# Applications of automated reasoning

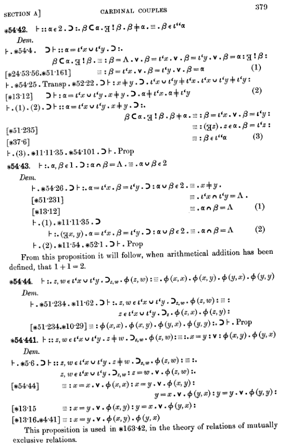
John Harrison

Intel Corporation

4th February 2014 (15:00–16:00)

## Formalization of Mathematics

# Some formal proofs from 1910



**∗54·42.** $\vdash :: \alpha \epsilon 2 . \supset :. \beta \subset \alpha . \exists ! \beta . \beta \neq \alpha . \equiv . \beta \epsilon \iota‘‘\alpha$

*Dem.*

$\vdash . *54·4 . \quad \supset \vdash :: \alpha = \iota‘x \cup \iota‘y . \supset :.$

$\qquad \beta \subset \alpha . \exists ! \beta . \equiv : \beta = \Lambda . \vee . \beta = \iota‘x . \vee . \beta = \iota‘y . \vee . \beta = \alpha : \exists ! \beta :$

[∗24·53·56.∗51·161] $\quad \equiv : \beta = \iota‘x . \vee . \beta = \iota‘y . \vee . \beta = \alpha \qquad (1)$

$\vdash . *54·25 . \text{Transp} . *52·22 . \supset \vdash : x \neq y . \supset . \iota‘x \cup \iota‘y \neq \iota‘x . \iota‘x \cup \iota‘y \neq \iota‘y :$

[∗13·12] $\quad \supset \vdash : \alpha = \iota‘x \cup \iota‘y . x \neq y . \supset . \alpha \neq \iota‘x . \alpha \neq \iota‘y \qquad (2)$

$\vdash . (1) . (2) . \supset \vdash :: \alpha = \iota‘x \cup \iota‘y . x \neq y . \supset :.$

$\qquad \beta \subset \alpha . \exists ! \beta . \beta \neq \alpha . \equiv : \beta = \iota‘x . \vee . \beta = \iota‘y :$

[∗51·235] $\quad \equiv : (\exists z) . z \epsilon \alpha . \beta = \iota‘z :$

[∗37·6] $\quad \equiv : \beta \epsilon \iota‘‘\alpha \qquad (3)$

$\vdash . (3) . *11·11·35 . *54·101 . \supset \vdash . \text{Prop}$

**∗54·43.** $\vdash :. \alpha, \beta \epsilon 1 . \supset : \alpha \cap \beta = \Lambda . \equiv . \alpha \cup \beta \epsilon 2$

*Dem.*

$\vdash . *54·26 . \supset \vdash :. \alpha = \iota‘x . \beta = \iota‘y . \supset : \alpha \cup \beta \epsilon 2 . \equiv . x \neq y .$

[∗51·231] $\quad \equiv . \iota‘x \cap \iota‘y = \Lambda .$

[∗13·12] $\quad \equiv . \alpha \cap \beta = \Lambda \qquad (1)$

$\vdash . (1) . *11·11·35 . \supset$

$\qquad \vdash :. (\exists x, y) . \alpha = \iota‘x . \beta = \iota‘y . \supset : \alpha \cup \beta \epsilon 2 . \equiv . \alpha \cap \beta = \Lambda \qquad (2)$

$\vdash . (2) . *11·54 . *52·1 . \supset \vdash . \text{Prop}$

From this proposition it will follow, when arithmetical addition has been defined, that $1 + 1 = 2$.

**∗54·44.** $\vdash :. z, w \epsilon \iota‘x \cup \iota‘y . \supset_{z,w} . \phi(z, w) : \equiv . \phi(x, x) . \phi(x, y) . \phi(y, x) . \phi(y, y)$

*Dem.*

$\vdash . *51·234 . *11·62 . \supset \vdash :. z, w \epsilon \iota‘x \cup \iota‘y . \supset_{z,w} . \phi(z, w) : \equiv :$

$\qquad z \epsilon \iota‘x \cup \iota‘y . \supset_z . \phi(z, x) . \phi(z, y) :$

[∗51·234.∗10·29] $\equiv : \phi(x, x) . \phi(x, y) . \phi(y, x) . \phi(y, y) :. \supset \vdash . \text{Prop}$

**∗54·441.** $\vdash :: z, w \epsilon \iota‘x \cup \iota‘y . z \neq w . \supset_{z,w} . \phi(z, w) : \equiv :. x = y : \vee : \phi(x, y) . \phi(y, x)$

*Dem.*

$\vdash . *5·6 . \supset \vdash :: z, w \epsilon \iota‘x \cup \iota‘y . z \neq w . \supset_{z,w} . \phi(z, w) : \equiv :.$

$\qquad z, w \epsilon \iota‘x \cup \iota‘y . \supset_{z,w} : z = w . \vee . \phi(z, w) :.$

[∗54·44] $\quad \equiv : x = x . \vee . \phi(x, x) : x = y . \vee . \phi(x, y) :$

$\qquad y = x . \vee . \phi(y, x) : y = y . \vee . \phi(y, y) :$

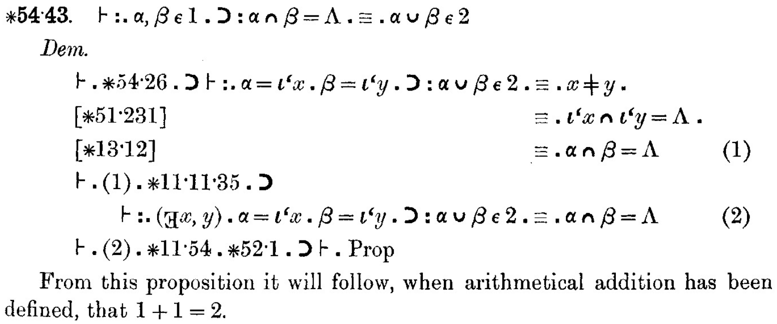
[∗13·15 $\quad \equiv : x = y . \vee . \phi(x, y) : y = x . \vee . \phi(y, x) :$

[∗13·16.∗4·41] $\equiv : x = y . \vee . \phi(x, y) . \phi(y, x)$

This proposition is used in ∗163·42, in the theory of relations of mutually exclusive relations.

This is p379 of Whitehead and Russell's *Principia Mathematica*.

## Zooming in . . .



$\ast$**54·43.** $\vdash :. \alpha, \beta \in 1 . \supset : \alpha \cap \beta = \Lambda . \equiv . \alpha \cup \beta \in 2$

*Dem.*

$\vdash . \ast 54{\cdot}26 . \supset \vdash :. \alpha = \iota`x . \beta = \iota`y . \supset : \alpha \cup \beta \in 2 . \equiv . x \neq y .$

$[\ast 51{\cdot}231] \qquad \equiv . \iota`x \cap \iota`y = \Lambda .$

$[\ast 13{\cdot}12] \qquad \equiv . \alpha \cap \beta = \Lambda \qquad (1)$

$\vdash . (1) . \ast 11{\cdot}11{\cdot}35 . \supset$

$\vdash :. (\exists x, y) . \alpha = \iota`x . \beta = \iota`y . \supset : \alpha \cup \beta \in 2 . \equiv . \alpha \cap \beta = \Lambda \qquad (2)$

$\vdash . (2) . \ast 11{\cdot}54 . \ast 52{\cdot}1 . \supset \vdash .$ Prop

From this proposition it will follow, when arithmetical addition has been defined, that $1 + 1 = 2$.

## 100 years since Principia Mathematica

*Principia Mathematica* was the first sustained and successful actual formalization of mathematics.

## 100 years since Principia Mathematica

*Principia Mathematica* was the first sustained and successful actual formalization of mathematics.

- This practical formal mathematics was to forestall objections to Russell and Whitehead's 'logicist' thesis, not a goal in itself.

## 100 years since Principia Mathematica

*Principia Mathematica* was the first sustained and successful actual formalization of mathematics.

- This practical formal mathematics was to forestall objections to Russell and Whitehead's 'logicist' thesis, not a goal in itself.
- The development was difficult and painstaking, and has probably been studied in detail by very few.

## 100 years since Principia Mathematica

*Principia Mathematica* was the first sustained and successful actual formalization of mathematics.

- This practical formal mathematics was to forestall objections to Russell and Whitehead's 'logicist' thesis, not a goal in itself.
- The development was difficult and painstaking, and has probably been studied in detail by very few.
- Subsequently, the idea of actually formalizing proofs has not been taken very seriously, and few mathematicians do it today.

## 100 years since Principia Mathematica

*Principia Mathematica* was the first sustained and successful actual formalization of mathematics.

- This practical formal mathematics was to forestall objections to Russell and Whitehead's 'logicist' thesis, not a goal in itself.
- The development was difficult and painstaking, and has probably been studied in detail by very few.
- Subsequently, the idea of actually formalizing proofs has not been taken very seriously, and few mathematicians do it today.

But thanks to the rise of the computer, the actual formalization of mathematics is attracting more interest.

## The importance of computers for formal proof

Computers can both help *with* formal proof and give us new reasons to be interested in it:

## The importance of computers for formal proof

Computers can both help *with* formal proof and give us new reasons to be interested in it:

- Computers are expressly designed for performing formal manipulations quickly and without error, so can be used to check and partly generate formal proofs.

## The importance of computers for formal proof

Computers can both help *with* formal proof and give us new reasons to be interested in it:

- Computers are expressly designed for performing formal manipulations quickly and without error, so can be used to check and partly generate formal proofs.
- Correctness questions in computer science (hardware, programs, protocols etc.) generate a whole new array of difficult mathematical and logical problems where formal proof can help.

## The importance of computers for formal proof

Computers can both help *with* formal proof and give us new reasons to be interested in it:

- Computers are expressly designed for performing formal manipulations quickly and without error, so can be used to check and partly generate formal proofs.
- Correctness questions in computer science (hardware, programs, protocols etc.) generate a whole new array of difficult mathematical and logical problems where formal proof can help.

Because of these dual connections, interest in formal proofs is strongest among computer scientists, but some ‘mainstream’ mathematicians are becoming interested too.

## Russell was an early fan of mechanized formal proof

Newell, Shaw and Simon in the 1950s developed a 'Logic Theory Machine' program that could prove some of the theorems from *Principia Mathematica* automatically.

## Russell was an early fan of mechanized formal proof

Newell, Shaw and Simon in the 1950s developed a ‘Logic Theory Machine’ program that could prove some of the theorems from *Principia Mathematica* automatically.

> *“I am delighted to know that Principia Mathematica can now be done by machinery [...] I am quite willing to believe that everything in deductive logic can be done by machinery. [...] I wish Whitehead and I had known of this possibility before we wasted 10 years doing it by hand.” [letter from Russell to Simon]*

## Russell was an early fan of mechanized formal proof

Newell, Shaw and Simon in the 1950s developed a 'Logic Theory Machine' program that could prove some of the theorems from *Principia Mathematica* automatically.

> *"I am delighted to know that Principia Mathematica can now be done by machinery [...] I am quite willing to believe that everything in deductive logic can be done by machinery. [...] I wish Whitehead and I had known of this possibility before we wasted 10 years doing it by hand." [letter from Russell to Simon]*

Newell and Simon's paper on a more elegant proof of one result in PM was rejected by JSL because it was co-authored by a machine.

## Formalization in current mathematics

Traditionally, we understand *formalization* to have two components, corresponding to Leibniz's *characteristica universalis* and *calculus ratiocinator*.

## Formalization in current mathematics

Traditionally, we understand *formalization* to have two components, corresponding to Leibniz's *characteristica universalis* and *calculus ratiocinator*.

- Express *statements* of theorems in a formal language, typically in terms of primitive notions such as sets.

## Formalization in current mathematics

Traditionally, we understand *formalization* to have two components, corresponding to Leibniz's *characteristica universalis* and *calculus ratiocinator*.

- Express *statements* of theorems in a formal language, typically in terms of primitive notions such as sets.
- Write *proofs* using a fixed set of formal inference rules, whose correct form can be checked algorithmically.

## Formalization in current mathematics

Traditionally, we understand *formalization* to have two components, corresponding to Leibniz's *characteristica universalis* and *calculus ratiocinator*.

- Express *statements* of theorems in a formal language, typically in terms of primitive notions such as sets.
- Write *proofs* using a fixed set of formal inference rules, whose correct form can be checked algorithmically.

Correctness of a formal proof is an objective question, algorithmically checkable in principle.

## Mathematics is reduced to sets

The explication of mathematical concepts in terms of sets is now quite widely accepted (see *Bourbaki*).

- A real number is a set of rational numbers ...
- A Turing machine is a quintuple $(\Sigma, A, \ldots)$

Statements in such terms are generally considered clearer and more objective. (Consider pathological functions from real analysis ...)

## Symbolism is important

The use of symbolism in mathematics has been steadily increasing over the centuries:

> *“[Symbols] have invariably been introduced to make things easy. [. . . ] by the aid of symbolism, we can make transitions in reasoning almost mechanically by the eye, which otherwise would call into play the higher faculties of the brain. [. . . ] Civilisation advances by extending the number of important operations which can be performed without thinking about them.” (Whitehead,* An Introduction to Mathematics*)*

## Formalization is the key to rigour

Formalization now has a important conceptual role in principle:

> *"... the correctness of a mathematical text is verified by comparing it, more or less explicitly, with the rules of a formalized language." (Bourbaki,* Theory of Sets*)*
>
> *"A Mathematical proof is rigorous when it is (or could be) written out in the first-order predicate language $L(\in)$ as a sequence of inferences from the axioms ZFC, each inference made according to one of the stated rules." (Mac Lane,* Mathematics: Form and Function*)*

What about in practice?

## Mathematicians don't use logical symbols

Variables were used in logic long before they appeared in mathematics, but logical symbolism is rare in current mathematics. Logical relationships are usually expressed in natural language, with all its subtlety and ambiguity.
Logical symbols like '$\Rightarrow$' and '$\forall$' are used *ad hoc*, mainly for their abbreviatory effect.

> *"as far as the mathematical community is concerned George Boole has lived in vain" (Dijkstra)*

## Mathematicians don't do formal proofs . . .

The idea of actual formalization of mathematical proofs has not been taken very seriously:

> *"this mechanical method of deducing some mathematical theorems has no practical value because it is too complicated in practice." (Rasiowa and Sikorski,* The Mathematics of Metamathematics*)*
>
> *"[. . . ] the tiniest proof at the beginning of the Theory of Sets would already require several hundreds of signs for its complete formalization. [. . . ] formalized mathematics cannot in practice be written down in full [. . . ] We shall therefore very quickly abandon formalized mathematics" (Bourbaki,* Theory of Sets*)*

## . . . Poincaré's had a particular aversion . . .

*I see in logistic only shackles for the inventor. It is no aid to conciseness — far from it, and if twenty-seven equations were necessary to establish that 1 is a number, how many would be needed to prove a real theorem?*
*If we distinguish, with Whitehead, the individual $x$, the class of which the only member is $x$ and [...] the class of which the only member is the class of which the only member is $x$ [...], do you think these distinctions, useful as they may be, go far to quicken our pace?*

## . . . and the few people that do end up regretting it

> *"my intellect never quite recovered from the strain of writing [*Principia Mathematica*]. I have been ever since definitely less capable of dealing with difficult abstractions than I was before." (Russell,* Autobiography*)*

However, now we have computers to check and even automatically generate formal proofs.

Our goal is now not so much philosphical, but to achieve a real, practical, useful increase in the precision and accuracy of mathematical proofs.

## Are proofs in doubt?

Mathematical proofs are subjected to peer review, but errors often escape unnoticed.

> *"Professor Offord and I recently committed ourselves to an odd mistake (Annals of Mathematics (2) 49, 923, 1.5). In formulating a proof a plus sign got omitted, becoming in effect a multiplication sign. The resulting false formula got accepted as a basis for the ensuing fallacious argument. (In defence, the final result was known to be true.)" (Littlewood,* Miscellany*)*

A book by Lecat gave 130 pages of errors made by major mathematicians up to 1900.
A similar book today would no doubt fill many volumes.

## Even elegant textbook proofs can be wrong

*"The second edition gives us the opportunity to present this new version of our book: It contains three additional chapters, substantial revisions and new proofs in several others, as well as minor amendments and improvements, many of them based on the suggestions we received. It also misses one of the old chapters, about the "problem of the thirteen spheres," whose proof turned out to need details that we couldn't complete in a way that would make it brief and elegant." (Aigner and Ziegler,* Proofs from the Book*)*

## Most doubtful informal proofs

What are the proofs where we do in practice worry about correctness?

- Those that are just very long and involved. Classification of finite simple groups, Seymour-Robertson graph minor theorem
- Those that involve extensive computer checking that cannot in practice be verified by hand. Four-colour theorem, Hales's proof of the Kepler conjecture
- Those that are about very technical areas where complete rigour is painful. Some branches of proof theory, formal verification of hardware or software

## Formalized theorems and libraries of mathematics

Interactive provers have been used to check quite non-trivial results, albeit not close to today's research frontiers, e.g.

- Jordan Curve Theorem — Tom Hales (HOL Light), Andrzej Trybulec et al. (Mizar)
- Prime Number Theorem — Jeremy Avigad et al (Isabelle/HOL), John Harrison (HOL Light)
- Dirichlet's Theorem — John Harrison (HOL Light)
- First and second Cartan Theorems — Marco Maggesi et al (HOL Light)

## Formalized theorems and libraries of mathematics

Interactive provers have been used to check quite non-trivial results, albeit not close to today's research frontiers, e.g.

- Jordan Curve Theorem — Tom Hales (HOL Light), Andrzej Trybulec et al. (Mizar)
- Prime Number Theorem — Jeremy Avigad et al (Isabelle/HOL), John Harrison (HOL Light)
- Dirichlet's Theorem — John Harrison (HOL Light)
- First and second Cartan Theorems — Marco Maggesi et al (HOL Light)

According to the *Formalizing 100 theorems* page, 88% of a list of the 'top 100 mathematical theorems' have been formalized using interactive theorem provers.
In the process, provers are building up ever-larger libraries of pre-proved theorems that can be deployed in future proofs.

## The four-colour Theorem

Early history indicates fallibility of the traditional social process:

- Proof claimed by Kempe in 1879
- Flaw only point out in print by Heaywood in 1890

## The four-colour Theorem

Early history indicates fallibility of the traditional social process:

- Proof claimed by Kempe in 1879
- Flaw only point out in print by Heaywood in 1890

Later proof by Appel and Haken was apparently correct, but gave rise to a new worry:

- How to assess the correctness of a proof where many explicit configurations are checked by a computer program?

## The four-colour Theorem

Early history indicates fallibility of the traditional social process:

- Proof claimed by Kempe in 1879
- Flaw only point out in print by Heaywood in 1890

Later proof by Appel and Haken was apparently correct, but gave rise to a new worry:

- How to assess the correctness of a proof where many explicit configurations are checked by a computer program?

In 2005, Georges Gonthier formalized the entire proof in Coq, making use of the "SSReflect" proof language and replacing ad-hoc programs by evaluation within the logical kernel.

## The odd-order theorem

## The odd-order theorem

- The fact that every finite group of odd order is solvable was a landmark result proved by Feit and Thompson in 1963.

## The odd-order theorem

- The fact that every finite group of odd order is solvable was a landmark result proved by Feit and Thompson in 1963.
- At the time it was one of the longest mathematical proofs ever published, and it plays a major part in the full classification of simple groups.

## The odd-order theorem

- The fact that every finite group of odd order is solvable was a landmark result proved by Feit and Thompson in 1963.
- At the time it was one of the longest mathematical proofs ever published, and it plays a major part in the full classification of simple groups.
- In 2012 a team led by Georges Gonthier completed a formalization in Coq, consisting of about $150,000$ lines of code.

## The odd-order theorem

- The fact that every finite group of odd order is solvable was a landmark result proved by Feit and Thompson in 1963.
- At the time it was one of the longest mathematical proofs ever published, and it plays a major part in the full classification of simple groups.
- In 2012 a team led by Georges Gonthier completed a formalization in Coq, consisting of about $150,000$ lines of code.
- A fairly extensive library of results in algebra was developed in the process, including Galois theory and group characters.

## The odd-order theorem

- The fact that every finite group of odd order is solvable was a landmark result proved by Feit and Thompson in 1963.
- At the time it was one of the longest mathematical proofs ever published, and it plays a major part in the full classification of simple groups.
- In 2012 a team led by Georges Gonthier completed a formalization in Coq, consisting of about $150,000$ lines of code.
- A fairly extensive library of results in algebra was developed in the process, including Galois theory and group characters.
- Uses the "SSReflect" proof language for Coq that was used in the four-colour proof.

## The Kepler conjecture

The *Kepler conjecture* states that no arrangement of identical balls in ordinary 3-dimensional space has a higher packing density than the obvious ‘cannonball’ arrangement.
Hales, working with Ferguson, arrived at a proof in 1998:

- 300 pages of mathematics: geometry, measure, graph theory and related combinatorics, . . .
- 40,000 lines of supporting computer code: graph enumeration, nonlinear optimization and linear programming.

Hales submitted his proof to *Annals of Mathematics* . . .

## The response of the reviewers

After a full four years of deliberation, the reviewers returned:

> *"The news from the referees is bad, from my perspective. They have not been able to certify the correctness of the proof, and will not be able to certify it in the future, because they have run out of energy to devote to the problem. This is not what I had hoped for.*
> *Fejes Toth thinks that this situation will occur more and more often in mathematics. He says it is similar to the situation in experimental science — other scientists acting as referees can't certify the correctness of an experiment, they can only subject the paper to consistency checks. He thinks that the mathematical community will have to get used to this state of affairs."*

## The birth of Flyspeck

Hales's proof was eventually published, and no significant error has been found in it. Nevertheless, the verdict is disappointingly lacking in clarity and finality.
As a result of this experience, the journal changed its editorial policy on computer proof so that it will no longer even try to check the correctness of computer code.
Dissatisfied with this state of affairs, Hales initiated a project called *Flyspeck* to completely formalize the proof.

## Flyspeck

Flyspeck = ‘Formal Proof of the Kepler Conjecture’.

> *“In truth, my motivations for the project are far more complex than a simple hope of removing residual doubt from the minds of few referees. Indeed, I see formal methods as fundamental to the long-term growth of mathematics. (Hales,* The Kepler Conjecture*)*

The formalization effort has been running for a few years now with a significant group of people involved, some doing their PhD on Flyspeck-related formalization.
In parallel, Hales has simplified the informal proof using ideas from Marchal, significantly cutting down on the formalization work.

## Flyspeck: current status

A large team effort led by Hales has brought Flyspeck close to completion:

## Flyspeck: current status

A large team effort led by Hales has brought Flyspeck close to completion:

- Essentially all the ordinary mathematics has been formalized in HOL Light: Euclidean geometry, measure theory, *hypermaps*, *fans*, results on packings.

## Flyspeck: current status

A large team effort led by Hales has brought Flyspeck close to completion:

- Essentially all the ordinary mathematics has been formalized in HOL Light: Euclidean geometry, measure theory, *hypermaps*, *fans*, results on packings.
- The graph enumeration process has been verified (and improved in the process) by Tobias Nipkow in Isabelle/HOL.

## Flyspeck: current status

A large team effort led by Hales has brought Flyspeck close to completion:

- Essentially all the ordinary mathematics has been formalized in HOL Light: Euclidean geometry, measure theory, *hypermaps*, *fans*, results on packings.
- The graph enumeration process has been verified (and improved in the process) by Tobias Nipkow in Isabelle/HOL.
- A highly optimized way of formally proving the linear programming part in HOL Light has been developed by Alexey Solovyev, following earlier work by Steven Obua.

## Flyspeck: current status

A large team effort led by Hales has brought Flyspeck close to completion:

- Essentially all the ordinary mathematics has been formalized in HOL Light: Euclidean geometry, measure theory, *hypermaps*, *fans*, results on packings.
- The graph enumeration process has been verified (and improved in the process) by Tobias Nipkow in Isabelle/HOL.
- A highly optimized way of formally proving the linear programming part in HOL Light has been developed by Alexey Solovyev, following earlier work by Steven Obua.
- A method has been developed by Alexey Solovyev to prove all the nonlinear optimization results, though it still needs a lot of runtime to solve them all.

## Univalent Foundations

- Provers already use quite a variety of foundations, including variants of ZFC set theory (Mizar), higher-order logic (HOL and relatives), and constructive type theory (Coq).

## Univalent Foundations

- Provers already use quite a variety of foundations, including variants of ZFC set theory (Mizar), higher-order logic (HOL and relatives), and constructive type theory (Coq).
- Vladimir Voevodsky proposed a new “Homotopy Type Theory” to give ‘univalent’ foundations for mathematics, based on relations between homotopy and type theory.

## Univalent Foundations

- Provers already use quite a variety of foundations, including variants of ZFC set theory (Mizar), higher-order logic (HOL and relatives), and constructive type theory (Coq).
- Vladimir Voevodsky proposed a new “Homotopy Type Theory” to give ‘univalent’ foundations for mathematics, based on relations between homotopy and type theory.
- In some sense it allows isomorphic objects to be identified, formalizing an intuitive principle often used by mathematicians.

## Univalent Foundations

- Provers already use quite a variety of foundations, including variants of ZFC set theory (Mizar), higher-order logic (HOL and relatives), and constructive type theory (Coq).
- Vladimir Voevodsky proposed a new "Homotopy Type Theory" to give 'univalent' foundations for mathematics, based on relations between homotopy and type theory.
- In some sense it allows isomorphic objects to be identified, formalizing an intuitive principle often used by mathematicians.
- Voevodsky has led a major research effort resulting in new results, implementations in Coq and Agda, and a textbook.

## Univalent Foundations

- Provers already use quite a variety of foundations, including variants of ZFC set theory (Mizar), higher-order logic (HOL and relatives), and constructive type theory (Coq).
- Vladimir Voevodsky proposed a new "Homotopy Type Theory" to give 'univalent' foundations for mathematics, based on relations between homotopy and type theory.
- In some sense it allows isomorphic objects to be identified, formalizing an intuitive principle often used by mathematicians.
- Voevodsky has led a major research effort resulting in new results, implementations in Coq and Agda, and a textbook.

An encouraging feature of both Flyspeck and Univalent Foundations is that *the driving force behind each one is a major mainstream mathematician*.

## Formal Verification

## Formal verification

In most software and hardware development, we lack even *informal* proofs of correctness.

Correctness of hardware, software, protocols etc. is routinely "established" by testing.

However, exhaustive testing is impossible and subtle bugs often escape detection until it's too late.

The consequences of bugs in the wild can be serious, even deadly.

Formal verification (*proving* correctness) seems the most satisfactory solution, but gives rise to large, ugly proofs.

## Recent formal proofs in computer system verification

Some successes for verification using theorem proving technology:

- CompCert verified compiler from significant subset of the C programming language into PowerPC assembler (Xavier Leroy et al., Coq)
- Designed-for-verification version of L4 operating system microkernel (Gerwin Klein et al., Isabelle/HOL).

Again, these indicate that complex and subtle computer systems can be verified, but significant manual effort was needed, perhaps tens of person-years for L4.

## A diversity of activities

Intel is best known as a hardware company, and hardware is still the core of the company's business. However this entails much more:

- Microcode
- Firmware
- Protocols
- Software

## A diversity of activities

Intel is best known as a hardware company, and hardware is still the core of the company's business. However this entails much more:

- Microcode
- Firmware
- Protocols
- Software

If the Intel® Software and Services Group (SSG) were split off as a separate company, it would be in the top 10 software companies worldwide.

## A diversity of verification problems

This gives rise to a corresponding diversity of verification problems, and of verification solutions.

- Propositional tautology/equivalence checking (FEV)
- Symbolic simulation
- Symbolic trajectory evaluation (STE)
- Temporal logic model checking
- Combined decision procedures (SMT)
- First order automated theorem proving
- Interactive theorem proving

Most of these techniques (trading automation for generality / efficiency) are in active use at Intel.

## A spectrum of formal techniques

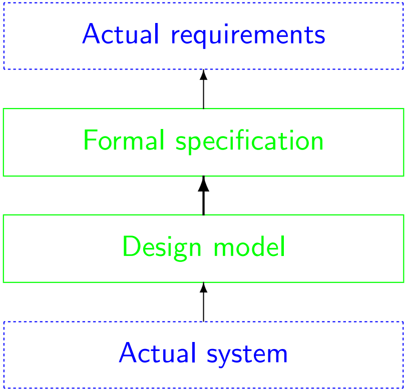
Traditionally, formal verification has been focused on complete proofs of functional correctness.
But recently there have been notable successes elsewhere for 'semi-formal' methods involving abstraction or more limited property checking.

- Airbus A380 avionics
- Microsoft SLAM/SDV

One can also consider applying theorem proving technology to support testing or other traditional validation methods like path coverage.
These are all areas of interest at Intel.

## Models and their validation

We have the usual concerns about validating our specs, but also need to pay attention to the correspondence between our models and physical reality.



## Physical problems

Chips can suffer from physical problems, usually due to overheating or particle bombardment (‘soft errors’).

## Physical problems

Chips can suffer from physical problems, usually due to overheating or particle bombardment (‘soft errors’).

- In 1978, Intel encountered problems with ‘soft errors’ in some of its DRAM chips.

## Physical problems

Chips can suffer from physical problems, usually due to overheating or particle bombardment (‘soft errors’).

- In 1978, Intel encountered problems with ‘soft errors’ in some of its DRAM chips.
- The cause turned out to be alpha particle emission from the packaging.

## Physical problems

Chips can suffer from physical problems, usually due to overheating or particle bombardment (‘soft errors’).

- In 1978, Intel encountered problems with ‘soft errors’ in some of its DRAM chips.
- The cause turned out to be alpha particle emission from the packaging.
- The factory producing the ceramic packaging was on the Green River in Colorado, downstream from the tailings of an old uranium mine.

## Physical problems

Chips can suffer from physical problems, usually due to overheating or particle bombardment (‘soft errors’).

- In 1978, Intel encountered problems with ‘soft errors’ in some of its DRAM chips.
- The cause turned out to be alpha particle emission from the packaging.
- The factory producing the ceramic packaging was on the Green River in Colorado, downstream from the tailings of an old uranium mine.

However, these are rare and apparently well controlled by existing engineering best practice.

## The FDIV bug

Formal methods are more useful for avoiding design errors such as the infamous FDIV bug:

## The FDIV bug

Formal methods are more useful for avoiding design errors such as the infamous FDIV bug:

- Error in the floating-point division (FDIV) instruction on some early Intel®Pentium® processors

## The FDIV bug

Formal methods are more useful for avoiding design errors such as the infamous FDIV bug:

- Error in the floating-point division (FDIV) instruction on some early Intel®Pentium® processors
- Very rarely encountered, but was hit by a mathematician doing research in number theory.

## The FDIV bug

Formal methods are more useful for avoiding design errors such as the infamous FDIV bug:

- Error in the floating-point division (FDIV) instruction on some early Intel®Pentium® processors
- Very rarely encountered, but was hit by a mathematician doing research in number theory.
- Intel eventually set aside US $475 million to cover the costs.

## The FDIV bug

Formal methods are more useful for avoiding design errors such as the infamous FDIV bug:

- Error in the floating-point division (FDIV) instruction on some early Intel®Pentium® processors
- Very rarely encountered, but was hit by a mathematician doing research in number theory.
- Intel eventually set aside US $475 million to cover the costs.

This did at least considerably improve investment in formal verification.

## Layers of verification

If we want to verify from the level of software down to the transistors, then it's useful to identify and specify intermediate layers.

## Layers of verification

If we want to verify from the level of software down to the transistors, then it's useful to identify and specify intermediate layers.

- Implement high-level floating-point algorithm assuming addition works correctly.

## Layers of verification

If we want to verify from the level of software down to the transistors, then it's useful to identify and specify intermediate layers.

- Implement high-level floating-point algorithm assuming addition works correctly.
- Implement a cache coherence protocol assuming that the abstract protocol ensures coherence.

## Layers of verification

If we want to verify from the level of software down to the transistors, then it's useful to identify and specify intermediate layers.

- Implement high-level floating-point algorithm assuming addition works correctly.
- Implement a cache coherence protocol assuming that the abstract protocol ensures coherence.

Many similar ideas all over computing: protocol stack, virtual machines etc.

## Layers of verification

If we want to verify from the level of software down to the transistors, then it's useful to identify and specify intermediate layers.

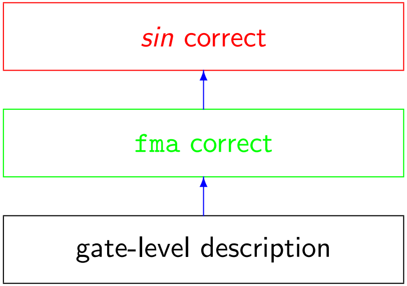
- Implement high-level floating-point algorithm assuming addition works correctly.
- Implement a cache coherence protocol assuming that the abstract protocol ensures coherence.

Many similar ideas all over computing: protocol stack, virtual machines etc.

If this clean separation starts to break down, we may face much worse verification problems...

## How some of our verifications fit together

For example, the `fma` behavior is the *assumption* for my verification, and the *conclusion* for someone else's.



But this is not quite trivial when the verifications use different formalisms!

## Our work

We have formally verified correctness of various floating-point algorithms.

- Division and square root (Marstein-style, using fused multiply-add to do Newton-Raphson or power series approximation with delicate final rounding).
- Transcendental functions like *log* and *sin* (table-driven algorithms using range reduction and a core polynomial approximations).

Proofs use the HOL Light prover

- `http://www.cl.cam.ac.uk/users/jrh/hol-light`

## Our HOL Light proofs

The mathematics we formalize is mostly:

- Elementary number theory and real analysis
- Floating-point numbers, results about rounding etc.

Needs several special-purpose proof procedures, e.g.

- Verifying solution set of some quadratic congruences
- Proving primality of particular numbers
- Proving bounds on rational approximations
- Verifying errors in polynomial approximations

## Example: tangent algorithm

- The input number $X$ is first reduced to $r$ with approximately $|r| \leq \pi/4$ such that $X = r + N\pi/2$ for some integer $N$. We now need to calculate $\pm tan(r)$ or $\pm cot(r)$ depending on $N$ modulo $4$.

## Example: tangent algorithm

- The input number $X$ is first reduced to $r$ with approximately $|r| \leq \pi/4$ such that $X = r + N\pi/2$ for some integer $N$. We now need to calculate $\pm tan(r)$ or $\pm cot(r)$ depending on $N$ modulo $4$.
- If the reduced argument $r$ is still not small enough, it is separated into its leading few bits $B$ and the trailing part $x = r - B$, and the overall result computed from $tan(x)$ and pre-stored functions of $B$, e.g.

$$tan(B + x) = tan(B) + \frac{\frac{1}{sin(B)cos(B)} tan(x)}{cot(B) - tan(x)}$$

## Example: tangent algorithm

- The input number $X$ is first reduced to $r$ with approximately $|r| \leq \pi/4$ such that $X = r + N\pi/2$ for some integer $N$. We now need to calculate $\pm tan(r)$ or $\pm cot(r)$ depending on $N$ modulo $4$.
- If the reduced argument $r$ is still not small enough, it is separated into its leading few bits $B$ and the trailing part $x = r - B$, and the overall result computed from $tan(x)$ and pre-stored functions of $B$, e.g.

$$tan(B + x) = tan(B) + \frac{\frac{1}{sin(B)cos(B)} tan(x)}{cot(B) - tan(x)}$$

- Now a power series approximation is used for $tan(r)$, $cot(r)$ or $tan(x)$ as appropriate.

## Overview of the verification

To verify this algorithm, we need to prove:

## Overview of the verification

To verify this algorithm, we need to prove:

- The range reduction to obtain $r$ is done accurately.

## Overview of the verification

To verify this algorithm, we need to prove:

- The range reduction to obtain $r$ is done accurately.
- The mathematical facts used to reconstruct the result from components are applicable.

## Overview of the verification

To verify this algorithm, we need to prove:

- The range reduction to obtain $r$ is done accurately.
- The mathematical facts used to reconstruct the result from components are applicable.
- Stored constants such as $tan(B)$ are sufficiently accurate.

## Overview of the verification

To verify this algorithm, we need to prove:

- The range reduction to obtain $r$ is done accurately.
- The mathematical facts used to reconstruct the result from components are applicable.
- Stored constants such as $tan(B)$ are sufficiently accurate.
- The power series approximation does not introduce too much error in approximation.

## Overview of the verification

To verify this algorithm, we need to prove:

- The range reduction to obtain $r$ is done accurately.
- The mathematical facts used to reconstruct the result from components are applicable.
- Stored constants such as $tan(B)$ are sufficiently accurate.
- The power series approximation does not introduce too much error in approximation.
- The rounding errors involved in computing with floating point arithmetic are within bounds.

## Overview of the verification

To verify this algorithm, we need to prove:

- The range reduction to obtain $r$ is done accurately.
- The mathematical facts used to reconstruct the result from components are applicable.
- Stored constants such as $tan(B)$ are sufficiently accurate.
- The power series approximation does not introduce too much error in approximation.
- The rounding errors involved in computing with floating point arithmetic are within bounds.

Most of these parts are non-trivial. Moreover, some of them require more pure mathematics than might be expected.

## Why mathematics?

Controlling the error in range reduction becomes difficult when the reduced argument $X - N\pi/2$ is small.
To check that the computation is accurate enough, we need to know:

> *How close can a floating point number be to an integer multiple of $\pi/2$?*

Even deriving the power series (for $0 < |x| < \pi$):

$$cot(x) = 1/x - \frac{1}{3}x - \frac{1}{45}x^3 - \frac{2}{945}x^5 - \ldots$$

is much harder than you might expect.

## Why HOL Light?

We need a general theorem proving system with:

- High standard of logical rigor and reliability
- Ability to mix interactive and automated proof
- Programmability for domain-specific proof tasks
- A substantial library of pre-proved mathematics

Other theorem provers such as ACL2, Coq and PVS have also been used for verification in this area (see other talks here).

## Conclusions

- Formal proof is still rather painstaking, but computers are making it relatively less difficult.
- More and more mathematics is being successfully formalized, including some from the 20th century and even near-contemporary results like Kepler.
- Formalizing mathematics is not disjoint from formal verification but can help to support it by building up background knowledge.