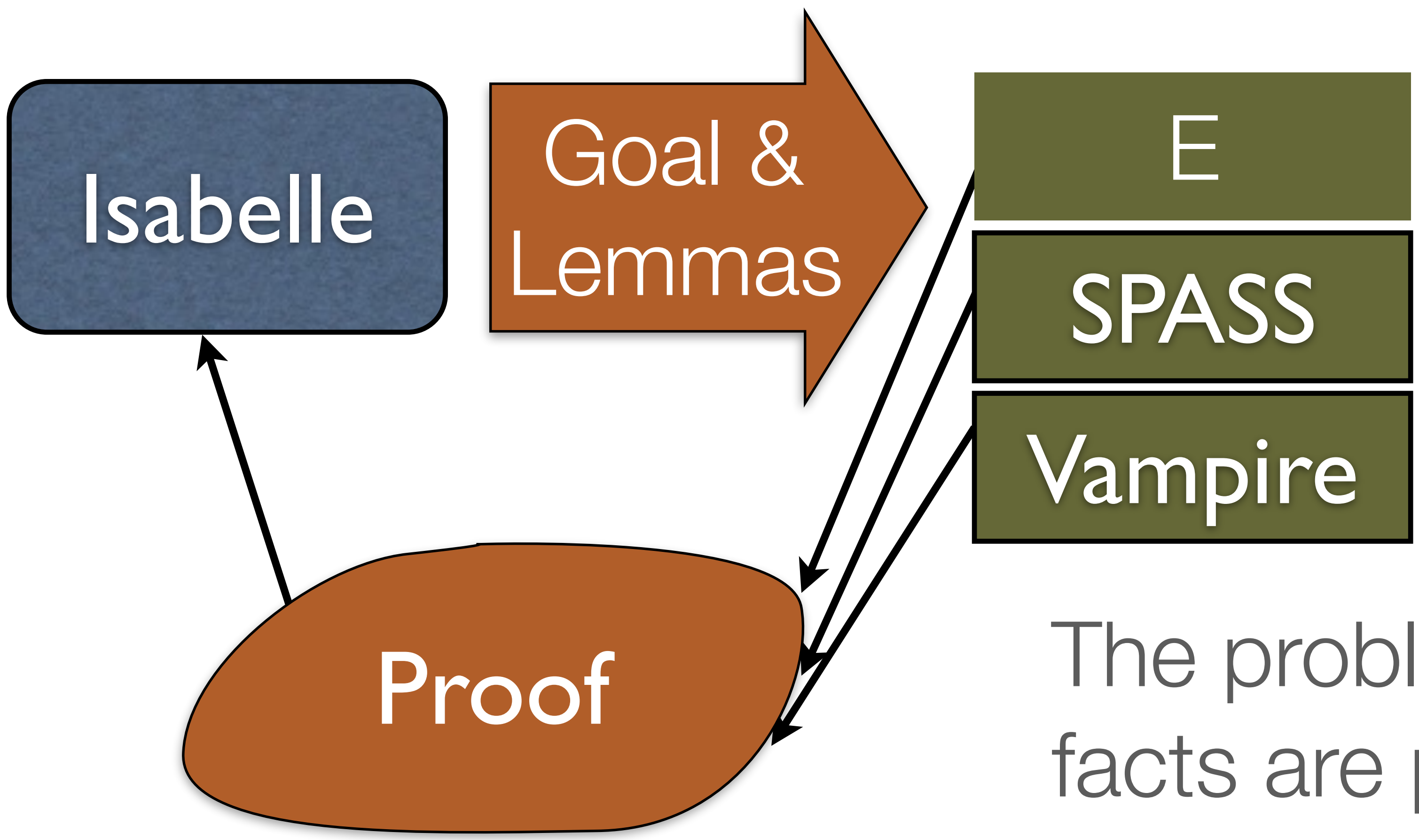
decision procedures:
linear arithmetic,
elementary set theory,
Gröbner basis methods

heuristic methods:
obvious rewriting and
chaining steps, e.g.
x+0 = x

fast, predictable,
powerful… but of
limited scope

flexible but ad-hoc;
changes can break proofs

Sledgehammer: The Ultimate in Heuristic Search



The problem and all known facts are preprocessed and sent to *external provers*.

- Any proof is returned as *source text*.
- We **don't trust** the external provers.
- Our tools write their own proofs!

Syntax, or the Legibility Problem

Mathematical notation is elegant but ambiguous!

$$f(x) \quad f(X) \quad f^{-1}[X]$$

$$x^{-1}y \quad f^{-1}(x) \quad \sin^{-1}(x) \quad \sin^2(x)$$

$$xy \quad x \cdot y \quad \frac{d^2 f}{dx}$$

`Machine notations are merely hideous`

Irrationality of $\sqrt{2}$ in **Coq**

```
Theorem sqrt2_irrational : ~(EX f : frac | ‘f = sqrt 2‘).
Proof.
Move=> [f Df]; Step [Hf22 H2f2]: ‘(mulf f f) = F2‘.
  Apply: (eqr_trans (fracr_mul ? ? ?)); Apply: eqr_trans (fracrz R (Znat 2)).
  By Apply: eqr_trans (square_sqrt (ltrW (ltr02 R))); Apply mulr_morphism.
Step Df2: (eqf F2 (mulf f f)) By Apply/andP; Split; Apply/(fracr_leqPx R ? ?).
Move: f Df2 {Hf22 H2f2 Df} => [d m]; Rewrite: /eqf /= -eqz_leq; Move/eqP.
Rewrite: scalez_mul -scalez_scale scalez_mul mulzC {-1 Zpos}lock /= -lock.
Step []: (Zpos (S d)) = (scalez d (Znat 1)).
  By Apply esym; Apply: eqP; Rewrite scalez_pos; Elim d.
Step [n []]: (EX n | (mulz (Zpos n) (Zpos n)) = (mulz m m)).
  Case: m => [n | n]; LeftBy Exists n.
  By Exists (S n); Rewrite: -{1 (Zneg n)}oppz_opp mulz_oppl -mulz_oppr.
Pose i := (addn (S d) n); Move: (leqnn i) {m}; Rewrite: {1}/i.
Elim: i n d => // [i Hrec] n d Hi Dn2; Move/esym: Dn2 Hi.
Rewrite: -{n}odd_double_half double_addnn !zpos_addn; Move/half: n (odd n) => n.
Case; [Move/((congr oddz) ? ?) | Move/((congr halfz) ? ?)].
  By Rewrite: !mulz_addr oddz_add mulzC !mulz_addr oddz_add !oddz_double.
Rewrite: add0n addnC -addnA add0z mulz_addr !halfz_double mulzC mulz_addr.
Case: n => [|n] Dn2 Hi; LeftBy Rewrite: !mulz_nat in Dn2.
Apply: Hrec Dn2; Apply: (leq_trans 3!i) Hi; Apply: leq_addl.
Qed.
```

Irrationality of $\sqrt{2}$ in **HOL**

```
let NSQRT_2 = prove
 (`!p q. p * p = 2 * q * q ==> q = 0`,
  MATCH_MP_TAC num_WF THEN REWRITE_TAC[RIGHT_IMP_FORALL_THM] THEN
  REPEAT STRIP_TAC THEN FIRST_ASSUM(MP_TAC o AP_TERM `EVEN`) THEN
  REWRITE_TAC[EVEN_MULT; ARITH] THEN REWRITE_TAC[EVEN_EXISTS] THEN
  DISCH_THEN(X_CHOOSE_THEN `m:num` SUBST_ALL_TAC) THEN
  FIRST_X_ASSUM(MP_TAC o SPECL [`q:num`; `m:num`]) THEN
  POP_ASSUM MP_TAC THEN CONV_TAC SOS_RULE);;
```

```
let SQRT_2_IRRATIONAL = prove
 (`~rational(sqrt(&2))`,
  SIMP_TAC[rational; real_abs; SQRT_POS_LE; REAL_POS; NOT_EXISTS_THM] THEN
  REPEAT GEN_TAC THEN DISCH_THEN(CONJUNCTS_THEN2 ASSUME_TAC MP_TAC) THEN
  DISCH_THEN(MP_TAC o AP_TERM `\x. x pow 2`) THEN
  ASM_SIMP_TAC[SQRT_POW_2; REAL_POS; REAL_POW_DIV; REAL_POW_2; REAL_LT_SQUARE;
               REAL_OF_NUM_EQ; REAL_EQ_RDIV_EQ] THEN
  ASM_MESON_TAC[NSQRT_2; REAL_OF_NUM_EQ; REAL_OF_NUM_MUL]);;
```

Irrationality of $\sqrt{2}$ in **Isabelle/HOL**

```
theorem  assumes "prime p"  shows "sqrt p ∉ ℚ"
proof
  from ‹prime p› have p: "1 < p" by (simp add: prime_def)
  assume "sqrt p ∈ ℚ"
  then obtain m n :: nat where
      n: "n ≠ 0" and sqrt_rat: "¦sqrt p¦ = m / n"
    and "coprime m n" by (rule Rats_abs_nat_div_natE)
  have eq: "m² = p * n²"
  proof -
    from n and sqrt_rat have "m = ¦sqrt p¦ * n" by simp
    then show "m² = p * n²"
      by (metis abs_of_nat of_nat_eq_iff of_nat_mult power2_eq_square real_sqrt_abs2 rea
  qed
  have "p dvd m ∧ p dvd n"
  proof
    from eq have "p dvd m²" ..
    with ‹prime p› show "p dvd m" by (rule prime_dvd_power_nat)
    then obtain k where "m = p * k" ..
    with eq have "p * n² = p² * k²" by (auto simp add: power2_eq_square ac_simps)
    with ‹prime p› show "p dvd n"
      by (metis dvd_triv_left nat_mult_dvd_cancel1 power2_eq_square prime_dvd_power_nat
  qed
  then have "p dvd gcd m n" by simp
  with ‹coprime m n› have "p = 1" by simp
  with p show False by simp
qed
```

sledgehammer proofs

Legible proofs (Mizar, Isar) are Necessary!

- To support *maintenance*
  - sometimes definitions must be corrected
  - heuristic proof methods can change
- To allow *reuse*, eventually *translation* to other systems
- To build confidence in the *correctness of verification tools*

  (since **we can inspect the reasoning**)

What do Real Mathematicians Want?

- Harvey Friedman: *set theoretic foundations* with "soft typing" and traditional mathematical notation. *Free logic* for undefined terms!
- Tim Gowers: *automatic* theorem proving, no search, proofs expressed in *natural language*
- NG de Bruijn: *dependent types* but with classical logic. **NO** to set theory!
- Dana Scott: interested in existing technology; expert on free logic and sympathetic to intuitionism.

What Does Isabelle/HOL Give Them?

- Polymorphism with axiomatic type classes
- “Don't care” about undefined values, indeed x/0 = 0!
- Heuristic proof search including *sledgehammer*
- Structured proof language (Isar)
- Interactive development environment (IDE) with "live editing” of proofs

Not a perfect match, but better than some…

3. Formalised Computer Algebra Techniques

real quantifier elimination (QE)

Automatic removal of quantifiers for problems involving **real polynomials**

$$\exists x \, [ax^2 + bx + c = 0]$$

$$\iff$$

$$b^2 \geq 4ac \wedge (c = 0 \vee a \neq 0 \vee \sout{b^2 > 4ac})$$

$$b \neq 0$$

But the equivalent quantifier-free formula can be messy and enormous…

real QE: some history and applications

- Tarski (1930): A first-order RCF* formula can be replaced by an equivalent, quantifier-free one.
- Implies the decidability of RCF and of Euclidean geometry.
- Collins (1975): *Cylindrical Algebraic Decomposition* (CAD), feasible but doubly exponential
- For constraint solving, optimisation, etc., involving polynomials

*RCF (*real-closed field*): any field elementarily equivalent to the reals

Computer Algebra + Verification: Some Milestones

- J Harrison: real QE using semi-definite programming, sum-of-squares and other decision procedures

- Isabelle: proof methods for algebra using SOS [as above] and Gröbner bases

- Muñoz et al.: Bernstein polynomials, Sturm's theorem, etc., for proving polynomial inequalities [in PVS]

- Cohen and Mahboubi: implementation of CAD in Coq, a theory of the *real algebraic numbers** and a proof of QE using pseudo-remainder sequences

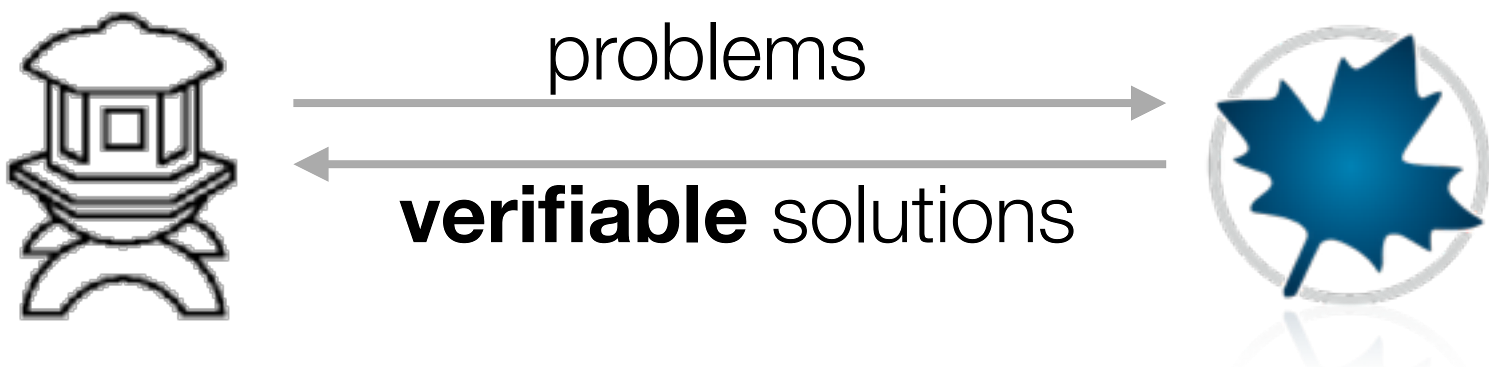
Aside: What Are Real Algebraic Numbers?

They are *real roots of polynomials* (with integer coefficients)

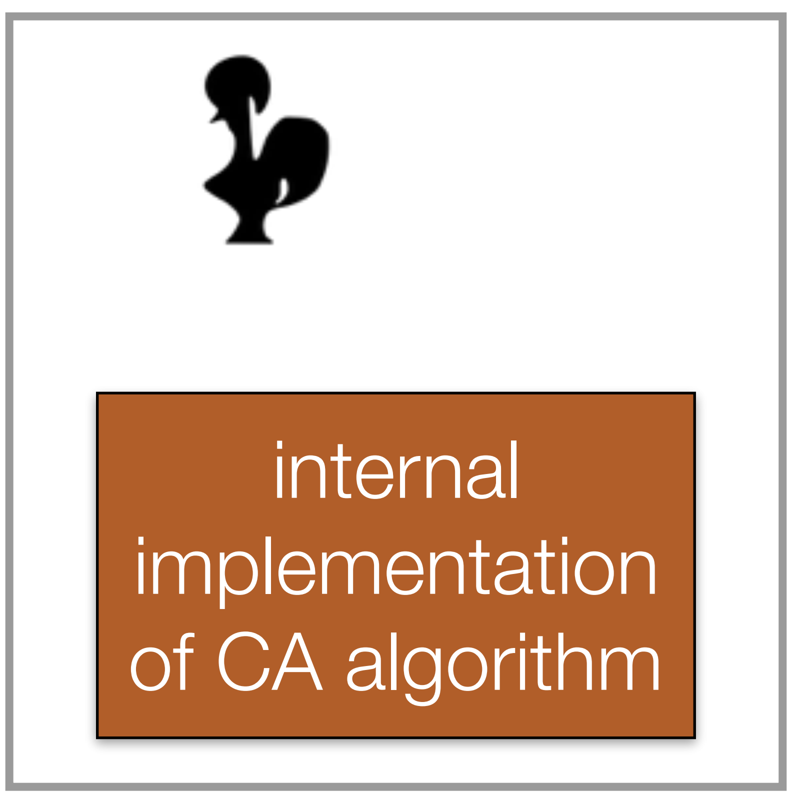
- typically represented by a squarefree polynomial, along with a positive integer or an interval to isolate the root
- arithmetic performed symbolically (and **exactly**) by polynomial manipulations
- equality is *decidable*: they are a subset of the *computable reals* (equality is undecidable on real numbers)

They are the foundation of many CA algorithms.

Computer Algebra in a Proof Assistant?



Harrison and Théry, 1998



Mahboubi, 2005

We don't trust any external CA system.
So we must either…

- Ask for a *verifiable certificate*.
- *Write our own code* and verify it.

Towards Real QE in Isabelle (work by Wenda Li)

- Univariate case: CAD returns the list of roots of a rational polynomial (as *real algebraic numbers*). This list can be verified using the Sturm-Tarski theorem.

- Can be extended to decide the sign of a *bivariate* polynomial at a real algebraic point. Algebraic arithmetic can be performed using external code, then verified.

- The recent verification of Cauchy's residue theorem, the argument principle and Rouché’s theorem will allow the verification of *bivariate certificates*.

4. The Future of Formalised Mathematics

Coq

- The famous “Mathematical Components" library used to formalise the *odd order theorem*
- Ongoing projects to certify *numerical algorithms* for differential equations
- Continued work on *real algebraic geometry* and *real QE*

HOL Light

- Huge inbuilt library of over 23,000 theorems, including 12,400 in complex and multivariate analysis
  - including 86 of the “100 famous theorems” list maintained by Freek Wiedijk.
  - largely the work of a single person: John Harrison

Isabelle's Archive of Formal Proofs

- Online repository for users' proof developments
- Currently over 280 entries, arriving one per week!
- Nearly 2 million lines of proof text
- These have been maintained through successive versions of Isabelle since 2004.

The Future?

- *Most undergraduate mathematics* will be formalised. But proofs should be intelligible to people, not just machines.
- Current verification efforts focus on the **digital** world: compilers, operating systems, protocols…
  - Are we ready to deal with the real, **analogue**, world?
  - Can we do mathematics as well as we do verification?

Acknowledgements

- The development of Isabelle has been supported by various Engineering and Physical Sciences Research Council grants.

- Tobias Nipkow's group at the Technical University of Munich has made enormous contributions, as has the worldwide Isabelle community.

Our tools are coded in **Standard ML**.