The Formal Proof

Susan Gillmor and Samantha Rabinowicz

Project for MA341: Appreciation of Number Theory

Boston University Summer Term 1 2009

Instructor: Kalin Kostadinov

A proof verifies a conjecture by using axioms and previously proved theorems to a logical argument. Proving conjectures has been done for thousands of years and can take on a wide range of forms. Some techniques have been popularized for their versatility, for example, proof by induction, proof by contradiction, proof without words, and the two-column proof. Since the 1950's a new method has become available to us, proof by computer. Validating a proof's logic on a computer is called formalization. The program that performs this function is called a proof assistant. The term proof assistant encompasses both proof checkers and automated theorem provers. A proof checker is the earliest form of a proof assistant which requires the user to provide both the proposition and the proof and it simply verifies the logic. An automated theorem prover is the main focus of this paper. Automated theorem provers take user provided propositions and tactics into their 'proof engines' to interactively generate a proof. These systems partially construct formal proofs by filling in intermediate steps of a proof. The computer is less susceptible to logical error and thereby may uncover innovative outcomes which the human may have initially dismissed as implausible. Having no intuition or expected results, the computer is extremely thorough trying every possibility and provides logical proofs of theorems.

An example of this process is demonstrated through the most comprehensively verified proof in digital history, Gonthier's formalization of the Four-Color Theorem. The Four-Color Theorem is both controversial and exciting because it could not be proved by hand. This theorem asserts that any map divided into any number or arrangement of contiguous regions can be colored with only four colors with no two regions of same color adjacent. Using a computer as their method, computer scientists Appel and Haken were the first to ever prove this theorem in 1976. In fact, this was the first significant theorem proved using advanced computer assistance. The Four-Color Theorem is unique in that it was one of the first computer-based proofs; not only did the computer assist in confirming the logic, the computer was necessary.

There exist two major proof styles which have been utilized in a range of proof assistants, the procedural proof style and the declarative proof style. With the procedural proof style, the mathematician initiates a dialogue with the computer and engages in an "interactive game" in which the computer presents the user with proof obligations, or goals, and then the user executes tactics, breaking the goals into smaller, more manageable sub-goals. The alternative approach, declarative proof style, uses very small steps to develop a formal proof all at once, here, the user tells the computer what is to be proved and where to go using sub-goals.

The foundation of proof assistant programming language is type theory. These typed languages include lambda calculus and calculus of constructions. Just as there exist symbolic mathematical terms to represent the English language in a concise form, there exist well-chosen type-script to express formulas and prepositions on the computer. For example, within calculus of construction exist terms such as propositions, predicates, and judgments, which serve as inference rules and definitions of logical operators and data types. This typed language enables mathematicians to clearly differentiate between points and lines, numbers and set of numbers, etc.

The axioms and rules of inference, used by any proof assistant are stored in a kernel of the system. This small kernel verifies every line of code generated by the computer. With such a significant role in verifying sound, logical proofs, the kernel is checked in many different ways to eliminate any potential for bugs.

For this project, we explored the Higher Order Logic (HOL) Light proof assistant and the Coq proof assistant, which both create procedural style proofs. HOL Light uses a lambda-calculus type theory which utilizes lambdas (λ) to symbolize abstracted functions. This type theory uses equality as the basis for every rule.

HOL Light consists of three mathematical axioms as its foundation: the axiom of extensionality, the axiom of infinity, and the axiom of choice. The fact that every function is determined by its input-output nature is the extensionality axiom. The infinity axiom reports the existence of a function that is one-to-one but not onto. Lastly, the axiom of choice finds an element to satisfy a predicate, P(x).

Downloading the HOL Light system provides us with examples of proofs including the formalized version of the Law of Quadratic Reciprocity, first introduced by Euler and then later proved by Gauss. As we studied in class, the Law of Quadratic Reciprocity tells us whether a quadratic equation with a prime modulus has a solution. By studying this formalized proof and analyzing portions of the program we gain an understanding of the general language and format which is prevalent throughout all formal proofs generated in HOL Light. For example, let's focus on the third lemma of the computer generated proof:

```
 let CONG_MINUS1_SQUARE = prove
 (`2 <= p ==> ((p - 1) * (p - 1) == 1) (mod p)`,
  SIMP_TAC[LE_EXISTS; LEFT_IMP_EXISTS_THM] THEN REPEAT STRIP_TAC
THEN
  REWRITE_TAC[cong; nat_mod; ARITH_RULE `(2 + x) - 1 = x + 1`] THEN
  MAP_EVERY EXISTS_TAC [`0`; `d:num`] THEN ARITH_TAC);;
```

The structure of a proof in HOL Light consists of lemmas. Each lemma is designed with three parts: a label, a statement, and a proof. The label identifies the lemma so the result can be referenced later in the proof. The first line names the result by assigning a label, in this case CONG_MINUS!_SQUARE. The second line is the statement, and the last three lines are the encoded proof. You can see that this lemma references four earlier lemmas using their labels (LE_EXISITS, LEFT_IMP_EXISTS_THM, cong, nat_mod).

While HOL Light is one of the most highly respected proof assistants, it sometimes cannot express abstract concepts and certain outputs may be indecipherable to humans. Whereas HOL Light uses lambda-calculus type theory for function definition, function application, and recursion, the proof assistant Coq uses calculus of inductive constructions. This type theory is based on inductively defined relations. In constructive logic, if there is a proof, there is a function 'hidden' in the proof. The type-script is considered the formula for the function, and the generated proof is the program. As mentioned above, the Four-Color Theorem was proved in 1979 by computer; however, it wasn't until 2005 that it was proved by using a single computer program, Coq. Before Gonthier and Werner proved the Four-Color Theorem using Coq, many different computer programs were required to prove each segment of the proof separately.

We used the online software ProofWeb to demonstrate an example of the proof assistant Coq. We must note that ProofWeb is not as efficient as the Coq proof assistant and deviates from some normal syntax and commands which define the language of Coq. Coq builds natural deduction proofs either in a 'tree' or 'box' style. By executing *tactics*, or commands, the proof will start to "grow." The tactics in ProofWeb are usually denoted by a three letter abbreviation followed by an underscore and the letter 'i' or 'e' signifying introduction or elimination. The tactics are generally arguments labeled as formulas or terms. The formalized proof for the statement A /\ B → A appears as follows in ProofWeb:

```
Require Import ProofWeb.

Variables A B : Prop.

Theorem prop_001 : (A /\ B) -> A.
imp_i H.
con_e1 (B).
exact H.

Qed.
```

Let's break down this program line-by-line as we did for a section of a HOL Light proof. "Require Import ProofWeb" simply imports the definitions needed to run Coq. Next, "Variables A B : Prop." declares propositional variables A and B. As with most other computer languages, a variable must be defined before it can be used. "Theorem prop_001 : (A /\ B) -> A" tells the computer what we are going to prove in this case A /\ B → A and labels it "prop_011." The three lines: "imp_i H," "con_e1 (B)," and "exact H" are the tactics, or commands, which make up the body of the proof. Once this program has run in ProofWeb the prop_001 is successfully defined.

As mathematicians continue to define theorems via computer, the formalized proofs enter into an electronic library. Once the library is complete with all known theorems, mathematicians speculate that the computer will be able to take the next step and conceive its own theorems and proofs, like an “automated mathematician." We are now faced with the question: Does designing a proof require thought? If so, does this mean computers will be able think? We are temped to argue no, because computers are not conceptualizing what they are doing, they are simply trying every possibility. Computers rely on the small kernel of axioms and inference rules to verify every step of the proof. In the body of this paper we have demonstrated the process, language, and format of two popular proof assistants, HOL Light and Coq, constantly acknowledging the user's responsibility in feeding the computer information. If the human were to mistype, or accidentally skip an essential key, the computer would respond with error or undefined. The computer relies on the input generated by the human and requires that the human logically and correctly feed information to the program. To have a thought requires consideration and evaluation - neither of which the computer does.

Rather than conceptualizing, computers are performing tasks as described to them by their users. As with any other proof, it is important to not simply rely on the logic of the creator because, by their nature, computer programs contain bugs. As pure mathematicians, many number theorists agree that before using a theorem one must prove it themselves. The justification for this rigorous process roots in the holes and flaws in so many of our published mathematics papers. By leaving the validation of proofs up to computers we are losing accuracy due to intrinsic bugs in computers and the mistakes of computer programmers. Additionally, we may be dropping a generation of mathematical understanding.

Many critics of proof assistants argue "that proofs of computer system correctness are often likely to be so long and tedious that humans cannot reasonably check them and discuss them" (Harrison 1402). If theorems are not constantly being proved around the world, the understanding, creativity, and insight it takes to draw a rigorous proof will be lost. Proving a theorem inspires new ideas and new intuitions that a computer is not capable of. Does this mean our future mathematicians will no longer see the beauty of proofs first-hand? Will we lose our inspiration to constantly be proving and creating mathematics?

Lastly, should we be concerned proof diversity? Theorems have been proved again and again by different mathematicians and using a range of methods. For instance, the Pythagorean Theorem has been proved in over 80 different ways using not only algebra and geometry but proofs without words and induction. If computers serve as our primary source of generating proofs, will mathematicians settle for only one proof-style per theorem? Taking all of these questions into consideration, we must be cautious that we preserve the integrity of mathematics. Technology has provided us with unlimited boundaries for the future of mathematics; however, we must keep in mind Pete Parker's words in Spiderman (2002): "with great power comes great responsibility."

Works Cited

"Coq." Wikipedia. 13 May 2009. 16 June 2009 <http://en.wikipedia.org/wiki/Coq>.

Geuvers, H. "Proof Assistants: History, Ideas and Future." Sadhana 34.1 (2009): 3-25.

Hales, Thomas C. "Formal Proof." Notices of the American Mathematical Society 55.11 (2008): 1370-1380.

Harrison, John. "Formal Proof--Theory and Practice." Notices of the American Mathematical Society 55.11 (2008): 1395-1406.

HOL Light. Vers. 2.20. 8 June 2009 <http://www.cl.cam.ac.uk/~jrh13/hol-light/>.

"HOL Light ." Wikipedia. 31 Mar. 2009. 10 June 2009 <http://en.wikipedia.org/wiki/hol_light>.

Kaliszyk, Cezary. "Provers: Coq." ProofWeb. 2007. Surf Foundation, Radboud U Nijmegen, Free U Amsterdam. 16 June 2009 <http://prover.cs.ru.nl/login.php>.

Weidijk, Freek. "Formal Proof -- Getting Started." Notices of the American Mathematical Society 55.11 (2008): 1408-1417.