Mechanized foundations of finite group theory

François Garillot
francois.garillot@inria.fr

INRIA - Microsoft Research Joint Lab
91890 Orsay Cedex
France

Abstract

We report on a long-term formalization effort on finite group algebra, aimed at a mechanized proof of the Feit-Thompson theorem. With a better understanding of this particularly long and intricate theorem as a goal, our work devotes considerable attention to the scalability of our proof engineering choices. We use the Coq proof assistant, extended by the SSReflect library to benefit from small scale reflection. Our approach already furnishes a leading development of finite group theory, that showcases the advantages of syntactic notation facilities, as well as type system constructs such as canonical structures.

1 Introduction

This poster describes an ongoing work formalizing finite group algebra, using the SSReflect [9] extension to the Coq theorem prover.

Proof assistants provide practical advantages for the development of formal mathematics. Through exhaustive checking, they give strong guarantees of manipulated properties, and often allow for a better understanding of the ramifications of a proof. To achieve this, Coq provides a dependently-typed language in which properties can be represented as programs, and in which type checking is the main mechanism to ensure those properties behave soundly with respect to each other.

However, the use of theorem provers is less frequent when dealing with mathematics than with computer science theorems. The time efficiency of computer reasoning is often blamed for it, particularly the poor ergonomy of the prover interface, and the necessity of building proofs from the ground up. Mathematics, being a highly erudite field, often require the user to treat several layers of concepts before enunciating any non-trivial statements. To manipulate them, he is then required to omit some of those layers on occasion, in order to adopt the level of abstraction that makes his discourse both pertinent and manageable -a process we intend to ease in our development.

The germ of this project comes from the work of Georges Gonthier and Benjamin Werner on the proof of the Four Color theorem [5]. This considerable formal development was joined with the drafting of a syntactic extension to the Coq theorem prover, aimed at making the proof-writing process shorter, that our work expands and refines.

On the theoretic level, we also develop formal building blocks for finite group algebra, with the particular orientation provided by the proof of the odd order theorem. This proof, joining distinct theories and techniques, namely local analysis [2] and character theory [8], will require us to bridge parts of our development in a flexible manner, for which conforming to one precise syntactic convention on theoretical statements proves unmanageable. The main challenge is to find the appropriate way of achieving compositionality in a rich, dependently-typed language, over a long development.

2 Implementing Finite Group Theory with Coq

Our approach combines interactive theorem proving in Coq with powerful rewriting techniques provided by the SSReflect extension through the use of small scale reflection. As described in [4], we can switch between the logical and boolean forms of properties quickly, allowing us to minimize the tedious steps of context bookkeeping by writing compact rewrite commands.

This has allowed us to develop the basic objects of group theory using canonical structures. These constructs -in an nutshell, dependent record types with an inference mechanism- allow us to define a dictionary of classes, and to register canonical instances of those inside Coq's type checking mechanism. The interesting increment here is using this feature (closely related to Haskell's type classes), with dependent types: when for example we declare a canonical instance of a record containing one object$x$ and a property $P(x)$ built this object, Coq will, at each occurrence of x, have immediate access to a proof of the associated property. Reasoning about these structures is also very convenient, providing a powerful extension to the notion of refinement types.

This styles works very well in practice, allowing us, as described in [4], to build a development proving the Sylow theorems, a point that to our knowledge, but one existing formalization of algebra reaches [6]. We have recently refactored the architecture described in this paper using an encoding of multiple inheritance, and are experiencing significant improvements in the clarity of our formalization, and in making those canonical structures work smoothly with the Coq engine.

An interesting side effect of dealing with mathematical objects is the lessons learned from the intricacy of their structure. For instance, the iteration of an algebraic operator over any finite sequence of elements is a multi-faceted object, whose properties depend on those of said operator, such as commutativity - even if the common 'fold' operation is usually complex enough to describe it from a programmer's point of view. In an upcoming paper, we described how we managed to formalize this operation generically in a compact library, while still providing the user with an ease of use comparable to that of the `\big` operator in LaTeX.

3 Conclusion

Computer-assisted reasoning is a technique that enjoys a fast-growing adoption from researchers interested in programming languages. This is in part because the field has, through active campaigning ([1]), given rise to a set of techniques and best practices that makes the proof engineering process straightforward, if only for specific type of proofs. Mathematics, while being a field on which manageable formalized proving could have a great impact -since the size of proofs is sometimes numbered in the hundreds of pages-, lack the tools to do the same.

We are confident that formalized theorem proving for mathematics can be made easier through the use of carefully-crafted languages and tools such as SSReflect. Meanwhile, we believe that devising flexible mathematical components can organize mathematical theories in a compositional manner, allowing mathematicians to find formal building blocks of their specific subject at the time they want to craft a proof. Finally, we hope that the intricacy of mathematical objects will naturally yield powerful tools that whose usefulness will go beyond the formalization of mathematics.

References

[1] B. Aydemir et al, The POPLMark Challenge, *Theorem Proving in Higher Order Logics, 2005*:50-65.

[2] Helmut Bender and Georges Glauberman, *Local Analysis for the Odd Order Theorem*. Number 188 in London Mathematical Society Lecture Note Series, Cambridge University Press, 1994.

[3] Y. Bertot et al., Canonical big operators. to appear : *Theorem Proving in Higher Order Logics, 2008*

[4] G. Gonthier et al. A modular formalization of Finite Group Theory. *Theorem Proving in Higher Order Logics, 2007*:86-101.

[5] G. Gonthier. A computer-checked proof of the four-color theorem. Available at `http://research.microsoft.com/~gonthier/4colproof.pdf`

[6] F. Kammüler and L. C; Paulson. A formal proof of Sylow's theorem : An experiment in abstract algebra with Isabelle HOL. *J. Automated Reasoning* 23(3-4):235-264(1999).

[7] I. Pasca. A Formal Verification for Kantorovich's Theorem. In *Journées Francophones des Langages Applicatifs, january 2008*.

[8] Thoms Peterfalvi. *Character Theory for the odd Order Theorem*. Number 272 in London Mathematical Society Lecture Note Series. Cambridge University Press, 2000.

[9] The SSReflect extension to the Coq theorem prover. Available at `http://www.msr-inria.inria.fr/Projects/math-components/`