Séminaire BOURBAKI June 2014
66ème année, 2013-2014, n$^{\text{o}}$ 1086

DEVELOPMENTS IN FORMAL PROOFS

by **Thomas C. HALES**

Si la mathématique formalisée était aussi simple que le jeu d'échecs, . . . il n'y aurait plus qu'à rédiger nos démonstrations dans ce langage, comme l'auteur d'un traité d'échecs écrit dans sa notation. . . . Mais les choses sont loin d'être aussi faciles, et point n'est besoin d'une longue pratique pour s'apercevoir qu'un tel projet est absolument irréalisable. – Bourbaki, 1966 [16].

A proof assistant is interactive computer software that humans use to prepare scripts of mathematical proofs. These proof scripts can be parsed and verified directly from the fundamental rules of logic and the foundational axioms of mathematics. The technology underlying proof assistants and formal proofs has been under development for decades and grew out of efforts in the early twentieth century to place mathematics on solid foundations. Proof assistants have been built upon various mathematical foundations, including Zermelo-Fraenkel set theory (Mizar), Higher Order Logic (HOL), and dependent type theory (Coq) [50], [37], [14]. A *formal proof* is one that has been verified from first principles (generally by computer).

This report will focus on three particular technological advances. The HOL Light proof assistant will be used to illustrate the design of a highly reliable system. Today, proof assistants can verify large bodies of advanced mathematics; and as an example, we will turn to the formal proof in Coq of the Feit-Thompson Odd Order theorem in group theory. Finally, we will discuss advances in the automation of formal proofs, as implemented in proof assistants such as Mizar, Coq, Isabelle, and HOL Light.

1. BUILDING A TRUSTWORTHY SYSTEM WITH HOL LIGHT

HOL Light is a lightweight implementation of a foundational system based on Higher Order Logic (HOL). Because it is such a lightweight system, it is a natural system to use for explorations of the reliability of formal proof assistants.

1.1. Naive type theory

HOL, the foundational system of mathematics that we describe in this section, is based on a simply typed λ-calculus. This subsection describes a simple type theory in naive terms.

A salient feature of set theory is that it is so amorphous; everything is a set: ordered pairs are sets, elements of sets are sets, and functions between sets are sets. Thus, it is meaningful in set theory to ask bizarre questions such as whether a Turing machine is a minimal surface. In type theory, the very syntax of the language prohibits this question. Computer systems benefit from the extra structure provided by types.

Naively, a simple type system is a countable collection of disjoint nonempty sets called types. The collection of types satisfies a closure property: for every two types A and B, there is a further type, denoted $A \to B$, that can be identified with the set of functions from A to B.

In addition to types, there are terms, which are thought of as elements of types. Each term t has a unique type A. This relationship between a term and its type is denoted $t : A$. In particular, $f : A \to B$ denotes a term f of type $A \to B$.

There are variables that range over types called *type variables*, and another collection of variables that run over terms.

1.2. Models of HOL

The naive interpretation of types as sets can be made precise. We build a model of HOL in Zermelo-Fraenkel-Choice (ZFC) set theory to prove that HOL is consistent assuming that ZFC is. In this section, we review this routine exercise in model theory. At the same time, we will give some indications of the structure of HOL Light. See [36] for a more comprehensive introduction to HOL Light.

The interpretation of variable-free types as sets is recursively defined. We use a superscript M to mark the interpretation of a type as a set. Specifically, the types in HOL are generated by the boolean type `bool` (which we interpret as a set $\texttt{bool}^M = \{\top, \bot\}$ of cardinality two with labeled elements representing true and false) and the infinite type I (which we interpret as a countably infinite set I^M). Recursively, for any two variable-free types A and B, the type $A \to B$ is interpreted as the set $(A \to B)^M$ of all functions from A^M to B^M. We can arrange that the sets interpreting these types are all disjoint.

In summary so far, we fix an interpretation M, determining a countable collection $\mathcal{T} = \{A^M\}$ of nonempty sets in ZFC. We now extend our interpretation M to a *valuation* $v = (M, v_1, v_2)$, where v_1 is a function from the set of type variables in HOL to $\mathcal{T}$, and v_2 is a function from the set of term variables in HOL to $\bigcup \mathcal{T}$. The valuation v extends recursively to give a mapping that assigns a set $A^v \in \mathcal{T}$ to every type A. We require v_2 to be chosen so that whenever x is a variable of type A, then $x^{v_2} \in A^v$. The valuation v extends recursively to give a mapping on all terms:

$$t \mapsto t^v \in A^v \in \mathcal{T}, \quad \text{for all } t : A.$$

For example, for every type A, there is a HOL term $(=)$ of type $A \to (A \to \texttt{bool})$ representing equality for that type.[(1)] This term is interpreted as the function in $(A \to (A \to \texttt{bool}))^v$ that maps $a \in A^v$ to the delta function δ_a supported at a (where the support of the function means the preimage of $\top$).

A *sequent* is a pair (L, t), traditionally written $L \vdash t$, where L is a finite set of terms called the *assumptions*, and t is a term called the *conclusion*. The terms of L and t must all have type `bool`. If L is empty, it is omitted from the notation.

If L is a finite set of boolean terms, and if v is a valuation extending M, write L^v for the corresponding set of elements of the set $\texttt{bool}^M$. We say a sequent $L \vdash t$ is *logically valid* if for every valuation v for which every element of L^v is $\top \in \texttt{bool}^M$, we also have $t^v = \top$ in $\texttt{bool}^M$.

A *theorem* in HOL is a sequent that is generated from the mathematical axioms and rules of logic. There is a constant `FALSE` in HOL. The following amounts to saying that HOL does not prove `FALSE`.

THEOREM 1. — *If ZFC is consistent, then HOL is consistent.*

Proof sketch. — We give the proof in ZFC. Here, HOL is treated purely syntactically as a set of strings in a formal language.

We run through the rules of logic of HOL one by one and check that each one preserves validity.[(2)] For example, the reflexive law of equality in HOL states that for any term t of any type A, we have a theorem $\vdash t = t$. By the interpretation of equality described above, under any valuation v, this equation is interpreted as the value $\delta_{t^v}(t^v) \in \texttt{bool}^M$, which is $\top$. Hence the reflexive law preserves validity. The other rules (transitivity of equality, and so forth) are checked similarly.

We may well-order each set in the collection $\mathcal{T}$. HOL posits a choice operator of type $(A \to \texttt{bool}) \to A$ for every type A. The well-ordering allows us to interpret HOL's choice operator as an operator that maps a function $f \in (A \to \texttt{bool})^v$ with nonempty support to the minimal element of its support.

We run through the mathematical axioms of HOL and check their validity. There are only three. The *axiom of infinity* positing the existence of an infinite type I is logically valid by our requirement to interpret I as a countably infinite set. The *axiom of choice* is logically valid by the well-ordering we have placed on the sets A^M. The *axiom of extensionality* also holds in this model, because it holds for sets. Thus, every axiom is logically valid. Since all axioms are logically valid and every rule of inference preserves validity, every theorem is logically valid.

1. The convention in HOL is to curry functions: using the bijection $X^{Y \times Z} = (X^Z)^Y$ to write a function whose domain is a product as a function of a single argument taking values in a function space. In particular, equality is a curried function of type $A \to (A \to \texttt{bool})$ rather than a relation on $A \times A$.

2. There are ten such rules, giving the behavior of equality, λ-abstractions, β-reduction, and the discharge of assumptions. For reference purposes, an appendix lists the inference rules of HOL. The analysis in this section omits the rules for the creation of new term constants and types.

There is a boolean constant `FALSE` in HOL. It is defined to be the term $\forall p.\ p$. An easy calculation based on its definition gives that $\texttt{FALSE}^v = \perp$ for every valuation v. Hence, $\vdash$ `FALSE` is not a logically valid sequent, and not a theorem. This proves HOL consistent. □

1.3. Computer implementation

The bare consistency proof is just the beginning. We can push matters much further when the logic is implemented in computer code.

The HOL Light system is implemented in the Objective Caml programming language, which is one of the many dialects of the ML language. The language ML (an acronym for Meta Language) was originally designed as a metalanguage to automate mathematical proof commands [27]. It is significant that the development of the language and the development of proof assistants have progressed hand in hand, with many of the same researchers participating in language design and formal proofs. The result is a programming language that can stand up to intense mathematical scrutiny.

This parallel development of ML and proof assistants also means that there are striking similarities between the syntax of ML and the syntax of HOL.[3] The code listing shows a few parallels between OCaml syntax and HOL syntax.

OCaml	versus	**HOL**
-----		---
3: int		3:num
[0;1;2;3]		[0;1;2;3]
let x = 3 **and** y = 4 **in** x + y		**let** x = 3 **and** y = 4 **in** x + y
map (fun x −> x + 1) [0;1;2]		map (\x. x + 1) [0;1;2]

In both ML and HOL every term has a type, and functions $f : A \to B$ of the right type are required to convert from a term of one type A to another B. One of Milner's key ideas in the design of ML was to have an abstract datatype representing theorems. The strict type system of the language ML prevents the construction of any theorems except through a carefully secured kernel that expresses the axioms and rules of inference. In a formal proof in HOL, absolutely every theorem – no matter how long or how complex – is checked exhaustively by the kernel.

1.4. Verification of the code that implements HOL Light

The code in the kernel that expresses the rules of HOL is of critical importance. Even a minor bug in the kernel might be exploited to create an inconsistent system. Fortunately, there are good reasons to believe that the kernel does not have a single bug.

1. The kernel is remarkably small. It takes only about 400 lines of computer code to express all of the kernel functions, including the type system, the term constructors,

3. HOL is a descendant of the LCF theorem prover that spurred the development of ML.

sequents, the rules of inference, the axioms, and theorems. For example, it only takes seven lines of computer code to describe the datatypes for HOL types, terms, and theorems, as shown in the following listing of code [37].

```
type hol_type = Tyvar of string
              | Tyapp of string *  hol_type list

type term = Var of string * hol_type
          | Const of string  * hol_type
          | Comb of term * term
          | Abs of term * term

type thm = Sequent of (term list * term)
```

2. The code has been written in a clean, readable style and has been scrutinized by many computer scientists, logicians, and mathematicians (including me).

3. The correctness of the kernel has been formally verified, using the HOL Light proof assistant itself (extended by a large cardinal) [33]. (4) Specifically, a model of HOL can be built inside HOL itself along the same lines as the model of HOL in ZFC described above.

This formal verification of HOL in HOL goes further than the construction of a model. It also checks that the code implementing the logic is bug free. The code verification is based on the parallels mentioned above between the metalanguage and HOL itself, allowing the OCaml source code for the HOL kernel to be translated back into HOL for verification. A stricter standard of code verification, based on the semantics of the programming language, is discussed in the next subsection. The proof of HOL in HOL removes most practical doubts about the correctness of the kernel.

As independent corroboration, the correctness proof of the kernel of HOL Light has been automatically translated into the HOL Zero and HOL4 assistants and reverified there [1].

1.5. HOL in machine code

The formal verification of HOL in HOL does not settle the issue of trust once and for all. The reliability of the proof assistant ultimately rests on the entire computer environment in which the software operates, including the semantics of programming languages, compilers, operating systems, and hardware. These issues should be a concern of every mathematician who cares about the foundations of mathematics at a time when the practice of mathematics is gradually migrating to computers.

4. A large cardinal axiom gives the existence of a large type corresponding to the set $\bigcup \mathcal{T}$ that we used above in the construction of a model. By Gödel, we do not expect to construct a model of HOL in HOL except by adding an axiom to strengthen the system.

The current working goal of researchers is to create an unbroken formally-verified chain extended from the HOL Light kernel all the way down to machine code. Most of the links in the chain have been forged. (5)

CakeML is a dialect of ML with mathematically rigorous operational semantics. According to its designers, “Our overall goal for CakeML is to provide the most secure system possible for running verified software and other programs that require a high-assurance platform” [46]. The CakeML team has improved the HOL in HOL verification [53], [52], [45]. This work closes various gaps in the verification of the OCaml implementation of HOL, such as *object magic* (a mechanism that defeats the OCaml type system) and *mutable strings* (which allow a theorem to be edited to state something different from what was proved). The HOL system can be extended by adding new definitions and new types. The formal verification now covers these extensions. It verifies the soundness of an implementation of the HOL kernel in CakeML, according to the formally specified operational semantics of CakeML.

In related work, a verified compiler has been constructed for the CakeML language [46], [59]. This brings us close to end-to-end verification of HOL Light, from its high-level logical description down to execution in machine code.

1.6. Disclaimers

At the conclusion of this section, we make the usual disclaimers. Without exception, all physical devices are prone to failure. Soft errors (typically caused by alpha particle interactions between memory and its environment) produce a steady stream of errors depending on complex factors such as hardware architecture and the height above sea level at which the calculation is performed. A formal proof in HOL of the correctness of HOL carries the evident dangers of self-justification. Proofs of correctness are made relative to mathematically precise idealized descriptions of things rather than the physical objects themselves.

Notwithstanding all these issues, formalization reduces defect rates in proofs to levels that are simply not possible by any other available process. Now that the formal proof of HOL in HOL has been translated into other systems, the dangers of self-justification are minimal. Overall, formalized mathematics can now claim to be orders of magnitude more reliable than traditionally refereed papers.

---

5. A brilliant success has been the construction of a formally verified C compiler [47]. Another remarkable project is the full formal verification of an operating system kernel [22]. Concerning the formal verification of Coq and Milawa (an ACL2-like system), see [11] [51]. In this survey article, we focus on the work done on formal verification related to the ML programming language, because it fits more closely with our narrative of building a trustworthy system in HOL.

2. ADVANCED DEVELOPMENTS IN COQ: THE ODD ORDER THEOREM

In the previous section, we discussed the construction of a reliable proof assistant. In this section, we turn to a different proof assistant, Coq, and look at the formalization of group theory. We warn the reader that the Coq system, which is based on the Calculus of Inductive Constructions, is significantly different from the HOL system in the previous section [64], [18].

Feit and Thompson published their famous theorem in 1963 [20].

THEOREM 2 (The Odd Order Theorem, Feit-Thompson). — *All finite groups of odd order are solvable.*

```
Theorem Feit_Thompson (gT : finGroup Type) (G : {group gT}) :
  odd #|G| -> solvable G.
```

Solomon writes this about the significance of the Odd Order theorem, "This short sentence and its long proof were a moment in the evolution of finite group theory analogous to the emergence of fish onto dry land. Nothing like it had happened before; nothing quite like it has happened since" [62].

The Odd Order paper broke through various barriers that cleared the way for a remarkably fruitful massive research collaboration that eventually led to the classification of finite simple groups. Significantly, the 255 page Odd Order proof triggered an avalanche of long complex proofs related to the classification, culminating in the 1221 page classification of quasi-thin groups by Aschbacher and Smith [5].

Background material for the proof appears in textbooks *Finite Groups* [28], *Finite Group Theory* [3], and *Character Theory of Finite Groups* [38]. The necessary background includes a basic graduate-level understanding of rings, modules, linear and multilinear algebra (including direct sums, tensor products and determinants); fields, algebraic closures, and basic Galois theory; the structure theorems of Sylow and Hall; Jordan-Hölder; Wedderburn's structure theorem for semisimple algebras; representation theory with induced representations, Schur's lemma, Clifford theory, and Maschke's theorem; and character theory including Frobenius reciprocity, and orthogonality.

I will not say much about the actual proof of the Odd Order theorem. We have now had more than fifty years to assimilate the ideas of the proof. There are numerous surveys of the proof [28, p. 450], [23], [65], [62]. The original proof of Feit and Thompson was later reworked and simplified in two books [13], [56].

In very brief terms, the proof starts by assuming a minimal counterexample to the statement. This counterexample will be a finite simple group G of odd order in which every proper subgroup is solvable. Each maximal subgroup of G is *p-local*; that is, the normalizer of a nontrivial subgroup of G of p-power order, for some prime p. The first major part of the proof of the Odd Order theorem consists in establishing restrictions on the structure of the maximal subgroups and their embeddings into G. In the special case when G is a *CN-group* (a group whose centralizers of non-identity elements are all nilpotent), the maximal subgroups are *Frobenius groups* [19]. A Frobenius group is a nontrivial semidirect product $K \rtimes H$, where H is disjoint from its conjugates, and H is its own normalizer [28, Th. 7.7]. In the general case, the strategy is to prove that as many of the maximal subgroups as possible are as close as possible to being Frobenius groups. This strategy encounters many exceptions and detours, but eventually the local analysis shows that the maximal subgroups are mostly Frobenius-like [13, Sec. 16].

The second major portion of the proof uses the complex character theory of G to obtain inequalities over the real numbers that restrict and ultimately eliminate all possibilities for G.

The final part of the proof uses generators and relations to remove a special case in which there are maximal subgroups isomorphic to the group of all permutations of a finite field $\mathbb{F}$ of the form $x \mapsto a\,\sigma(x) + b$, where σ is a field automorphism, $a, b \in \mathbb{F}$, and a is an element of norm 1. It is in this part of the proof that Galois theory is most relevant.

2.1. Formal verification

The Odd Order theorem has been formally verified in the Coq proof assistant by a team led by Gonthier [26]. This is an extraordinary milestone in the history of formal proofs. What is particularly significant about the formalization itself?

1. The Odd Order theorem itself has never been seriously questioned, but the premature announcement of the classification of finite simple groups drew sustained criticism. Gorenstein wrote, "In February 1981, the classification of the finite simple groups was completed" [28, page 1]; and yet essential work on the classification continued for decades after that date [4]. Serre wrote in 2004 that "for years I have been arguing with group theorists who claimed that the 'Classification Theorem' was a 'theorem', i.e. had been proved" [57]. Because of the prominence of the Odd Order theorem within the classification, this formal proof plants a flag in the middle of the larger classification project.

2. Traditional methods of refereeing mathematical research become strained, when proofs are unusually long or computer assisted. There is a Wikipedia page listing numerous proofs in mathematics that set records for length [69]. Long papers cluster in certain areas such as finite group theory related to the classification, the Langlands program, and graph theory. The Odd Order theorem, the Four-Color theorem (Gonthier's previous formalization project), and the Kepler conjecture are all on that list. These three recent formalization projects send a clear defiant message to mathematicians: no matter how long or how complex your mathematical proofs may be, we can formalize them.

Absolutely no technological barriers prevent the formalization of large parts of the mathematical corpus. The issues now are how to make the technology more efficient, cost effective, and user friendly.

2.2. Constructive proof of the Odd Order theorem

The formal proof of the Odd Order theorem is based on the second-generation proof described in two books cited above [13], [56]. If we were to translate the formal proof of Odd Order back into a humanly readable book, the most notable difference would be that the original proof uses classical logic, but the formal proof uses constructive logic. In particular, a constructive proof avoids proofs by contradiction and does not invoke the law of excluded middle $\phi \vee \neg\phi$ as a general principle.

Several different strategies were used to translate the paper proof into a constructive computer proof.

1. Often, mathematicians use proof by contradiction out of sheer laziness when a direct proof would work equally well. "Let G be the finite group of minimal order that is a counterexample" is replaced with an induction on the order of the group, and so forth.

2. In [56], the character theory of finite groups relies heavily on vector spaces over $\mathbb{C}$ and complex conjugation. In the constructive formal proof, the corresponding vector spaces over an algebraic closure $\bar{\mathbb{Q}}$ are used. An algebraic closure of $\mathbb{Q}$ is obtained as the union of an increasing tower of number fields $\mathbb{Q}(\alpha_i)$, for $i \in \mathbb{N}$, each an explicitly constructed splitting field over the preceding one. Complex conjugation is replaced with conjugation with respect to a maximal real subfield of $\bar{\mathbb{Q}}$, also constructed as an explicit union of real subfields of finite degree over $\mathbb{Q}$.

3. Some constructions are justified by the decision procedure for the first-order theory of algebraically closed fields [17], [35]. For example, the socle of a module is defined as the sum of its simple submodules. This decision procedure gives a constructive test for the simplicity of a submodule, leading to a constructive definition of the socle (for a finitely generated $k[G]$-module where G is a finite group and k is an algebraically closed field) [25].

On a related note, the construction of a simple submodule of a given module M requires choice. By confining attention to modules that are countable as sets, countable choice suffices, which is provable in Coq from the well-foundedness of the natural numbers.

4. Various intermediate results in Odd Order theorem use classical logic, but the use of classical logic is eliminated from the final statement of the Odd Order theorem. The wedding of constructive and classical logics is arranged through a predicate *classically* ϕ that marks every result proved by classical logic. The definition of *classically* ϕ is

$$\text{classically } \phi : \mathit{Prop} := \forall b : \texttt{bool}, (\phi \to b) \to b.$$

When ϕ is quantifier free, this is essentially the usual double negation translation of a classical formula into constructive logic. The definition of *classically* works in such a way that ϕ may be used as an assumption in the proof of any boolean goal b, whenever *classically* ϕ is known.

5. Function extensionality does not hold in Coq; that is, it is not provable that two functions are equal, if they are provably equal on every input. This creates various complications. There are no general quotient types: types corresponding to the set of classes on a type under an equivalence relation. Instead, Coq introduces *setoids* – a set together with an equivalence relation on that set – without passing to the quotient. Unfortunately, setoids carry a certain overhead, which was not acceptable in the Odd Order proof, which makes ubiquitous use of quotient groups.

The group theory is developed in the context of *finite types* with *decidable equality*; (that is, the type is equipped with boolean procedure that decides whether any two elements of that type are equal). In this context, function extensionality holds and genuine quotients groups can be formed.

6. For all its advantages, at times the type theory of Coq is best forgotten. In particular, in arguments involving a finite number of finite groups, there would be excessive overhead in a separate type for each group, a separate binary operation on each group, and explicit homomorphisms embedding subgroups into groups. For such arguments, the finitely many finite groups are often considered as subgroups of a larger ambient finite group with a common binary operation and type. These ambient group arguments are one of the biggest stylistic differences between the Coq proof and the original.

2.3. Library of abstract algebra

Most of the work for this project went into the development of the libraries in abstract algebra and the related computer infrastructure. The libraries include formal proofs of all the necessary background material described at the beginning of this section, from Frobenius reciprocity to Wedderburn's theorem. From a software engineering point of view, it has been a major undertaking to get the computer to understand algebra at a level comparable to that of a working mathematician, and for it all to be formally justified. This includes much of the implicit domain knowledge that is required to read a proof. When we write $g * h$, we expect the computer to infer the correct binary operation. When we take the intersection $K \cap H$ of two subgroups, we expect it to go without saying that the intersection is again a subgroup.

3. AUTOMATING FORMALIZATION

One way to make proof assistants more usable is to increase the amount of automation. Ideally, the human should provide the high level structure of a proof as it is done in a traditionally published paper, and the computer should use search algorithms to insert the low-level reasoning. As technology has developed, computers have become capable of providing more and more of the low-level reasoning. For an overview of automated reasoning in formal proofs, see [35].

As we explained in the first section of this report, it is a matter of great importance for the underlying logic of the proof assistant to be sound and for the implementation in code to be free of bugs. Proof search algorithms are based on fundamentally different design considerations than the proof assistants themselves. With search algorithms, we can take a more relaxed approach, with the understanding that their results must eventually be verified by the proof assistant before being admitted as part of formal proof.

There is no general decision procedure for mathematics as a whole. Some decision procedures such as the Presburger algorithm for the additive first-order theory of natural numbers or quantifier elimination for real closed fields only have a limited practical value for formal proofs because they are so slow.

There are dangers in combining partial automation with human interaction. There is a psychological tendency for the human to "wait and see" rather than race against the computer, when the human and computer are both engaged in the same task. An automated procedure might make sudden complicated changes to the proof state that strand the user in uncharted territory. For these reasons, it is good to have procedures that fail quickly when they fail and that take a fully-documented circumscribed step forward when they succeed.

3.1. First-order logic

A *first-order formula with equality in the predicate calculus* is a formula built from variables $x, y, \ldots$, logical operations $\neg, \wedge, \Rightarrow, \ldots$, n-ary function symbols $f, g \ldots$, n-ary predicate symbols $P, Q, \ldots$ and quantified variables $\forall x, \exists y, \ldots$, according to syntactic rules that we will not spell out here. For example,

$$((\exists x.\ P(x)) \Rightarrow (\forall y.\ Q(y))) \Leftrightarrow (\forall x \forall y.\ P(x) \Rightarrow Q(y))$$

is a first-order formula. In contrast to higher-order logics discussed earlier, in a first-order formula, quantifiers over function symbols and predicate symbols are not allowed.

We are interested in algorithms that are *refutation complete*; that is, given the input of an unsatisfiable first-order formula, the algorithm outputs a proof of its unsatisfiability. A refutation complete algorithm can also produce proofs of logically valid formulas, by refuting the negation of the formula.

What has been understood for a long time are rules of inference (*resolution* together with special inferences that deal with equality) that are refutation complete. Recent research seeks to find efficient ways to manage and contain the vast explosion of clauses that can result by repeated application of the inference rules. Poorly designed algorithms exhaust available memory before a refutation is found.

Today, automated first-order theorem provers can handle problems with thousands of axioms [10], [35]. In recent years, the *Vampire* theorem prover has dominated the annual competition, with other strong contenders such as the *SPASS* and *E* theorem provers [44], [58], [68], [60].

There are some automated theorem provers based on higher-order logic but the most prevalent practice is to translate problems from higher-order logic into first-order logic, solve them there, then translate the answers back into higher-order logic, with an automated reconstruction of a formal proof inside the proof assistant. (6)

An early example of translating proofs in this way is Harrison's MESON procedure (based on model elimination), implemented in HOL Light. In this approach, the human supplies all the relevant theorems, and the automated procedure generates the logical glue that combines the given theorems into a proof.

Paulson advocates shipping the goal together with large libraries of theorems to a first-order theorem prover, and letting it figure out which of many theorems to use in the proof [54]. The human is no longer forced to search through the libraries to find the relevant theorems. The method is called a *sledgehammer* for its ability to deliver a powerful blow and its complete lack of finesse.

In a refinement of this approach, an automated heuristic procedure selects a few hundred theorems deemed to be the most relevant and ships those to the theorem prover. For example, to prove a new trig identity, we might select other trig identities as likely to be relevant. There is an art to selecting relevant premises wisely, and machine learning algorithms can be trained to do this effectively [67], [41], [40], [2].

Here is one simple example of a theorem proved in this way [42]. The theorem states that the convex hull of any three points in $\mathbb{R}^3$ is a set of measure zero. The automated procedure was able to comb through large libraries of previously established theorems to locate the following relevant facts: the convex hull is a subset of the affine hull; the affine hull of three points in $\mathbb{R}^3$ is a set of measure zero; and a subset of a set of measure zero has measure zero. The procedure then combines these facts with the necessary logic to give a fully automated formal proof.

Sledgehammers and machine learning algorithms have led to visible success. Fully automated procedures can prove 40% of the theorems in the Mizar math library, 47% of the HOL Light/Flyspeck libraries, with comparable rates in Isabelle [39], [42], [15]. These automation rates represent an enormous savings in human labor.

3.2. SAT Solvers and SMT

Boolean satisfiability (SAT) solvers implement algorithms that test for the satisfiability of formulas in propositional logic [49]. Is there an assignment of truth values to propositional variables for which a given formula evaluates to true? SMT (*satisfiability modulo theories*) algorithms combine the propositional reasoning of SAT solvers with reasoning within a given theory [12]. The input to the algorithm is a propositional formula in which the boolean variables have been replaced with predicates from the

6. There are many issues that come up in translating formulas in higher-order logic into first-order logic that I will not discuss here. In particular, some of the procedures might not be logically sound.

given first-order theory. An SMT algorithm searches for a valuation of the predicates in the theory that satisfies the propositional formula.

For example, the following calculation comes up in the proof of the Odd Order theorem. Let G be a finite group, and let (β_{ij}) be a matrix of virtual characters, with each entry a linear combination $\pm\chi \pm \chi' \pm \chi''$ of three distinct irreducible characters. Assume the matrix has at least four rows and two columns. Assume the inner product relations

$$\langle \beta_{ij}, \beta_{i'j'} \rangle = \delta_{ii'} + \delta_{jj'}, \quad \text{for } (i,j) \neq (i',j'),$$

with respect to the usual inner product on class functions with an orthonormal basis consisting of irreducible characters. The conclusion is that the virtual characters in each column of the matrix have a common irreducible character as constituent with a common sign.

A moment's reflection reveals that it is enough to show the conclusion for arbitrary 4×2 submatrices. On each 4×2 block, up to renaming the irreducible characters, a finite enumeration gives all ways to express the entries β_{ij} as a signed combination of three irreducibles. Thus, the proof reduces to a case analysis. Symmetry arguments further reduce the number of cases. The case analysis is done by hand in [56]. In the formal proof, Théry programmed the claim into a quantifier-free problem with uninterpreted symbols in an SMT solver. The SMT solution was then transcribed into a formal proof in Coq [26]. We list a code snippet from the formal proof of the Odd Order theorem. The first line of code asserts unsatisfiability: no irreducible character χ_1 appears with positive coefficient in β_{ii} and negative coefficient in β_{ji}.

```
Let unsat_J : unsat ⊨ & x1 in b11 & −x1 in b21.
Let unsat_II: unsat ⊨ & x1, x2 in b11 & x1, x2 in b21.
```

3.3. Other forms of automation

Automation in proof assistants takes many forms, including the evaluation of arithmetic expressions, the verification of polynomial and vector space identities, and decision procedures for linear real arithmetic.

Gröbner basis algorithms provide a particularly useful tool to prove general ring identities. If S is any system of equalities and inequalities of polynomials that holds in every integral domain of characteristic zero, then it can be transformed into a Gröbner basis problem over $\mathbb{Q}$. Buchberger's algorithm has been implemented in many of the major proof assistants. For example, one line of code suffices to generate a formal proof of the following isogeny of elliptic curves over $\mathbb{R}$. The symbol `&` denotes the function embedding $\mathbb{N}$ into $\mathbb{R}$.

$$\begin{aligned}
\vdash \quad & a' = \&2 * a \wedge b' = a * a - \&4 * b \wedge \\
& x_2 * y_1 = x_1 \wedge y_2 * y_1^2 = \&1 - b * x_1^4 \wedge \\
& y_1^2 = \&1 + a * x_1^2 + b * x_1^4 \\
& \Longrightarrow y_2^2 = \&1 - a' * x_2^2 + b' * x_2^4
\end{aligned}$$

Other powerful forms of automation in proof assistants include the certification of linear programs and nonlinear inequalities over the real numbers [63].

4. FINAL REMARKS

The aim of this report has been to describe some of the recent developments in formal proofs. Space and time do not permit a comprehensive survey, but in this final section, I briefly mention a few other projects.

4.1. Homotopy Type Theory

My report will be directly followed by a report by Coquand on dependent types and the univalence axiom, so I will be brief in my remarks on homotopy type theory.

Homotopy type theory (HoTT) is a foundational system for mathematics that includes dependent type theory, the univalence axiom, and higher inductive types. Introductions to homotopy type theory can be found at [66], [55]. For models of HoTT, see [9], [43]. HoTT has set quotients and function extensionality, giving remedies to some of Coq's nuisances.

It goes without saying that as mathematicians, we construct the ground on which we stand; the foundations of mathematics are of our choosing, subject to only mild constraints such as plausible consistency, expressive power, and a community of users. In particular, nothing but our own limited imaginations prevents us from relocating the foundations much closer to home.

By being a foundational system that is close to the actual practice of homotopy theory, HoTT makes the formalization of this branch of mathematics surprisingly refreshing. In the last two years many new formal proofs and constructions have been obtained: loop spaces, computations of various fundamental groups of spheres, the Freudenthal suspension theorem, the Seifert-van Kampen theorem, construction of Eilenberg-Mac Lane spaces [48], and the Blakers-Massey theorem. Formalization of these results in other systems would have been much more labor intensive. A new line of research develops homotopy theory within HoTT foundations.

As an example, we list Grayson's code that constructs the classifying space as the type of a torsor in HoTT [29]. We include his proof that the classifying space BG is connected. I challenge any other system to pass from the foundations of math to classifying spaces so directly and elegantly!

```
Definition ClassifyingSpace G := pointedType (Torsor G) (trivialTorsor G).
Definition E := PointedTorsor.
Definition B := ClassifyingSpace.
Definition π {G:gr} := underlyingTorsor : E G -> B G.

Lemma isconnBG (G:gr) : isconnected (B G).
Proof. intros. apply (base_connected (trivialTorsor _)).
```

```
  intros X. apply (squash_to_prop (torsor_nonempty X)). { apply propproperty. }
  intros x. apply hinhpr. exact (torsor_eqweq_to_path (triviality_isomorphism X x)).
Defined.
```

4.2. Bourbaki on formalization

Over the past generation, the mantle for Bourbaki-style mathematics has passed to the formal proof community, in the way it deliberates carefully on matters of notation and terminology, finds the appropriate level of generalization of concepts, and situates different branches of mathematics within a coherent framework.

The opening quote claims that formalized mathematics is absolutely unrealizable. Bourbaki objected that formal proofs are too long ("*la moindre démonstration . . . exigerait déjà des centaines de signes*"), that it would be a burden to forego the convenience of abuses of notation, and that they do not leave room for useful metamathematical arguments and abbreviations [16].

Bourbaki is correct in the strict sense that no human artifact is absolutely trustworthy and that the standards of mathematics evolve in a historical process, according to available technology. Nevertheless, the technological barriers hindering formalization have fallen one after another. Today, computer verifications that check millions of inferences are routine. As Gonthier has convincingly shown in the Odd Order theorem project, many abuses of notation can actually be described by precise rules and implemented as algorithms, making the term *abuse of notation* really something of a misnomer, and allowing mathematicians to work formally with customary notation. Finally, the trend over the past decades has been to move more and more features out of the metatheory and into the theory by making use of features of higher-order logic and reflection. In particular, it is now standard to treat abbreviations and definitions as part of the logic itself rather than metatheory.

4.3. Future work

This report has described three major projects in the world of formal proofs: trustworthy systems with HOL, advanced mathematical theorems formalized in Coq, and increased automation.

We are still far from having an automated mathematical journal referee system, but close enough to propose this as a realistic research program. Already some 10% of all papers of the Principles of Programming Languages (POPL) symposium in computer science are completely formalized [61]. Other recent research automates the translation of mathematical prose from English into a computer-parsable form with semantic content [21]. As these technologies develop, we may anticipate the day when the precise formal statements of mathematical theorems may be extracted from the prose. Once sufficiently many statements from the natural language proof can similarly be extracted, proof automation will take over, filling in the remaining details, to produce a formal proof from the natural language text.

For other surveys of formal proofs, see [7], [30].

5. APPENDIX. SOME FORMALLY VERIFIED THEOREMS

This appendix gives examples of some theorems that have been successfully formalized in various proof assistants. The purpose of these examples is to showcase the range of what can be obtained by current technologies.

The Four-Color theorem was formalized in Coq [24].

```
Variable R : real_model.
Theorem four_color : (m : (map R))
    (simple_map m) -> (map_colorable (4) m).
Proof.
   Exact (compactness_extension four_color_finite).
Qed.
```

The elementary proof of the Prime Number Theorem by Erdös and Selberg was formalized in Isabelle [6]. The analytic proof by Hadamard and de la Vallée Poussin was formalized in HOL Light [34]. In the statement that follows, the symbol & denotes the function embedding the natural numbers into the real numbers.

```
// Prime Number Theorem:
((\n. &(CARD {p | prime p ∧ p <= n}) / (&n / log(&n)))
  ---> &1) sequentially
```

Here is the formal statement of the Brouwer fixed point theorem, which was formalized in HOL Light by Harrison.

```
∀ f:real^N->real^N s.
compact s ∧ convex s ∧ ~(s = {}) ∧ f continuous_on s ∧ IMAGE f s SUBSET s
⟹ ∃x. x IN s ∧ f x = x
```

The formalization of the central limit theorem was carried out earlier this year in Isabelle [8].

```
theorem (in prob_space) central_limit_theorem:
  fixes
    X :: "nat ⇒ 'a ⇒ real" and
    μ :: "real measure" and
    σ :: real and
    S :: "nat ⇒ 'a ⇒ real"
  assumes
    X_indep: "indep_vars (λi. borel) X UNIV" and
    X_integrable: "⋀n. integrable M (X n)" and
```

```
  X_mean_0: "⋀n. expectation (X n) = 0" and
  σ_pos: "σ > 0" and
  X_square_integrable: "⋀n. integrable M (λx. (X n x)²)" and
  X_variance: "⋀n. variance (X n) = σ²" and
  X_distrib: "⋀n. distr M borel (X n) = μ"
defines
  "S n ≡ λx. Σi<n. X i x"
shows
  "weak_conv_m (λn. distr M borel (λx. S n x / sqrt (n ∗ σ²)))
      (density lborel standard_normal_density)"
```

The Kepler conjecture asserts that no packing of congruent balls in $\mathbb{R}^3$ can have density greater than the face-centered cubic packing. The Kepler conjecture is a theorem whose proof relies on many computer calculations [31]. The Kepler conjecture has been formalized (7) in a combination of the HOL Light and Isabelle proof assistants [32]. This formalization has been a large collaborative effort.

```
⊢ the_nonlinear_inequalities

⊢ import_tame_classification ∧
   the_nonlinear_inequalities ∧
   ⟹ the_kepler_conjecture

⊢ the_kepler_conjecture ⟺
     (∀V. packing V
            ⟹ (∃c. ∀r. &1 ≤ r
                        ⟹ &(CARD(V ∩ ball(vec 0,r))) ≤
                            π ∗ r³ / sqrt(&18) + c ∗ r²))
```

6. APPENDIX. THE INFERENCE RULES OF HOL

The type system, the terms, sequents, and axioms of HOL have been described in the first section of this report. For reference purposes, we list all the inference rules of HOL, as formulated by Harrison. We borrow from the presentation in [30].

The system has ten inference rules and a mechanism for defining new constants and types. Each inference rule is depicted as a fraction; the inputs to the rule are listed in the numerator, and the output in the denominator. The inputs to the rules may be terms or other theorems. In the following rules, we assume that p and p' are equal, up to a renaming of bound variables, and similarly for b and b'. (Such terms are called α-equivalent.)

7. This project was completed on August 10, 2014.

On first reading, ignore the assumption lists Γ and Δ. They propagate silently through the inference rules, but are really not what the rules are about. When taking the union $\Gamma \cup \Delta$, α-equivalent assumptions should be considered as equal.

Equality is reflexive:

$$\frac{a}{\vdash a = a}$$

Equality is transitive:

$$\frac{\Gamma \vdash a = b; \quad \Delta \vdash b' = c}{\Gamma \cup \Delta \vdash a = c}$$

Equal functions applied to equals are equal:

$$\frac{\Gamma \vdash f = g; \quad \Delta \vdash a = b}{\Gamma \cup \Delta \vdash f\, a = g\, b}$$

The rule of abstraction holds. Equal function bodies give equal functions:

$$\frac{x; \quad \Gamma \vdash a = b}{\Gamma \vdash \lambda x.\ a\ = \lambda x.\ b} \quad \text{(if } x \text{ is not free in } \Gamma)$$

The application of the function $x \mapsto a$ to x gives a:

$$\frac{(\lambda x.\ a)\, x}{\vdash (\lambda x.\ a)\, x = a}$$

Assume p, then conclude p:

$$\frac{p{:}bool}{p \vdash p}$$

An 'equality-based' rule of modus ponens holds:

$$\frac{\Gamma \vdash p; \quad \Delta \vdash p' = q}{\Gamma \cup \Delta \vdash q}$$

If the assumption q gives conclusion p and the assumption p gives q, then they are equivalent:

$$\frac{\Gamma \vdash p; \quad \Delta \vdash q}{(\Gamma \setminus q) \cup (\Delta \setminus p) \vdash p = q}$$

Type variable substitution holds. If arbitrary types are substituted in parallel for type variables in a sequent, a theorem results. Term variable substitution holds. If arbitrary terms are substituted in parallel for term variables in a sequent, a theorem results.

Acknowledgements. I would like to thank the many people who helped me during the preparation of this report, particularly Jeremy Avigad, John Harrison, Chris Kapulkin, Wöden Kusner, and Josef Urban. I wish to give special thanks to Assia Mahboubi for answering many questions related to Coq and the Odd Order theorem. I would also like to thank the many speakers and participants at the special program on *Semantics of proofs and certified math* at IHP.

REFERENCES

[1] Mark Adams. Introducing HOL Zero. In *Mathematical Software–ICMS 2010*, pages 142–143. Springer, 2010.

[2] Jesse Alama, Tom Heskes, Daniel Kühlwein, Evgeni Tsivtsivadze, and Josef Urban. Premise selection for mathematics by corpus analysis and kernel methods. *Journal of Automated Reasoning*, 52(2):191–213, 2014.

[3] Michael Aschbacher. *Finite group theory*, volume 10. Cambridge University Press, 2000.

[4] Michael Aschbacher. The status of the classification of the finite simple groups. *Notices of the AMS*, 51(7):736–740, 2004.

[5] Michael Aschbacher and Stephen D Smith. *The classification of quasithin groups*. American Mathematical Soc., 2004.

[6] Jeremy Avigad, Kevin Donnelly, David Gray, and Paul Raff. A formally verified proof of the prime number theorem. *ACM Transactions on Computational Logic (TOCL)*, 9(1):2, 2007.

[7] Jeremy Avigad and John Harrison. Formally verified mathematics. *Communications of the ACM*, 57(4):66–75, 2014.

[8] Jeremy Avigad, Johannes Hölzl, and Luke Serafin. A formally verified proof of the central limit theorem. *arXiv preprint arXiv:1405.7012*, 2014.

[9] Steve Awodey and Michael A Warren. Homotopy theoretic models of identity types. *Math. Proc. Cambridge Philos. Soc.*, 146:45–55, 2000.

[10] Leo Bachmair and Harald Ganzinger. Resolution theorem proving. *Handbook of automated reasoning*, 1:19–99, 2001.

[11] Bruno Barras. Sets in Coq, Coq in sets. *Journal of Formalized Reasoning*, 3(1):29–48, 2010.

[12] Clark W Barrett, Roberto Sebastiani, Sanjit A Seshia, and Cesare Tinelli. Satisfiability modulo theories. *Handbook of satisfiability*, 185:825–885, 2009.

[13] Helmut Bender, George Glauberman, and Walter Carlip. *Local analysis for the odd order theorem*, volume 188. Cambridge University Press, 1994.

[14] Y. Bertot and P. Castéran. *Interactive Theorem Proving and Program Development Coq'Art: The Calculus of Inductive Constructions*. Springer, 2004.

[15] Sascha Böhme and Tobias Nipkow. Sledgehammer: judgement day. In *Automated Reasoning*, pages 107–121. Springer, 2010.

[16] Nicolas Bourbaki. *Théorie des ensembles*. Springer, 2006. (third edition, Hermann, 1966).

[17] Cyril Cohen and Assia Mahboubi. A formal quantifier elimination for algebraically closed fields. In *Intelligent Computer Mathematics*, pages 189–203. Springer, 2010.

[18] Thierry Coquand and Gérard P. Huet. The calculus of constructions. *Inf. Comput.*, 76(2/3):95–120, 1988.

[19] Walter Feit, Marshall Hall Jr., and John G. Thompson. Finite groups in which the centralizer of any non-identity element is nilpotent. *Math Z.*, 74:1–17, 1960.

[20] Walter Feit and John G Thompson. *Solvability of groups of odd order*. Pacific Journal of Mathematics, 1963.

[21] Mohan Ganesalingam. *The language of mathematics*. Springer, 2013.

[22] Kevin Elphinstone Toby Murray Thomas Sewell Rafal Kolanski Gerwin Klein, June Andronick and Gernot Heiser. Comprehensive formal verification of an OS microkernel. *ACM Transactions on Computer Systems*, 32(1):2:1–2:70, February, 2014.

[23] George Glauberman. A new look at the Feit-Thompson odd order theorem. *Mat. Contemp*, 16:73–92, 1999.

[24] Georges Gonthier. Formal proof–the four-color theorem. *Notices of the AMS*, 55(11):1382–1393, 2008.

[25] Georges Gonthier. Point-free, set-free concrete linear algebra. In *Interactive Theorem Proving*, pages 103–118. Springer, 2011.

[26] Georges Gonthier, Andrea Asperti, Jeremy Avigad, Yves Bertot, Cyril Cohen, François Garillot, Stéphane Le Roux, Assia Mahboubi, Russell O'Connor, Sidi Ould Biha, et al. A machine-checked proof of the odd order theorem. In *Interactive Theorem Proving*, pages 163–179. Springer, 2013.

[27] Mike Gordon. From LCF to HOL: a short history. *Proof, language, and interaction: essays in honour of Robin Milner*, pages 169–185, 2000.

[28] Daniel Gorenstein. *Finite groups*, volume 301. American Mathematical Soc., 2007.

[29] Daniel Grayson. K-theory: Group action, 2014. `https://github.com/UniMath/UniMath/blob/master/UniMath/Ktheory/GroupAction.v`.

[30] Thomas C. Hales. Formal proof. *Notices of the AMS*, 55(11):1370–1380, December 2008.

[31] Thomas C. Hales and Samuel P. Ferguson. The Kepler conjecture. *Discrete and Computational Geometry*, 36(1):1–269, 2006.

[32] Thomas C. Hales, Alexey Solovyev, and Hoang Le Truong et al. The Flyspeck Project, 2014. `http://code.google.com/p/flyspeck`.

[33] John Harrison. Towards self-verification of HOL Light. In *proceedings of IJCAR 2006*, volume 4130 of *Lect. Notes in Comp. Sci.*, pages 177–191. springer, 2006.

[34] John Harrison. Formalizing an analytic proof of the prime number theorem. *Journal of Automated Reasoning*, 43(3):243–261, 2009.

[35] John Harrison. *Handbook of Practical Logic and Automated Reasoning*. Cambridge University Press, 2009.

[36] John Harrison. HOL Light: An overview. In *Theorem Proving in Higher Order Logics*, pages 60–66. Springer, 2009.

[37] John Harrison. The HOL Light theorem prover, 2010. `http://www.cl.cam.ac.uk/~jrh13/hol-light/index.html`.

[38] I. Martin Isaacs. *Character theory of finite groups*. Courier Dover Publications, 2013.

[39] Cezary Kaliszyk and Josef Urban. MizAR 40 for Mizar 40. *CoRR*, abs/1310.2805, 2013.

[40] Cezary Kaliszyk and Josef Urban. HOL(y)Hammer: Online ATP service for HOL Light. *Mathematics in Computer Science*, 2014. In press, `http://arxiv.org/abs/1309.4962`.

[41] Cezary Kaliszyk and Josef Urban. Learning-assisted automated reasoning with Flyspeck. *Journal of Automated Reasoning*, 2014. `http://dx.doi.org/10.1007/s10817-014-9303-3`, arXiv:1211.7012.

[42] Cezary Kaliszyk and Josef Urban. Learning-assisted theorem proving with millions of lemmas. *Journal of Symbolic Computation*, 2014. In press, `http://arxiv.org/abs/1402.3578`, arXiv:1402.3578.

[43] Chris Kapulkin, Peter LeFanu Lumsdaine, and Vladimir Voevodsky. The simplicial model of univalent foundations. *arXiv preprint arXiv:1211.2851*, 2012.

[44] Laura Kovács and Andrei Voronkov. First-order theorem proving and Vampire. In *Computer Aided Verification*, pages 1–35. Springer, 2013.

[45] Ramana Kumar, Rob Arthan, Magnus O Myreen, and Scott Owens. HOL with definitions: Semantics, soundness, and a verified implementation. *Interactive Theorem Proving (ITP), LNCS. Springer*, 2014.

[46] Ramana Kumar, Magnus O Myreen, Michael Norrish, and Scott Owens. CakeML: A verified implementation of ML. In *Proceedings of the 41st annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 179–192. ACM, 2014.

[47] Xavier Leroy. Formal certification of a compiler back-end, or: programming a compiler with a proof assistant. *33rd ACM symposium on Principles of Programming Languages*, pages 42–54, 2006. http://compcert.inria.fr/.

[48] Daniel R. Licata and Eric Finster. Eilenberg-Mac Lane spaces in homotopy type theory. *CSL-LICS 2014*, pages 1–9, 2014.

[49] Joao Marques-Silva, Inês Lynce, and Sharad Malik. Conflict-driven clause learning sat solvers. *Handbook of satisfiability*, 185:131–153, 2009.

[50] Mizar home page, 2013. http://mizar.org/.

[51] Magnus O Myreen and Jared Davis. The reflective Milawa theorem prover is sound. 2012.

[52] Magnus O Myreen and Scott Owens. Proof-producing translation of higher-order logic into pure and stateful ML. *Journal of Functional Programming*, pages 1–32, 2014.

[53] Magnus O Myreen, Scott Owens, and Ramana Kumar. Steps towards verified implementations of HOL Light. In *Interactive Theorem Proving*, pages 490–495. Springer, 2013.

[54] Lawrence Paulson. Three years of experience with sledgehammer, a practical link between automatic and interactive theorem provers. Paar-2010, Practical Aspects of Automated Reasoning, 2010.

[55] Álvaro Pelayo and Michael A Warren. Homotopy type theory and Voevodsky's univalent foundations. *Bulletin of the A.M.S.*, 2014. to appear, arXiv preprint arXiv:1210.5658.

[56] Thomas Peterfalvi. *Character theory for the odd order theorem*, volume 272. Cambridge University Press, 2000.

[57] Martin Raussen and Christian Skau. Interview with Jean-Pierre Serre. *Notices of AMS*, 51(2):210–214, 2004.

[58] Alexandre Riazanov and Andrei Voronkov. The design and implementation of vampire. *AI communications*, 15(2):91–110, 2002.

[59] Susmit Sarkar, Peter Sewell, Francesco Zappa Nardelli, Scott Owens, Tom Ridge, Thomas Braibant, Magnus O Myreen, and Jade Alglave. The semantics of x86-CC multiprocessor machine code. In *ACM SIGPLAN Notices*, volume 44, pages 379–391. ACM, 2009.

[60] Stephan Schulz. E - A Brainiac Theorem Prover. *AI Commun.*, 15(2-3):111–126, 2002.

[61] Peter Sewell. POPL 2014 program chair's report. `http://www.cl.cam.ac.uk/~pes20/`, 2014.

[62] Ronald Soloman. A brief history of the classification of the finite simple groups. *Bulletin AMS*, 38(3):315–352, 2001.

[63] A. Solovyev. Formal methods and computations, 2012. thesis, University of Pittsburgh, `http://d-scholarship.pitt.edu/16721/`.

[64] The Coq development team. The Coq proof assistant: reference manual. `http://coq.inria.fr/refman/`, Version v8.4pl4, 2014.

[65] John G. Thompson. Nonsolvable finite groups all of whose local subgroups are solvable. *Bulletin of the American Mathematical Society*, 74(3):383–437, 1968.

[66] The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics.* `http://homotopytypetheory.org/book`, Institute for Advanced Study, 2013.

[67] Josef Urban, Krystof Hoder, and Andrei Voronkov. Evaluation of automated theorem proving on the Mizar mathematical library. In *Mathematical Software–ICMS 2010*, pages 155–166. Springer, 2010.

[68] Christoph Weidenbach, Dilyana Dimova, Arnaud Fietzke, Rohit Kumar, Martin Suda, and Patrick Wischnewski. SPASS Version 3.5. In *CADE*, pages 140–145, 2009.

[69] Wikipedia list of longest proofs, accessed 11/2013. `http://en.wikipedia.org/wiki/Longest_proof`.

Thomas C. HALES

University of Pittsburgh
Department of Mathematics
Pittsburgh, PA 15260-2341
U.S.A.
E-mail : `hales@pitt.edu`