

Teaching a HOL Course: Experience Report

Konrad Slind, Steven Barrus, Seungkeol Choe, Chris Condrat, Jianjun Duan,
Sivaram Gopalakrishnan, Aaron Knoll, Hiro Kuwahara, Guodong Li, Scott
Little, Lei Liu, Steffanie Moore, Robert Palmer, Claurissa Tuttle, Sean Walton,
Yu Yang, and Junxing Zhang

School of Computing, University of Utah
slind@cs.utah.edu

Abstract. Experience from teaching a course in Higher Order Logic theorem proving is recounted, from both the student and teacher perspective.

1 Introduction

Higher Order Logic is wonderful fun to work with, but we all had to learn it somehow. In the past, highly talented and persistent researchers and PhD students struggled to learn the logic (the easy part) and the capabilities of the implementations (the hard part). In many cases, successful practitioners absorbed the necessary collection of incantations, minutiae, tricks, and other dark arts by sitting at the feet (or at a desk in the same office) of a Master, who had quite often been trained in a similar fashion. Such apprenticeships are often successful, since they allow very efficient information exchange. However, they are rather less efficient when the number of students increases.

The more structured approach of designing and delivering a course in a classroom setting has been taken many times before, of course. The lead author himself learned HOL from Graham Birtwistle, in a course largely based on painstaking reading of hardware verification transcripts in **hol88**. We would pore over proofs, following each tactic application and its result. This approach has much to recommend it, but is perhaps overly monastic for the students of today. Plus, the range of technologies available for disseminating information is much greater today.

A successful course by Tom Melham was used to teach the fundamentals of HOL and the **hol88** system for many years. The course provided a thorough, bottom-up route to facility with HOL: first, ML was taught, then the basics of the logic (types, term formation, primitive rules of inference), then more advanced topics such as tactics, then higher-level packages, and finally extended case studies. The course could be adjusted to various time frames, up to a full week. Both the Isabelle and PVS communities have also conducted similar courses, and trained many accomplished researchers as a result.

2 Course Structure

The course was advertised as a graduate-level course with the only requirement being “desire and ability to reason formally”. In the end sixteen students (some undergraduates) completed the course, and a number of others sat in. None of the participants had any knowledge of mechanized theorem proving at the start of the course. Only a few had been exposed to ML.

The first month and a half of the course were devoted to learning the basics of the HOL-4 system. This involved approximately two weeks learning ML, and two assignments:

- A collection of proofs in the theories of lists and regular expressions.
- Proofs about summation (Σ), applications of decision procedures, and proofs in set theory

In all exercises in the assignments, the goals were explicitly given so that students did not have to formulate correct goals.

Classroom lectures on were first focused on gaining familiarity with how to make definitions and express statements in the HOL logic. After that, proof tools were discussed, beginning with high-level proof tools (the simplifier, first order proof search, and decision procedures) and finishing with low-level tactics. Only a slight amount of discussion was devoted to forward inference vs. tactics: the ongoing assumption was that most proofs would be performed by tactic proof.¹ In general, high-level proof tools were given much more emphasis than low-level tools. In retrospect, the emphasis on highly automated tools was one of the more arguable choices.

Subsequently, the students submitted a project proposal and embarked on a project, which took the duration of the course. The remainder of the lectures in the course were given over to a survey of automated theorem proving. This included the following topics:

- Equational Logic
 - unification and matching
 - proofs in equational logic
 - rewriting (unconditional and conditional)
 - how to write your own simplifier in ML
- Propositional Logic
 - proof, satisfaction, and refutation
 - normal forms
 - Soundness and Completeness
 - proof procedures (truth tables, resolution, and DPLL)
- First Order Logic
 - syntax, proofs, and Skolemization
 - models (Soundness, Deduction Theorem, Compactness, Completeness (Henkin’s proof))

¹ In some cases, this hampered students in their project work.

- Decision Procedures
 - Decidability, undecidability
 - Fourier-Motzkin (Dense Linear Orders)
 - Quantifier Elimination, Cooper’s algorithm

3 Projects

One of the prime reasons for the spread of higher order logic theorem provers is their expressive power: since these systems can basically formalize ‘anything of interest’, there is scope for a wide range of formalizations. This is reflected in the following projects which were proposed and undertaken by the students. Some students originally proposed projects (*e.g.*, Lie Algebra) requiring resources beyond those of the current version of HOL-4; in those cases, the instructor tried to point students towards more immediately fruitful proposals. However, in general little control of content was exercised, other than that the student had to know the subject matter well.

3.1 Expression Compiler

Initially, a compiler for a language with assignments and While loops was proposed. Specifying the implementation and correctness took some time, but real problems only occurred when the proofs were started (goals were simply too large). This prompted a retreat to simpler languages. Eventually, a compiler for arithmetic expressions was proved correct. The correctness proof required a significant generalization of the goal, which required instructor intervention.

3.2 Gödel Encoding

Gödelizing a formula means to translate it to a unique number. The project was to define the encoding and decoding functions and prove that decoding inverts encoding. (The student also re-formalized much of basic arithmetic, in order to gain more experience in theorem proving.)

3.3 Regular Languages

Automata and regular languages were formalized, and standard meta-theorems were proved: closure of regular languages under boolean operations and Kleene star. Also the pumping lemma for regular languages was proved. The proof of the lemma requires the PigeonHole Principle, in a slightly non-standard form, which was not already available in HOL-4. Finally, the pumping lemma was applied to show the non-regularity of some specific languages. An interesting aspect of the work is that the student discovered that the obvious definition of regularity in HOL will not work, since it has two type variables occurring in the right-hand-side of the definition, and only one on the left-hand-side.

3.4 Attempting to Break SHA-1 using SAT

The SHA-1 cryptographic hash function is heavily used in security applications, although there have been recent reports that it has been broken. The first phase of the project formalized SHA-1 as a functional program and then attempted to find pairs of plaintext that would hash to the same value. This can be mapped to a SAT problem by symbolically executing SHA-1. The attempt was ultimately unsuccessful, and it is not clear this path is feasible. The first problem was that the result of symbolic execution was not in CNF, and the conversion to CNF using the existing HOL-4 mechanism was very slow. Since SHA-1 does 80 rounds of hashing, we reduced the number of rounds. At one round, things were feasible, but at two, they were not. Formulas became so large that it wasn't clear what was going on.

3.5 IDEA Crypto Algorithm

The IDEA encryption algorithm is used in PGP (Pretty Good Privacy) and provides another application of symbolic execution to prove functional correctness of encryption algorithms [4]. The correctness of IDEA requires that its version of multiplication ($\text{mod } 2^{16} + 1$) has an inverse. This required much work, the full formalization is discussed in [5].

3.6 Hardware Adders

Hardware verification was a popular topic: several projects on proving correctness of circuits were undertaken. One reason for this popularity may have been that a course on fast hardware implementation of arithmetic operations was being delivered by Erik Brunvand in the same semester². Basic implementations such as ripple-carry and carry-lookahead adders were proved correct. More advanced prefix adders (Ladner-Fisher and Brent-Kung) were also verified. Mostly, conventional $imp \supset spec$ proofs were performed; however, one project took a somewhat different tack, using a Perl module for parsing Verilog to parse (fixed width) Verilog circuit descriptions into HOL terms, and then checking their equivalence with a ripple-carry adder using `HolSatLib`, HOL-4's interface to commonly-used SAT solvers.

3.7 Hardware Multipliers

The class of *Carry/Save Array* multipliers were formalized, starting from the transistor level, and applied to prove the unsigned Baugh-Wooley multiplier correct. The definitions were quite general, and provide a platform from which the correctness of other fast multipliers can be proved with less effort.

² Webpage: <http://www.cs.utah.edu/classes/cs5830>

3.8 Abstract Interpretation of Analog/Mixed Signal Circuits

Analog/mixed circuits are becoming more prevalent in implementations. However, formal tools for dealing with their properties are few and far between and haven't managed to scale well yet. One solution to this problem is to discretize the behaviour, then run finite-state tools. But then the problem is correctness of the abstraction. The project was just to show the essence of this correctness argument, where behaviour is abstracted to 2D *zones* in the plane.

3.9 FFT over Polynomial Trees

This project worked from a paper by Venanzio Capretta on verifying the Fast Fourier Transform in Type Theory[1]. There was significant overhead to overcome, especially absorbing Type Theory after one had just managed to learn the basics of higher order logic. Ultimately, the major challenge was formulating and applying induction principles for the underlying tree representation used in the original paper.

3.10 Matrices

The theory of matrices was tackled. Initial attempts at defining a matrix as a list of lists led to complex proofs for simple goals, involving nested induction. This led to an exploration of a number of other formalizations, ending with the representation of a matrix as a triple, consisting of row bound, column bound, and indexing function. Based on this, the theory was developed up to the associativity of matrix multiplication. A large part of the effort was taken up by defining and developing a theory of summations.

3.11 Groups

The algebra of groups was tackled, and progressed to the First Isomorphism Theorem. The definition of groups was simple, but significant problems were encountered in applying the usual proof tools (first order proof search and simplification) in algebraic derivations. Simple identities were indeed simple, but solving group membership side-conditions needs special proof support (naively applying first order proof search works only in simple settings). Another challenge was dealing with the quotient construction in the Isomorphism theorem. Initially, Hilbert's Choice operator was used, but this led to the usual difficulties. Happily, Skolemization was applicable instead.

3.12 Mutual Exclusion

The mutual exclusion problem was formulated set-theoretically, and the theorem that at most one process could be in the critical section at one time was proved. Each process was formalized as a non-deterministic finite-state machine, and a global scheduler (an application of the Axiom of Choice) was used to arbitrarily pick from multiple processes attempting to enter the critical section. An interesting next step in this effort would be to deal with fairness and liveness.

3.13 Thread Interleaving and Transactions

The Reduction Theorem from the research paper [2] was formalized and proven. The effort to follow the informal presentation in the paper provided the main difficulty, but in the end the informal definitions and proof were clarified. Unlike many of the other projects, the number of definitions and the size of the statement of the theorem were quite large; there was therefore a large component of making sure that definitions actually captured the intended ideas.

3.14 Program Transformation

It is well-known that (subject to some constraints) linear recursions can be translated to tail recursions. However, it has been an ongoing challenge to automate this; the project was to implement a simple version of the translation, supported by formal proof. The code was applied to common recursive functions in the theory of numbers and lists, *e.g.*, factorial, reverse, flattening a list of lists, *etc.* The final version of the tool took a linear recursion, analyzed it, defined the corresponding tail recursion, and used automated proof to return an equality relating the two.

3.15 Java Program Verification

Krakatoa [3] is a system for verifying Java programs annotated with JML assertions. It is parameterized with backends for various theorem provers (the Coq port is most highly developed, but ports also exist for PVS, Harvey, CVC-Lite, among others). Integrating a HOL-4 backend and pushing through some standard examples required overcoming a wide range of obstacles. The HOL-4 port has recently been integrated into the Krakatoa release.

4 Comments from the Instructor

The students came up with an impressive range of examples, far more—and better—than I would have been able to come up with alone. Letting students choose projects in the intersection of their interest and competence is a good idea, although it can mean that the instructor is ‘at sea’ in some discussions. Since the instructor is there to provide theorem proving expertise, that lack of domain knowledge wasn’t usually a problem.

The main problem for the instructor was the sheer amount of time and energy required to teach students enough so they could ‘stand on their own two feet’. My office was packed for the entire semester with students seeking help. The lack of a Teaching Assistant was a contributing factor, but, had there been a TA, he/she would have been also completely swamped. The learning curve is very steep! This was mitigated by other factors, chief among them being that it is fun to help people learn theorem proving.

We have to admit that interactive theorem provers are very powerful but also bewildering:

- Simply managing to formulate correct statements (well-formed types, well-formed terms, sensible goals) is a significant hurdle. It is very easy to get into situations where complex formulations (*e.g.*, uniqueness, maximality, relativized statements, dealing with partial functions) require some logic sophistication.
- Finding the correct tool to use at any point can be difficult. There can be a ‘superstitious’ aspect to tool use, wherein, once a tool is found to solve one goal, the user tries to use it in all situations, when in fact other tools would be more appropriate. For example, first order proof search can be magically useful, but it can flounder on goals easily dealt with by a simplifier.
- Finding pre-proved theorem to apply (out of 11,000 or so), or even remembering how to look for existing theorems can be hard. This of course is an ongoing problem with all interactive proof systems.

A common response to the question *How do I do X in HOL?* is *Well, I know at least three ways*. This flexibility is good for experts, and frustrating for beginners.

Part of getting a proof tool user to stand on their own two feet is teaching them a workable proof methodology. Being able to decompose a verification into a series of definitions and lemmas is fundamental to success. Fortunately, this was not a difficult notion for the students and seemed to be naturally taken up.

There is an astonishing range of abilities when it comes to ‘picking up’ a piece of software and learning to use it effectively. Interactive theorem provers like HOL-4 provide a wide range of theories, features, tools, documentation, and so forth. Arriving at a mental picture of what the tool *is* was a struggle for some. (Simply saying that HOL-4 is a system for generating proofs in Higher Order Logic isn’t enough, alas.) Some students had little trouble navigating the system, while others were quickly blocked. How to eliminate this difference is an important problem.

A common phenomenon is that of *dialing-back*: the original proposal was far too ambitious, even with the requirement that the project proposal had to include a ‘fall-back’ plan. As the course played out, expectations had to be continuously revised. This happens with all projects, of course.

A more characteristic difficulty with theorem proving is the unpleasant phenomenon of *Being Stuck*. One can easily get into this situation by stating false goals, not knowing the proof, not being able to formalize a proof, *etc.* It is not catastrophic to Be Stuck, provided it is a temporary state of affairs. However, some students were extremely uncomfortable with the idea. This may be a principal difference between programming and proving: usually there is a way forward in programming, while getting Stuck is always a danger in formal proofs.

A majority of the students became quite enthusiastic. Quite a few projects have continued after the end of the course, sometimes because the student wanted to ‘finish that last theorem’, and sometimes because the project lies within their wider research goals.

Finally, given the project nature of the course, it was unclear whether the supplementary lectures were useful. However, attendance was surprisingly good all the way through the course. It is not clear whether this is because the content was so interesting or because of inertia.

5 Comments from the Students

Here are a mostly undigested list of comments from the students. Some of the comments reflect the students' lack of experience, but that's just the point: these are typical comments from beginners, just after they've finished the main task of learning a new system.

- The system is vast and frustrating, I can't get it to do what I want.
- The command-line interface is too primitive.
- It is easy to forget how to get things done.
- Using pre-proved theorems with automated theorem provers on some of the assignments was easy and fun.
- Searching for a suitable theorem for use by a proof procedure is a nightmare.
- I was surprised by case-sensitivity.
- Mostly the high-level tactics are not useful and low-level tactics had to be used for most proofs.
- It would be nice if tactics printed out useful information when they fail.
- It is amazing that the HOL system contains such an enormous set of theories and proofs.
- There should already be a library supporting verification of arithmetic circuits.
- More homework and representative examples before starting the project
- Dealing with assumptions is often irritating: they manage to confuse the automated provers when there are many assumptions.
- I had difficulty understanding error messages.
- The documentation needs more examples and explanations for students.
- There is a lack of examples when developing proofs
- Performing a proof step can cause 'things to magically change' and I don't understand how the change happened.
- I was unable to get automated reasoners to understand set notation
- I thought the most difficult part would be to do the reasoning steps but in fact getting the definitions right was the hard part.
- HOL itself was agreeable once things got rolling. The tool support I found most lacking was creating tactics. This is because it was time consuming and confusing to figure out when and where the case splits were in larger proofs
- It is good that each recursively defined function automatically gets its own induction theorem.
- It would be helpful if HOL supported the deletion of assumptions that are no longer needed.
- Seemingly obvious paths are not obvious to HOL.

- In many instances, the advantages of using case analysis vs. induction was not clear.
- Defining a procedural process as a provable statement of logic required a lot of effort. But once defined, the statement appeared simple, even elegant
- While the proof clearly was evident on paper, translating the steps into HOL was not intuitive, it seemed arcane.
- The tutorial and reference guides were not helpful, and the online help was less so.
- Develop a ‘How-To’ for various commonly performed proof steps.
- Write a FAQ for converting ML to HOL.
- HOL has a well-formed set theory, but more theorems could be added to HOL, especially cardinality theorems and equalities between set expressions.
- Writing and proving the universe was, perhaps, too large a jump.
- I feel that I outperformed myself in this project. I didn’t plan to complete the whole thing, because the verification of ... alone requires the effort of a project. But the topic of this project really intrigued me, so I spent double time and efforts on it.

6 Conclusion

In the instructor’s view the course results were quite encouraging: he had been expecting much resistance to the idea and practice of formal methods as performed in HOL-4, but most of the projects were interesting and successful. Although it was hard work (more than one student called it the hardest course they had ever taken), the majority of students did manage to learn enough to make a large and complex proof system do what they wanted.

One interesting issue is: *given that there is a finite amount of time (and student patience), what proof tools should be emphasized?*. This course took the approach of focusing on the most highly automated tools. However, once the going gets tough in a proof, more specialized steps may need to be taken (simplifying only one specific occurrence of a term, explicitly instantiating a theorem, *etc*) and the students did not have an adequate basis for going ahead with such steps. Possibly an assignment explicitly addressing such thorny aspects could be constructed.

One observation is that most of the successful projects started from previous formalizations, or at least a correct set of definitions. Perhaps attempting to solidify definitions and get substantial proofs done is too much of a burden for beginners. On the other hand, we don’t want to stifle application of the proof tools to already settled mathematics; several of the most successful projects of the course did involve much wrestling with definitions.

Next time the course is taught, there will be a more significant attempt at integrating the lectures with the content; otherwise, the lectures don’t seem to bear much relationship with the verifications in progress. Possibly another assignment could be added so that students can gain experience with writing their own proof tool, one of the major applications of HOL-4.

Focusing on a particular proof language, as is done with the Isar language in Isabelle could also ease the burden on the students, although understanding the behaviour of complex simplifiers and decision procedures would still be required.

In many universities, first order logic and its meta-theory, such as the soundness and completeness theorems, are routinely taught to undergraduates. A tool-supported undergraduate course in Higher Order Logic would seem to offer new benefits, *e.g.*, more compelling examples. The tool support, thanks largely to the long-term effort of the TPHOLs community, is substantially there; all that needs to be produced is an attractive curriculum for undergraduates. A project-based course described here still seems best for the graduate level though.

References

1. Venanzio Capretta. Certifying the Fast Fourier Transform with Coq. In Richard J. Boulton and Paul B. Jackson, editors, *Theorem Proving in Higher Order Logics: 14th International Conference, TPHOLs 2001*, volume 2152 of *Lecture Notes in Computer Science*, pages 154–168. Springer-Verlag, 2001.
2. Cormac Flanagan and Shaz Qadeer. Types for atomicity. In *Types in Language Design and Implementation (TLDI 2003)*, pages 1–12. ACM, 2003.
3. Claude Marche', Christine Paulin-Mohring, and Xavier Urbain. The KRAKATOA tool for certification of JAVA/JAVACARD programs annotated in JML. *Journal of Logic and Algebraic Programming*, 2005. To appear.
4. Konrad Slind. A verification of Rijndael in HOL. In V. A Carreno, C. A. Munoz, and S. Tahar, editors, *Supplementary Proceedings of TPHOLs 2002*, number CP-2002-211736 in NASA Conference Proceedings, August 2002.
5. Junxing Zhang and Konrad Slind. Verification of Euclid's algorithm for finding multiplicative inverses. In Joe Hurd, editor, *Emerging Trends: Theorem Proving in Higher Order Logics: 18th International Conference, TPHOLs 2005*, 2005. (If accepted).