Engineering Mathematics: The Odd Order Theorem Proof

Georges Gonthier

Microsoft Research Cambridge
7 JJ Thomson avenue
Camdridge, UK CB3 0FB
gonthier@microsoft.com

Abstract

Even with the assistance of computer tools, the formalized description and verification of research-level mathematics remains a daunting task, not least because of the talent with which mathematicians combine diverse theories to achieve their ends. By combining tools and techniques from type theory, language design, and software engineering we have managed to capture enough of these practices to formalize the proof of the Odd Order theorem, a landmark result in Group Theory.

Categories and Subject Descriptors F.4.1 [MATHEMATICAL LOGIC AND FORMAL LANGUAGES]: Mathematical Logic – Mechanical theorem proving. J.2 [PHYSICAL SCIENCES AND ENGINEERING] – Mathematics and statistics.

Keywords theorem proving; proof engineering; group theory; mathematical components; Coq; ssreflect.

1. The Challenge of Computer Mathematics

This past year we were reminded that the "killer app" that led Alan Turing to the invention of Computer Science was the completely formal verification of mathematical theories and proofs. Yet while computation with mechanical contraptions has found useful applications in nearly every human activity, this original goal has remained elusive. Even with the latest computer proof systems, which have become very good at the verification of code and POPL papers, dealing with the proofs produced by working mathematicians can be most painful.

My experience with the formalization of the four-color theorem in 2001-5 indicated that the problem might not be a lack of automation, as is often believed, but a lack of expressiveness of the tools and methods used to describe mathematics. Even though the four-color proof uses only the most elementary combinatorics, its formalization had required new ways of organizing and writing theories and proofs – but hardly any general-purpose proof-finding automation.

This suggested that language and software engineering techniques could perhaps be applied successfully to mainstream mathematics. In 2006 I founded the Mathematical Components project at the Microsoft Research – Inria Joint Center to test this thesis, by developing a comprehensive formal algebra using the Coq system. The aim was to cover most of a math undergrad syllabus, and the work was to be driven and validated by the formalization of a famously difficult and important result in Group Theory, Feit and Thompson's Odd Order theorem. In September 2012, we finally prevailed.

POPL'13, January 23–25, 2013, Rome, Italy.
ACM 978-1-4503-1832-7/13/01.

2. The Odd Order Theorem

Feit and Thompson's seminal 1963 paper began with the deceptively short statement

THEOREM. All finite groups of odd order are solvable.

The 255 pages that followed exposed the most difficult and complex group theory proof of its time, drawing on most of elementary results in algebra (including matrices, polynomials, normed vector spaces, modules, algebraic number theory, and Galois theory), and advanced results in group theory (such as the Schur-Zassenhaus and Hall-Higman theorem, or the theory of exceptional characters). The ground-breaking paper introduced several novel techniques that would later play a key role in the full classification of finite simple groups (e.g., Thompson transitivity, tamely imbedded subsets, and signalizer functors). Its result has applications throughout group theory, and can be seen as the starting point of the classification itself. Yet it was considered a very difficult paper to read, before a revision in 1994 and 2000 simplified and filled several gaps in the exposition. All of this made the Odd Order proof the perfect test for a mathematical formalization project.

Even the expanded statement of the theorem is short, because it only involves two definitions:

DEFINITION. H is a *normal subgroup* of a group G iff H is the kernel of a homomorphism from G to a *factor group* G / H: one can compute "mod H" in G. This is written $H \triangleleft G$.

DEFINITION. A group G is *solvable* if there is a *normal series* $1 = H_0 \triangleleft H_1 \triangleleft \cdots \triangleleft H_n = G$.such that each factor H_{i+1} / H_i is abelian.

Indeed, a full formalization of the statement from basic logic (including the necessary arithmetic and combinatorics) takes only two pages. Note that any finite group has a maximal normal series whose factors are all simple groups (this is why the classification is so important); for solvable groups these factors will all be prime cyclic groups. Also, solvable groups are tied to the resolution of polynomial equations, the motivating application of group theory.

The Feit-Thompson proof is just as complex as the Odd Order statement is simple; it proceeds by studying a minimal counter-example G, gradually refining its structure until a contradiction is reached. This study consists of three unequal parts:

Local Analysis One studies the proper subgroups of G, which are all solvable by assumption. Using finite representation theory (matrices with coefficients in a finite field) one shows that their intersection has rank at most 2; in turn this implies that most

of these groups closely resemble Frobenius groups, with two possible exceptions, similar to a finite field with its Galois group.

Character Theory One uses character theory (traces of matrices with complex coefficients) to extract information about G from the description of its subgroups. This is possible because characters have a Euclidean geometry, and there are isometries mapping certain subgroup characters to characters of G. The global norm inequalities imply that subgroups are exactly Frobenius, and that the two Galois exceptions occur exactly as well.

Generators and relations Using fiendish explicit calculations exploiting the relations between the finite fields embedded in G, one constructs a polynomial of degree q with $p > q$ roots in a field of order p^q; at last, contradiction!

The revised proof of the first part, by Bender and Glauberman, spans the 140 pages of volume 188 of the LMS series; the second part, by Peterflavi, spans 100 pages of volume 272, while the last is a mere 8-page appendix in volume 188. The prerequisites represent about twice that amount, from a variety of sources. The Coq formalization of the 1300 lemmas of the proof spans 47,000 LOC (out of 170,000 for the full archive).

3. Mathematical Components

The basic premise of project was that formalized theories would be more useful if they could be composed more freely, and that this could be achieved by replacing the rigid 19th century first order logic with CiC, the more modern dynamic higher-order logic supported by the Coq system. First order logic imposes an exact fit, at the symbol level, between the statement of the assumptions of a proof step and the facts that fulfil them. In CiC, parts of a statement can be executable, and matching is performed up to (functional) evaluation. Thus in principle it is possible to formulate theories so that they behave more like software components that automatically generate "glue code" to facilitate interoperability, rather than simple procedures that only work for a fixed data representation. Our challenge was to understand how this could be made useful in practice.

We realized early on that this flexibility was even more crucial for stating theorems than for proving them. Mathematical notation is cleverly honed to provide exactly the information needed to determine a mathematical object in a given context. Much of it makes little sense in the rigid, context-independent setting of first-order logic. For instance the factor group G / H we saw earlier depends on the multiplication operator used in G and H. This must be determined by the context, if we are to support the common practice of computing and equating G and H as sets, as in the statement of the isomorphism theorem $GH / H \simeq G / (G \cap H)$; the other isomorphism theorem $(G / K) / (H / K) \simeq G / H$, where the '/'s refer to three different multiplications, shows that "context" must be understood locally.

The solution to this overloading problem is the familiar "object" paradigm, specifically its functional programming equivalent, type classes. In fact in Coq type classes are significantly more powerful than in traditional functional programming, because they have an internal representation in the CiC logic as type-containing records. Instance inference is an integral part of type inference; it amounts to inverting projections of class records during higher-order unification, using a set of canonical instances declared by the user. It can supply both operations an properties, and be based both on types and data values. A single expression can exploit several type classes; for example in G / H a "finite group type" class is used to determine which multiplication is involved and to ensure that the definition is constructive, and a "group" class is used to ascertain that resulting set is actually a group (as long as G is).

In addition to supporting the traditional algebraic hierarchy of rings, vector spaces, fields, algebras, etc. our type classes can also recognize groups, sesquilinear functions, AC operators, subfields and even direct sums of subspaces. The extended type inference substantially helps the formalization process: it is not uncommon that a single line of LaTeX-like notation hides two pages of inferred annotations, and when we introduced the group class nearly half of our proof scripts vanished into type inference.

4. Big Theories, Big Theorems, Big Proofs

Software engineering techniques, albeit more mundane ones, are also instrumental in coping with the sheer size of the prerequisite theories. Ensuring that the 4200 definitions and 15,000 lemmas and theorems remain accessible requires a triple combination of namespace management (across modules), rational naming conventions (within modules), and effective search tools (in the IDE).

The distribution of formal proof lines is a close fit to a power (Pareto) law. This is a heavy-tailed distribution, which means that while most lemmas will have a very short proof (two lines or less), most proof lines will occur in the proof of large theorems, and that such proofs will get very large indeed. Thus the formalization of the Odd Order theorem accounts for 10% of the theorems, but 30% of the script lines, and includes a 1,200 line proof (for Theorem (9.11) in the character theory revised proof).

We also observed the need for large theorem statements: it is preferable to have one large theorem that collects all the facts about a given situation (e.g., the structure of an extremal p-group), rather than a plethora of factoids whose applicability must be justified separately. Our proof scripting language, implemented by the ssreflect Coq plugin, efficiently supports extracting a relevant fact subset from such a theorem.

In order to handle the big proofs, our scripting language helps enforcing explicit control and data flow. The latter also turned out to be an invaluable tool for mining the logical structure of a proof, uncovering in several cases major simplifications.

Finally, we identified several formal proof structures that cannot be expressed easily with the traditional assertion, case analysis and induction constructs. A general "without loss of generality" construct, was invaluable in capturing intricate symmetry arguments. A small library for three-way inequalities proved indispensable for formulating concisely circular inequalities.

Acknowledgments

The work described here was done in collaboration with the members and visitors of the Mathematical Components project: A. Asperti, J. Avigad, Y. Bertot, C. Cohen, F. Garillot, S. Le Roux, S. McLaughlin, A. Mahboubi, S. Ould Biha, R. O'Connor, L .Rideau, A. Solovyev, E. Tassi, and L. Théry.