PVS and the Pragmatics of Formal Proof[1]

N. Shankar

Computer Science Laboratory
SRI International
Menlo Park, CA

Dec 4, 2015

[1] Supported by NASA NRA NNA13AC55C, NSF Grant CNS-0917375, and DARPA under agreement number FA8750-16-C-0043. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of NASA, NSF, DARPA, or the U.S. Government.

The Rich Legacy of Alfred Tarski [1901–1983]

Proofs and Machines

In science nothing capable of proof ought to be accepted without proof. —Richard Dedekind

The development of mathematics toward greater precision has led, as is well known, to the formalization of large tracts of it, so one can prove any theorem using nothing but a few mechanical rules. –Kurt Gödel

Science is what we understand well enough to explain to a computer. Art is everything else we do. —Donald Knuth

Checking mathematical proofs is potentially one of the most interesting and useful applications of automatic computers. —John McCarthy

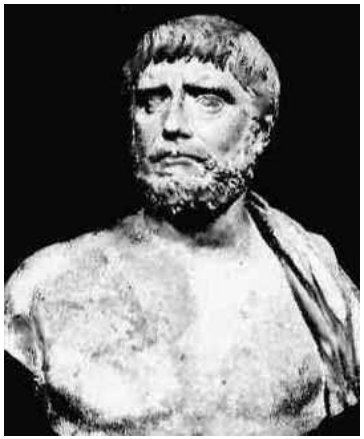
Overview

- Proofs are used to rigorously derive irrefutable conclusions using a small number of 'self-evident' principles of reasoning.
- Mathematical claims in many scientific disciplines are evaluated on the basis of proof.
- *Formal* proofs can be constructed from a small number of precisely stated axioms and rules of inference.
- With the aid of proof assistants, decision procedures, and theorem provers, it is possible to formally verify mathematical claims by checking their proofs.
- Will mathematicians (pure or applied) ever use proof assistants in their work?
- What attributes of a formal language/logic make it more or less usable for capturing mathematical definitions, claims, and proofs?
- In using a proof assistant, what is the best division of labor between the mathematician and the machine?

What is a Pragmatic Foundation?

- Foundational research, from the late nineteenth century on down, has has focused on axiomatic frameworks for formalizing mathematics such as Zermelo-Fraenkel (ZF) set theory and Principia Mathematica (PM).
- These formal calculi are the products of a *reductionist* approach to foundations and fall short as a mathematical vernacular.
- Developing proofs in these calculi has proved difficult.
- With the aid of mechanization, we can design a pragmatic foundation where both formal expression and proof construction are closer to ordinary mathematical discourse.
- A suitable pragmatic foundation can make the interactive development of abstract mathematical discourse a worthwhile creative activity.
- Disclaimer: SRI's Prototype Verification System (PVS) is only used to motivate the kind of features needed in a pragmatic foundation.

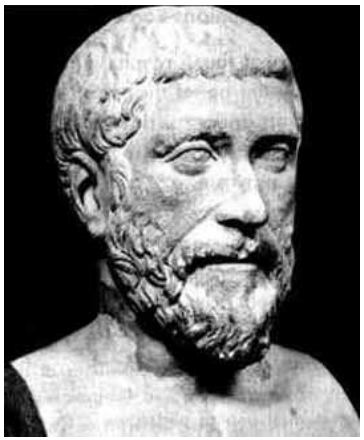
In the Beginning



Thales of Miletus (624 – 547 B.C.): Earliest known person to be credited with theorems and proofs.

> *It was Thales who first conceived the principle of explaining the multitude of phenomena by a small number of hypotheses for all the various manifestations of matter.*

Pythagoras of Samos (569–475 B.C.): Systematic study of mathematics for its own sake.

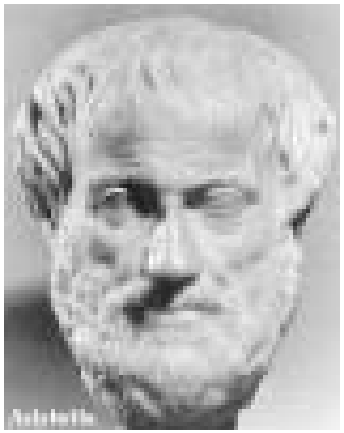


> *... he tried to use his symbolic method of teaching which was similar in all respects to the lessons he had learnt in Egypt. The Samians were not very keen on this method and treated him in a rude and improper manner.*

The Axiomatic Method

Plato (427–347 B.C.): Suggested the idea of a single axiom system for all knowledge.

> *the reality which scientific thought is seeking must be expressible in mathematical terms, mathematics being the most precise and definite kind of thinking of which we are capable.*



Aristotle (384–322 B.C.) of Stagira: Laid the foundation for scientific thought by proposing that all theoretical disciplines must be based on axiomatic principles.

The Elements

Euclid of Alexandria (325–265 B.C.): Systematic compilation and exposition of geometry and number theory.

1. A straight line segment can be drawn joining any two points.
2. Any straight line segment can be extended indefinitely in a straight line.
3. Given any straight line segment, a circle can be drawn having the segment as radius and one endpoint as center.
4. All right angles are congruent.
5. If two lines are drawn which intersect a third in such a way that the sum of the inner angles on one side is less than two right angles, then the two lines inevitably must intersect each other on that side if extended far enough. This postulate is equivalent to what is known as the parallel postulate.

A Glimmer of Rationality

Ramon Llull (1235–1316): Talked of *reducing all knowledge to first principles*. Developed a symbolic notation (*Ars Magna*) and conceived of a reasoning machine.

When he attempted to apply rational thinking to religion, Pope Gregor XI "accused him of confusing faith with reason and condemned his teachings."

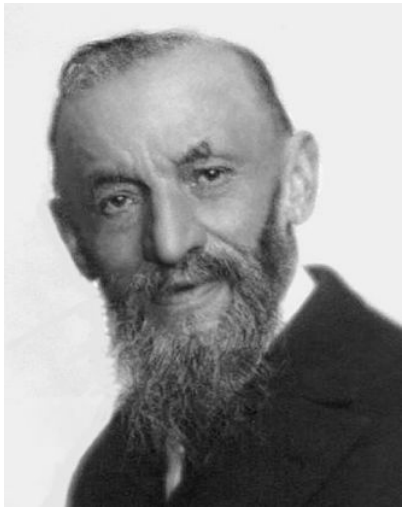
Gottfried Leibniz (1646–1716) The idea of a formal language (*characteristica universalis*) for expressing scholarly knowledge and a mechanical method for making deductions (*calculus ratiocinator*).

What must be achieved is in fact this: that every paralogism be recognized as an error of calculation, *and every* sophism *when expressed in this new kind of notation, appear as a* solecism *or* barbarism, *to be corrected easily by the laws of this philosophical grammar.*

Once this is done, then when a controversy arises, disputation will no more be needed between two philosophers than between two computers. It will suffice that, pen in hand, they sit down to their abacus and (calling in a friend, if they so wish) say to each other: let us calculate.

Formal Arithmetic

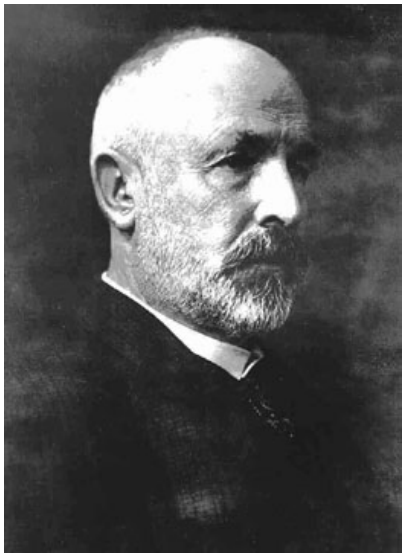
Richard Dedekind (1813–1916) : Early set theoretic ideas, infinite sets, Dedekind cuts, arithmetic.



Giuseppe Peano (1858–1932) : Modern logic notation, arithmetic.

$$0 \neq S(x)$$
$$S(x) = S(y) \implies x = y$$
$$A(0) \wedge (\forall x.A(x) \implies A(S(x))) \implies (\forall x.A(x))$$

Set Theory



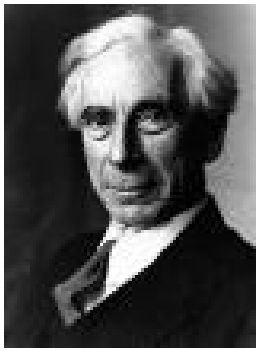
Georg Cantor (1845–1918): Set theory, transfinite sets, continuum hypothesis.

Ernst Zermelo (1871–1953): Formalized set theory, axiom of choice, well-ordering principle.

- Extensionality: $x = y \iff (\forall z.z \in x \iff z \in y)$
- Empty set $\forall x.\neg x \in \emptyset$
- Pairing: $\forall x, y.\exists z.\forall u.u \in z \iff u = x \vee u = y$
- Union: $\forall x.\exists y.\forall z.z \in y \iff (\exists w.z \in w \wedge w \in x)$
- Separation: $\{x \in y | A\}$, for any formula A, $y \notin \mathit{vars}(A)$.
- Infinity: There is a set containing all the finite ordinals.
- Power set: For any set, we have the set of all its subsets.

Modern Formal Logic

Gottlob Frege (1848–1925): A system of quantificational logic.



Bertrand Russell (1872–1970) and Alfred North Whitehead (1861–1947): Ramified type theory, rigorous formal development of a significant portion of mathematics.

- $()^i$ is the type of propositions of order i.
- $(\tau_1^{i_1}, \ldots, \tau_m^{i_m})^i$ with $i > max(i_1, \ldots, i_n)$, is the type of propositional functions from $\tau_1 \times \ldots \times \tau_m$, possibly quantifying over variables of order below i.

Simple Type Theory

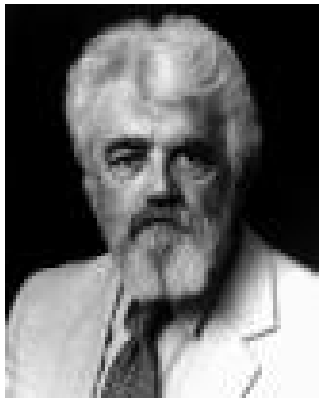
Kurt Gödel (1906–1978): Completeness: Every statement has a counter-model or a proof.
Incompleteness: Any consistent formal theory for arithmetic contains statements that are neither provable nor disprovable.
Such a theory cannot prove its own consistency.

Alonzo Church (1909–1995): Lambda calculus, recursive unsolvability, Church's thesis, simple type theory.

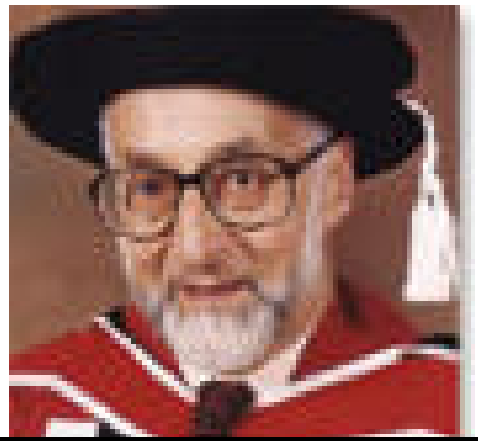
- The types of individuals i and propositions o at level 0
- The function type $S \to T$ is a type at level $n+1$ if S is a type at level at most n and T is a type at level at most $n+1$.

Early Proof Assistants [Automath, LCF, Thm/Nqthm/ACL2]

John McCarthy



Robin Milner



N. G. de Bruijn

Bob Boyer

Woody Bledsoe

J Moore

Higher-order Logic

- Types:
  - `bool` and `real` are types
  - $[T_1 \to T_2]$ and $[T_1, \ldots, T_n]$ are types if the T_i are.
- Products (in n-ary form) are useful so that functions don't have to be Curried.
- Terms:
  - Constants: `TRUE`, `FALSE`, 0, 1.
  - Variables
  - Application: f a
  - Abstraction: $\lambda(x : T) : t$
  - Pairing: (t_1, t_2)
  - Projections: $\texttt{PROJ}_i\ t$
- Polymorphic equality $b = c$ and conditional $\texttt{IF}[T](a, b, c)$, where a is of type `bool` and b and c are of type

Simple Type System: Type Rules

- $\dfrac{}{\Gamma \vdash x : T}\ \Gamma(x) = T$
- $\dfrac{\Gamma, x : S \vdash a : T}{\Gamma \vdash \lambda(x : S) : a : [S \to T]}$
- $\dfrac{\Gamma \vdash f : [S \to T] \qquad \Gamma \vdash a : S}{\Gamma \vdash f\ a : T}$

- $\dfrac{}{\Gamma \vdash \texttt{TRUE} : \texttt{bool}}$, $\dfrac{}{\Gamma \vdash \texttt{FALSE} : \texttt{bool}}$.
- $\dfrac{\Gamma \vdash a : T \qquad \Gamma \vdash b : T}{\Gamma \vdash a = b : \texttt{bool}}$
- $\dfrac{\Gamma \vdash a : \texttt{bool} \qquad \Gamma \vdash b : T \ \ \Gamma \vdash c : T}{\Gamma \vdash \texttt{IF}[T](a, b, c)}$
- $\dfrac{\Gamma \vdash a_1 : T_1 \qquad \Gamma \vdash a_2 : T_2}{\Gamma \vdash (a_1, a_2) : [T_1, T_2]}$
- $\dfrac{\Gamma \vdash a : [T_1, T_2]}{\Gamma \vdash \texttt{PROJ}_i a : T_i}$

Higher-Order Logic

- Higher-order logic makes it possible to define concepts like fixpoints and finiteness.
- Axiom schemes like induction can be written as axioms:

$$\forall P.P(0) \wedge (\forall n.P(n) \implies P(S(n))) \implies (\forall n.P(n)).$$

- The completeness axiom for reals can be stated as

```
real_complete: AXIOM
  FORALL S:
    (EXISTS y: upper_bound?(y, S)) IMPLIES
      (EXISTS y: least_upper_bound?(y, S))
```

- The semantics of higher-order logic can be given in Zermelo set theory.

What about Definitions?

- A typical reductionist attitude is to show that adding a definition is conservative.
- In a pragmatic foundation, definitions should be primitive.
- Primitive Recursive Arithmetic (PRA) [Skolem, Curry, and Gödel's System T for higher-order primitive recursion] allows definitions:
  - $f(\overline{x}) = g(\overline{x}, h_1(\overline{x}), \ldots h_m(\overline{x}))$
  - $f(0, \overline{x}) = g(\overline{x})$
    $f(S(n), \overline{x}) = h(f(n, \overline{x}), n, \overline{x})$
- The equality $a(n, \overline{x}) = b(n, \overline{x})$ holds if $a(n, \overline{x})$, $b(n, \overline{x})$ satisfy the same recursion scheme. [Goodstein]
- This was rediscovered in the context of Lisp as recursion-induction by McCarthy.
- The Boyer–Moore family of provers extended this to accept all recursive definitions with a decreasing ordinal measure.
- PVS accepts any recursive definition with a well-founded measure.

Domains of Definition: Division and Square Root

- With definitions, even primitive recursive ones, you can define addition, multiplication, exponentiation, etc.
- But how should division be defined? What is $1/0$? What is the definition of the real square-root operation on negative numbers?
- Several approaches have been tried:
  - Let $1/0$ be 0, but then the rule $x \cdot (y/x) = y$ clashes with $0 \cdot x = 0$.
  - Let $1/0$ remain uninterpreted, but again all the rules have to be qualified.
  - Allow undefined terms $\Longrightarrow$ free logics: a lot of clutter due to case analysis from undefined
- PVS uses predicate subtypes to precisely define the domain of a definition.

PVS Subtypes

Add the type $\{x : T | a\}$ or just (p) (for predicate p) to the simple type system:

- $$\frac{\Gamma \vdash T : \texttt{TYPE} \qquad \Gamma, x : T \vdash a : \texttt{bool}}{\Gamma \vdash \{x : T | a\} : \texttt{TYPE}}$$
- $$\frac{\Gamma \vdash a : T \qquad \Gamma \models b[a/x]}{\Gamma \vdash a : \{x : T | b\}}$$
- $$\frac{\Gamma \vdash a : \texttt{bool} \qquad \Gamma, a \vdash b : T \quad \Gamma, \neg a \vdash c : T}{\Gamma \vdash \texttt{IF}(a, b, c) : T}$$
- $$\frac{\Gamma \vdash f : [x : S \to T] \qquad \Gamma \vdash a : S}{\Gamma \vdash f\ a : T[a/x]}$$
- $$\frac{\Gamma, x : S \vdash a : T}{\Gamma \vdash \lambda(x : S) : a : [x : S \to T]}$$

- Typechecking becomes undecidable, as do type emptiness and type equivalence!
- Semantically, subtypes are subsets, even at higher types

Using Subtypes

- Division can be declared as

```
nzreal: NONEMPTY_TYPE = {r: real | r /= 0} CONTAINING 1
 /: [real, nzreal -> real]
```

- With /= representing disequality, division can be type-checked in context.

```
div1: CONJECTURE x /= y IMPLIES (x + y)/(x - y) /= 0
```

- Natural numbers are a subtype of integers are a subtype of rationals are a subtype of reals.

Proof Obligations

Typechecking `number_props` generates the proof obligation

```
% Subtype TCC generated (at line 6, column 44) for  (x - y)
  % proved - complete
div1_TCC1: OBLIGATION
   FORALL (x, y: real): x /= y IMPLIES (x - y) /= 0;
```

Proof obligations arising from typechecking are called Type Correctness Conditions (TCCs).

Type Errors

Many type errors correspond to unprovable TCCs, and some TCCs are provable, but surprising.

The standard definition of $\binom{n}{k}$ is as shown

```
n: VAR nat

factorial(n): RECURSIVE posint =
 (IF n = 0 THEN 1 ELSE n * factorial(n-1) ENDIF)
 MEASURE n

n_choose_k(n, (k : upto(n))): posnat =
  factorial(n) / (factorial(k) * factorial(n - k))
```

Typechecking generates the proof obligation

```
n_choose_k_TCC2: OBLIGATION
  FORALL (n: nat, (k: upto(n))):
    integer_pred(factorial(n) / (factorial(k) * factorial(n - k))) AND
     factorial(n) / (factorial(k) * factorial(n - k)) >= 0 AND
      factorial(n) / (factorial(k) * factorial(n - k)) > 0;
```

Typing Judgements

Proof obligations can also be annoying, but typing judgements allow type information to be cached and propagated.

```
px, py:   VAR posreal
nnx, nny: VAR nonneg_real

nnreal_plus_nnreal_is_nnreal:  JUDGEMENT
        +(nnx, nny) HAS_TYPE nnreal
nnreal_times_nnreal_is_nnreal: JUDGEMENT
        *(nnx, nny) HAS_TYPE nnreal
posreal_times_posreal_is_posreal: JUDGEMENT
        *(px, py) HAS_TYPE posreal
```

Judgements can capture closure conditions (composition of continuous functions is continuous) as well as implicit subtype relationships.

(Rank-invariant) Dependent Types

Dependent records have the form
$[\# \ l_1 : T_1, l_2 : T_2(l_1), \ldots, l_n : T_N(l_1, \ldots, l_{n-1}) \ \#]$.

```
finite_sequences [T: TYPE]: THEORY
 BEGIN
  finite_sequence: TYPE
     = [# length: nat, seq: [below[length] -> T] #]
 END finite_sequences
```

Dependent function types have the form $[x : T_1 \rightarrow T_2(x)]$.

```
  i, j: VAR nat

  g91(i): nat = (IF i > 100 THEN i - 10 ELSE 91 ENDIF)

  f91(i) : RECURSIVE {j | j  = g91(i)}
  = (IF i>100
      THEN i-10
      ELSE f91(f91(i+11))
     ENDIF)
  MEASURE (IF i>101 THEN 0 ELSE 101-i ENDIF)
```

Theories

```
Tarski_Knaster   [T : TYPE, ⊏ : PRED[[T, T]], ⊓ : [set[T] -> T] ]
                 : THEORY
  BEGIN
   ASSUMING
    x, y, z: VAR T

    X, Y, Z : VAR set[T]  %synonym for [T -> bool]

    f, g : VAR [T -> T]

    reflexivity: ASSUMPTION  x ⊏ x

    antisymmetry: ASSUMPTION  x ⊏ y AND y ⊏ x IMPLIES x = y

    transitivity : ASSUMPTION x ⊏ y AND y ⊏ z IMPLIES x ⊏ z

    glb_is_lb: ASSUMPTION  X(x) IMPLIES ⊓(X) ⊏ x

    glb_is_glb: ASSUMPTION
       (FORALL x: X(x) IMPLIES y ⊏ x)
      IMPLIES y ⊏ ⊓(X)
   ENDASSUMING
```

Tarski–Knaster Theorem

```
    ⋮
    mono?(f): bool = (FORALL x, y: x ⊏ y IMPLIES f(x) ⊏ f(y))

    lfp(f) : T = ⊓(x | f(x) ⊏ x)

    fixpoint?(f)(x): bool =
      (f(x) = x)

    TK1: THEOREM
     mono?(f) IMPLIES
       lfp(f) = f(lfp(f))

  END Tarski_Knaster
```

Monotone operators on complete lattices have fixed points. The fixed point defined above can be shown to be the least such fixed point.

Theory Interpretations

- Theories can be imported with or without explicit parameters.
- Theories can also be interpreted by assigning interpretations to uninterpreted symbols.
- For example, a complete meet-semilattice L is actually a complete lattice by interpreting L as L, $\sqsubset$ as $\sqsupset$ (transpose of $\sqsubset$, and $\sqcup$ as $\sqcap$ (the meet of the upper bounds).

Recursive Datatypes

- A list datatype with *constructors* `null` and `cons` is declared as

```
list [T: TYPE]: DATATYPE
 BEGIN
  null: null?
  cons (car: T, cdr:list):cons?

  END list
```

- The *accessors* for `cons` are `car` and `cdr`.
- The *recognizers* are `null?` for `null` and `cons?` for `cons`-terms.
- The declaration generates a family of theories with the datatype axioms, induction principles, and some useful definitions.

PVS Libraries (NASAlib)

Theorem	Author
Cauchy-Schwarz Inequality	Ricky Butler
Derivative of a Power Series	Ricky Butler
Fundamental Theorem of Arithmetic	Ricky Butler
Fundamental Theorem of Calculus	Ricky Butler
Fundamental Theorem of Interval Arithmetic	César Muñoz, A. Narkawicz
Inclusion Theorem of Interval Arithmetic	César Muñoz, A. Narkawicz
Infinitude of Primes	Ricky Butler

PVS Libraries (NASAlib)

Theorem	Author
Integral of a Power Series	Ricky Butler
Intermediate Value Theorem	Bruno Dutertre
Law of Cosines	César Muñoz
Mean Value Theorem	Bruno Dutertre
Mantel's Theorem	Aaron Dutle
Menger's Theorem	Jon Sjogren
Order of a Subgroup	David Lester
Pythagorean Property - Sine and Cosine	David Lester
Ramsey's Theorem	N. Shankar
Sum of a Geometric Series	Ricky Butler
Taylor's Theorem	Ricky Butler
Trig Identities: Sum and Diff of Two Angles	David Lester
Trig Identities: Double Angle Formulas	David Lester

PVS Libraries (NASAlib)

Theorem	Author
Schroeder-Bernstein Theorem	Jerry James
Denumerability of the Rational Numbers	Jerry James
Heine Theorem and Multiary Variants	Anthony Narkawicz
Fubini-Tonelli Lemmas	David Lester
Knuth-Bendix Critical Pair Theorem	André Galdino, Mauricio Ayala
Church-Rosser Theorem	André Galdino, Mauricio Ayala
Newman Lemma	André Galdino, Mauricio Ayala
Yokouchi Lemma	André Galdino, Mauricio Ayala
Robinson Unification	Andreia Avelar, Maurcio Ayala
Confluence of Orthogonal TRSs	Ana Rocha, Mauricio Ayala
Sturm's Theorem	Anthony Narkawicz
Tarski's Theorem	Anthony Narkawicz, Aaron Dutle

Toward Greater Expressiveness

- Proof assistants like HOL, HOL Light, ProofPower, and Isabelle/HOL are also based on classical higher-order logic, but lack subtyping and dependent typing.
- Nuprl and Coq are based on the propositions-as-types interpretation.
- In Coq,
  - Terms (proofs) can be applied to other terms.
  - Terms can be applied to types.
  - Types can depend on terms (dependent types).
  - Types can depend on types (polymorphic types).
  - Kinds (universes) can be applied to lesser kinds.
- In Martin-Löf style type theories, for each type S, there is a type (proposition) $Id_S(a, b)$ expressing the equality of two terms a and b of type S.
- Homotopy Type Theory builds an increasingly refined notion of equality proofs depending on whether such proofs are unique (propositions), the equality proofs of equality proofs are unique (sets), and so on.

Will Mathematicians Formalize their Arguments?

```
  From Laurent Théry
Date: Thursday 20 September 2012, 20:24
Re: [Coqfinitgroup-commits] r4105  trunk
Hi,

Just for fun

Feit Thompson statement in Coq:

Theorem Feit_Thompson (gT : finGroupType) (G : group gT) :
                          odd #|G| -> solvable G.

How big it is:

Number of lines ~ 170 000
Number of definitions ~15 000
Number of theorems ~ 4 200
Fun ~ enormous!
```

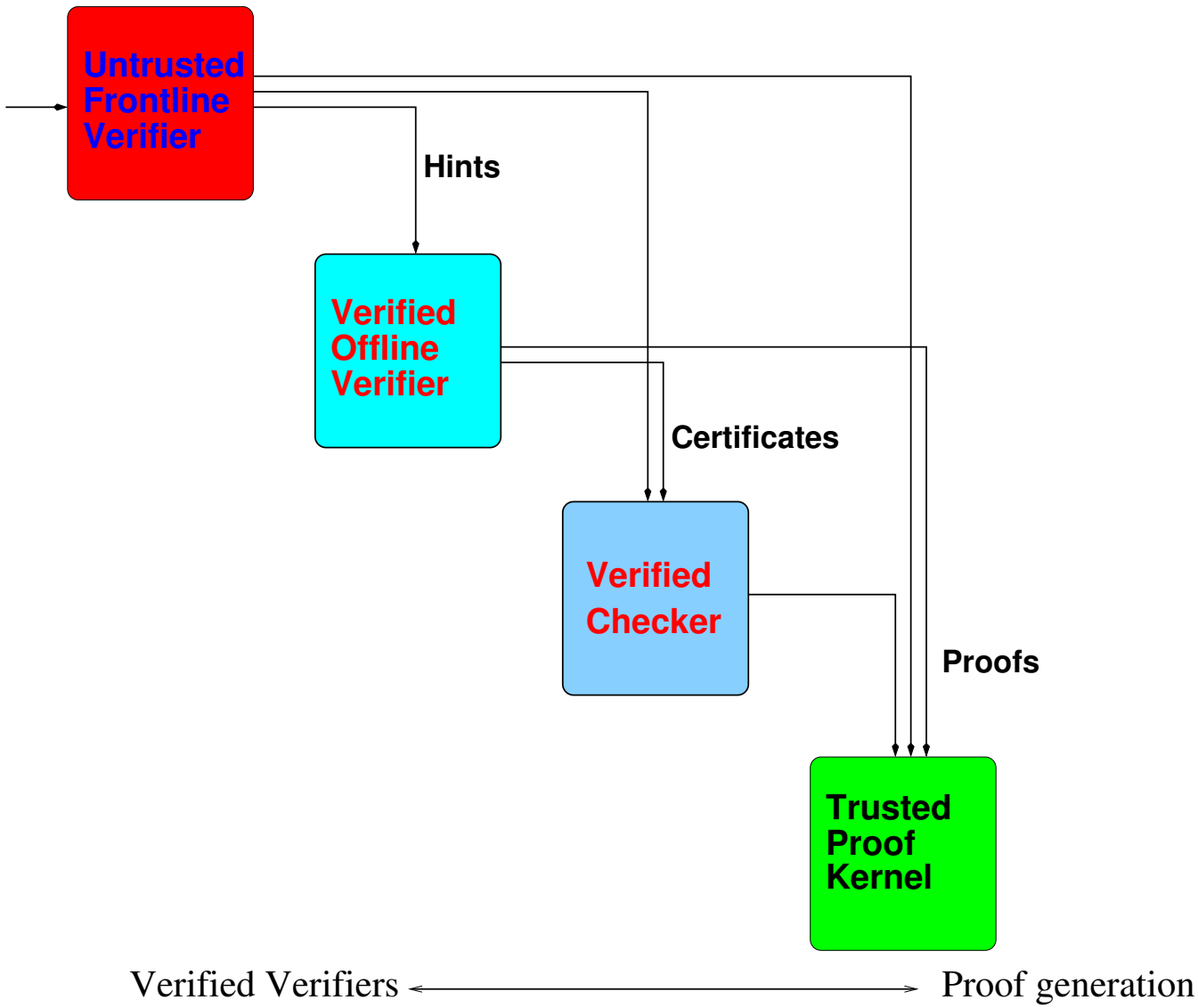
Can We Trust Automation?

. . . we ask whether this guarantee would be weakened by leaving the mechanical verification to a machine. This is a very reasonable, relevant and important question. It is related to proving the correctness of fairly extensive computer programs, and checking the interpretation of the specifications of those programs. And there is more: the hardware, the operating system have to be inspected thoroughly, as well as the syntax, the semantics and the compiler of the programming language. And even if all this would be covered to satisfaction, there is the fear that a computer might make errors without indicating them by total breakdown. I do not see how we ever can get to an absolute guarantee. But one has to admit that compared to human mechanical verification, computers are superior in every respect.

N. G. de Bruijn

- HOL Light has a small kernel (500 lines) that has been shown sound in a strengthened version of HOL Light
- Some theorem provers have been verified down to hardware.

Kernel of Truth



Conclusions

- The main value of a proof assistant is in the quality of dialogue, not the certification of correct proofs.
- Proof assistants can and should be used for developing new mathematical results as well as understanding existing mathematical developments.
- Mathematicians can play with these assistants in the way that chess grandmasters practice against chess programs.
- The reductionist approach to foundations serves a purpose
  - It reduces mathematics to a small set of accepted beliefs.
  - It curbs runaway abstraction.
- A pragmatic foundation should favor an abstractionist approach to support generality and reuse.

Further Reading

- John Rushby, *Formal Methods and their Role in the Certification of Critical Systems*, SRI-CSL Technical Report, 1995.
- John Rushby and Sam Owre and N. Shankar, *Subtypes for Specifications: Predicate Subtyping in PVS*, IEEE TSE, 1998.
- NASA Langley Formal Methods Team, *NASA PVS Library*, `http://shemesh.larc.nasa.gov/fm/ftp/larc/PVS-library/`.
- Jeremy Avigad and John Harrison. *Formally verified mathematics*. CACM 57, 2014.
- Thierry Coquand, *Type Theory*, `http://plato.stanford.edu/entries/type-theory/`.
- N. Shankar, *Model Checking and Deduction*, Handbook of Model Checking, 2016.