Teaching Experience: Logic and Formal Methods with Coq

Martin Henz and Aquinas Hobor*

National University of Singapore

Abstract. During the past three years we have been integrating mechanized theorem proving into a traditional introductory course on formal methods. We explain our goals for adding mechanized provers to the course, and illustrate how we have integrated the provers into our syllabus to meet those goals. We also document some of the teaching materials we have developed for the course to date, and what our experiences have been like.

1 Introduction

National University of Singapore's School of Computing teaches introductory formal methods as CS3234 (undergraduate) and CS5209 (graduate). In 2007 and 2008 the first author taught CS3234 using a traditional approach with the standard undergraduate textbooks *Mathematical Logic for Computer Science* [BA01] and *Logic in Computer Science* [HR00]. Sad to say, the results were equally "traditional":

1. The module was "hard" in the eyes the students due to the necessity of understanding of an unusual number concepts on several abstraction levels.
2. Students viewed formal systems as a subject far removed from useful applications.
3. Weaker students often found exercises and tutorials unusually "dry" and "boring".

The first point made for a steep learning curve, the second decreased the motivation of students to climb the curve, and the third posed further obstacles for those students who have enough motivation to even try. In short, there was clear room for improvement.

When the second author joined the team we proceeded to address these problems (after acknowledging the first one as only partially solvable). The goal was to shorten the gap between theory and practice by providing relevant and appealing applications and to implement a "hands-on" approach by introducing adequate didactic tools.

Several tools are popularly used to teach formal systems in computer science, including logic programming systems, model checkers, and SAT solvers. We found it difficult to justify the learning overhead that these tools require given that they are often only used for one or two sections of a module. Ideally, the same tool would be used *throughout* the module, reducing overhead to a minimum and allowing for more sophisticated use as the course progressed into more complex territory.

We determined to use the proof assistant Coq. While not having been developed specifically for didactic use, Coq's basic concepts have proved to be sufficiently easy for third year undergraduates (and even, sometimes, for graduate students). Initial results have been encouraging: the interactive discovery of proofs using Coq provided a useful

* Supported by a Lee Kuan Yew Postdoctoral Fellowship.

reinforcement of the conceptual material, and we have been successful in integrating the theorem prover into almost every part of the course. We feel that the consistent use of Coq has improved the students' comprehension of formal logic, and the quality of their proofs, as long as they are willing to invest the time for learning Coq.

Remainder of paper. We next go through our course syllabus, focusing for each topic on how we have added mechanized proving to a more traditional curriculum. We then describe the course format (*e.g.*, the number of assignments, weighting of various components in the final grade) and explain its rationale. We conclude with a discussion of the feedback we have received from our students and our own experiences.

Associated material. We developed a substantial amount of material (hundreds of pages) as part of modifying this course, including slides, lecture notes, homework assignments (both paper and Coq), laboratory exercises, Coq quizzes, and exams [HH10].[1] For much of the course this material was the primary reference for the students. When appropriate in what follows we shall provide pointers into specific parts of this material; readers are kindly reminded that this supplementary material is drawn from several iterations of the same course and is very much a work in progress. We eventually hope to package this material into some kind of book.

2 Syllabus

Orientation. The National University of Singapore (NUS) follows a relatively short 13-week semester. After week 13, students have a reading period before exams. In recent years, CS3234 has had between 30 and 37 students, with an unusually large number drawn from the strongest undergraduate students in the School of Computing. In contrast, CS5209 often has more than 50 students, largely because one of the qualifying exams (required to proceed with the PhD program) covers formal methods.

2.1 Traditional Logic: weeks 1 and 2

Motivation. Usually, courses in formal methods in computer science start with propositional logic because it is the simplest modern formal logical system. The challenge is that students are presented very early with substantially new concepts on two levels.

The conceptual level: the distinction of syntax and semantics, what constitutes a proof, proof theory (natural deduction), and semantic arguments (models).

The logic-specific level: Propositional formulas as elements of an inductively defined set (or of a context-free grammar), introduction and elimination rules for propositional logic, and a valuation-based semantics (truth tables).

We found it desirable to pursue a gentler approach at the beginning of the course, aiming for a shallower learning curve. The idea is to start with a logic framework that enjoys very simple syntax, semantics and formal reasoning techniques, allowing the students to focus on and properly digest the conceptual components. This approach will also give us the opportunity to introduce the nuts and bolts of Coq gently.

[1] Note: readers interested in seeing the solutions to the assignments and exams should contact us directly.

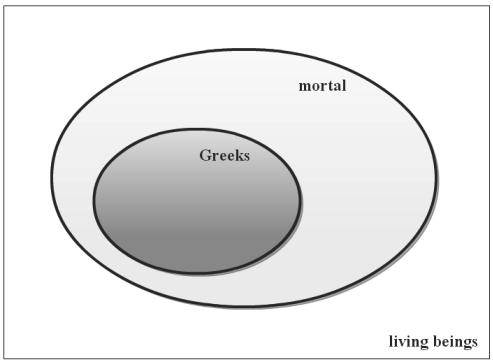

Fig. 1. Venn diagram for "All Greeks are mortal"

We believe that Aristotle's term logic [PH91] is appropriate for this purpose. Among several possible encodings of term logic in Coq, we chose a version that combines simplicity with consistency of presentation, compared to the next logic, propositional logic. Students get familiar with basic Coq concepts like definitions, proofs, and tactics, without getting bogged down in complex inductively defined syntax and semantics.

Basic components of term logic. The atomic unit of meaning in term logic are *categorical terms, e.g.*, `humans`, `Greeks`, and `mortal`. We encode this in Coq as follows:

```
Parameter Term : Type.
Parameters Greeks humans mortal : Term.
```

A *categorical proposition* then puts two such terms together as in the famous universal proposition "all Greeks are humans." Besides the quantity "universal" (all), we provide for "particular" (some) propositions, and beside the quality "affirmative", we provide for "negative" propositions, leading the the following definitions in Coq:

```
Structure Quantity : Type := universal | particular.
Structure Quality : Type := affirmative | negative.
```

Data structures of type `CategoricalProposition` are then constructed from a `Quantity`, a `Quality`, a subject `Term` and an object `Term`.

```
Record CategoricalProposition : Type := cp {
  quantity : Quantity;
  quality : Quality;
  subject : Term;
  object : Term }.
```

An appropriate Coq `Notation` enables the students to (most of the time) write propositions as natural English sentences such as `All Greeks are humans`.

Semantics from naïve set theory. A model $\mathcal{M}$ for a term logic can be given by providing a universe of objects $U^{\mathcal{M}}$, and a subset (or unary predicate) $t^{\mathcal{M}} \subseteq U^{\mathcal{M}}$, for each term t. The semantics of a universal proposition is then given by

$$(\texttt{All}\ \textit{subject}\ \texttt{are}\ \textit{object})^{\mathcal{M}} = \begin{cases} T & \text{if } \textit{subject}^{\mathcal{M}} \subseteq \textit{object}^{\mathcal{M}}, \\ F & \text{otherwise} \end{cases}$$

and can be visualized by a *Venn diagram* as in Figure 1. The reader can see [HH10, `notes/Traditional.pdf`] for the full exposition.

Introducing logical concepts. Categorical propositions are lifted into `Prop` using

```
Parameter holds : CategoricalProposition -> Prop.
```

consistent with and in preparation for more complex `holds` predicates introduced in propositional and modal logic. Facts can then be introduced interactively, as in:

```
Axiom HumansMortality: holds (All humans are mortal).
Axiom GreeksHumanity: holds (All Greeks are humans).
```

Such factual "axioms" allow for a quick access to reasonable examples. However, the rigidity of this style of reasoning does not escape the attentive students.

A graphical notation for axioms prepares the ground for natural deduction:

$$\frac{}{\texttt{All humans are mortal}}\,[\text{HumansMortality}]$$

A more interesting axiom—traditionally called Barbara as a mnemonic device—expresses transitivity of the subset relation:

$$\frac{\texttt{All}\ \textit{middle}\ \texttt{are}\ \textit{major} \qquad \texttt{All}\ \textit{minor}\ \texttt{are}\ \textit{middle}}{\texttt{All}\ \textit{minor}\ \texttt{are}\ \textit{major}}\,[\text{Barbara}]$$

Its representation in Coq introduces conjunction and implication at the meta-level.

```
Axiom Barbara : forall major minor middle,
    holds (All middle are major) /\ holds (All minor are middle)
 -> holds (All minor are major).
```

Basic tactics such as `split` can be observed in action in this proof of Greek mortality:

```
Lemma GreeksMortality : holds (All Greeks are mortal).
Proof.
  apply Barbara with (middle := humans).
  split.
  apply HumansMortality.
  apply GreeksHumanity.
Qed.
```

Interactive proof sessions. Equipped with the basic reasoning techniques of traditional logic, students can now proceed to more complex proofs. An attractive realm of "applications" are Lewis Carroll's logical puzzles. For example, from the following premises

- No ducks waltz.
- No officers ever decline to waltz.
- All my poultry are ducks.

we should be able to conclude, naturally enough, that no officers are my poultry. After defining appropriate terms such as `things_that_waltz` and a complement constructor for negative terms (`non`), we can define the corresponding lemma in Coq:

```
Lemma No_Officers_Are_My_Poulty :
   holds (No ducks are things_that_waltz) /\
   holds (No officers are non things_that_waltz) /\
   holds (All my_poultry are ducks)
   ->
   holds (No officers are my_poultry).
```

The proof uses tactics that implement traditional Aristotelian reasoning techniques such as obversion and contraposition [Bor06]; the interested reader is referred to [HH10, `notes/Traditional.pdf`] for details on their implementation in Coq. Attentive students now realize that in the proof, assumptions play the role of earlier factual axioms, but have the advantage of being localized to the proof.

We are able to cover the basics of Aristotelian term logic in a week and a half (the first half week being reserved for standard course introductory material such as statements on course policy). Afterwards, the students have achieved a basic understanding of the syntax/semantics distinction, models, axioms, lemmas, proofs, and tactics (all of which of course to be repeatedly reinforced throughout the course), and are thus ready to focus on the logic-specific aspects of propositional logic.

Later in the course—equipped with predicate logic—students can go back to their first logic, represent categorical terms by unary predicates, and prove Aristotle's axioms such as Barbara as lemmas in an easy exercise:

```
Lemma BarbaraInPred: forall (major minor middle: term -> Prop),
  forall x, ((middle(x) -> major(x)) /\ (minor(x) -> middle(x)))
            ->
            (minor(x) -> major(x)).
```

2.2 Propositional Logic: weeks 3 and 4

Prelude: rule-defined sets as data structures. First, we have to give some kind of intuition for what an inductive set is, so that we can define the syntax of the logic. However, we would prefer to defer formal discussion of such sets and their associated proof rules (*i.e.*, induction) until after we cover predicate logic in week 8 (§2.5).

We have discovered that the simplest way to give this intuition is to take advantage of the fact that we are teaching computer science majors, and make the connection between a (simple) inductive type and a (simple) data structure, such as a linked list. We provide some simple Java code that uses provides poor man's versions of the natural numbers (with `Zero` and `Succ(...)`) and binary trees; we then demonstrate the corresponding Coq code for these types as well [HH10, `notes/Induction.pdf`]. This would be simpler, of course, if our students knew ML, but we do not have that luxury. In practice demonstrating the idea in Java provides some intuition and reinforces the idea that logical formulas have a well-defined structure; in addition we can use the Java code to start to address interesting questions, *e.g.*, about cyclic data structures.

Encoding as an object logic. Our presentation of paper-based propositional logic is entirely standard: we give the syntax of formulas, define the semantics (*e.g.*, valuations, truth tables), give the natural deduction rules, and cover soundness/completeness. One small departure from the norm is that we bring up the idea of intuitionistic logic quite early, and informally explain its connection to computability.

More interesting is how we cover the topic in Coq. Because we already introduced some basic Coq in week 2 (§2.1), we have the advantage that some basic Coq concepts and tactics (e.g., implication and `intros`) are already sinking in. To reinforce that idea, and to keep the concepts of object- and metalogic strictly separate, we first cover propositional logic as an object logic and hew closely to how we defined it on paper. That is, we inductively define the syntax of formulas, introduce the idea of

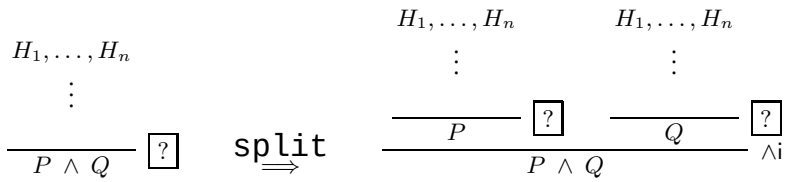

Fig. 2. A diagram that explains the `split` tactic.

a valuation, and then define an evaluator fixpoint that takes a formula and a valuation and produces a boolean value according to the standard truth table rules [HH10, `notes/Propositional_Logic.pdf`].

We then provide a module type that gives the various natural deduction rules, and assign two kinds of homework: first, we require that they use those rules to prove various standard problems in propositional logic using machine-checked natural deduction. One big advantage of the object-logic encoding is that they must use axioms explicitly (*e.g.* `apply Conj_I.`) instead of the typical Coq tactics (`split.`). We have found that the built-in tactics do a bit too much (*e.g.*, many overloadings for the `destruct` tactic), and by explicitly requiring named axioms we can match in Coq the exact shape of paper-based natural deduction proofs. For the second part of the homework, we require that they implement a module matching that module type, thereby proving the soundness of the rules [HH10, `coq/homework_02.v`]. For natural deduction, we encourage cross-pollination by assigning some of the same problems in both the paper portion of the homework and in the Coq portion.

Switching between the object logic and the meta logic. Once students have a handle on propositional logic, and have gotten a bit more of a feel for Coq, it is time to ask an obvious question: why are we defining our own version of conjunction, when Coq already provides it? In fact, it is extremely convenient to use Coq's built-in operators, since it greatly enhances automation possibilities[2]. We give some problems to ensure that students are familiar with the basic tactics; by this point, for most these are quite simple [HH10, `coq/Propositional_Logic_Lab2.v`].

Explaining what Coq is (approximately) doing. Students tend to be mystified by what Coq is exactly doing. This leads to undue "hacking", wherein students try tactics at random until for some unknown reason they hit on the right combination. After we have introduced propositional logic and the students have done some Coq homework, we try to explain what is going on by a series of diagrams like the one in Figure 2.

This kind of diagram shows a proof state transformation with the pre-state to the left and the post-state to the right. Here we show the transformation for the `split` tactic; the goal of the pre-state, appropriately enough, is a conjunction $P \wedge Q$. We have a series of hypotheses $H_1, \ldots, H_n$, but Coq is not sure how to proceed from them to the goal; we symbolize this by labeling the rule with a question mark (boxed for emphasis). The `split` tactic tells Coq that the two parts will be proven independently from our hypotheses; accordingly, afterwards, Coq presents us with two fresh goals: P and Q, and again asks us how to proceed. Coq has inserted the conjunction introduction axiom ($\wedge$i) to connect those goals into a proof of the conjunction.

[2] And when we get to predicate logic the ability to offload binders onto Coq is a godsend.

We have found that students understand the tactics much more clearly after we demonstrate the transformations they perform by using these kinds of diagrams. As an aside, one time we ran the course, we provided a series of tactics that followed the axioms of propositional logic a bit more clearly (*e.g.*, we defined a tactic `disj_e` that was identical to `destruct`). This turned out to be a bad idea: not only did it limit students' ability to look up documentation online, but it meant that they were not really able to use Coq in a standard style after completing the course.

Case study: tournament scheduling. Too often, formal systems appear to have little practical application. To combat this perception, we like to conclude our discussion of propositional logic with an example of using propositional logic to solve a computational problem, via its encoding as a propositional satisfiability problem: Hantao Zhang's encoding [Zha02] of the Atlantic Coast Conference 1997/98 benchmark [NT98], as a propositional formula. The fully automated proof of its satisfiability using the satisfiability checker SATO [Zha93] yields the solutions to the benchmark problem orders of magnitudes faster than techniques from operations research.

2.3 Predicate Logic: weeks 5 and 6

Just as in the case for propositional logic, our presentation of pen-and-paper predicate logic is largely standard: syntax, semantics, proof rules, metatheory. We have found that one place where our pen-and-paper explanation is aided by the use of the theorem prover is in substitution. It is quite simple to create some formulas in Coq and then use the `rewrite` tactic to substitute equalities, observing how Coq manages the binders.

Since we have already made the distinction between object- and metalogics, we take full advantage of Coq's binder management. That is, while we carefully define substitution for paper methods, and demonstrate how Coq handles the situation as explained above, we entirely avoid defining any mechanized substitution methods ourselves[3]. Among other advantages, this allows us to completely sidestep the quicksand of computable equality testing, which would be needed to define substitution in Coq.

Most of the tactics for predicate logic in Coq (`exists`, `destruct`, and `intros`) are fairly simple; one exception is the tactic for universal elimination (`generalize`), which is a little bit weird (why has my goal just changed?). Although usually we prefer to just teach the Coq tactics as-is, in this case we define a custom tactic that does a universal elimination by combining a `generalize` with an `intro` and a `clear`.

2.4 Midterm exam and case study: week 7

We find it convenient to give a midterm examination after predicate logic. This exam covers traditional, propositional, and predicate logic and is done entirely on paper. By this point, the students have already had several Coq quizzes, and so we are able to track their progress in the theorem prover that way. In addition, the logistics of running an exam in the laboratory are fairly complicated and so we only do so for the final (§3).

Network security analysis. After the midterm, the students are too jumpy to listen to anything formal, and so we do not want to start a fresh topic. Instead, just as with propositional logic, we like to present an example of applying predicate logic to a real-world problem, this time of network security analysis [OGA05].

[3] If students are interested we may briefly mention De Bruijn indices.

2.5 Formal Induction: week 8

After predicate logic, we return to the subject of induction. Whereas our treatment of induction in week 3 (§2.2) was informal and by analogy to data structures in computer science, by week 8 we are ready to be quite formal.

In previous years, we discovered that students had an extremely hard time understanding the nuances of formal structural induction; common errors include: not covering all cases, not proving the induction hypothesis in a case, assuming the wrong induction hypothesis in a case, failing to generalize the induction hypothesis, etc. The advantage of deferring the formal treatment of induction until after predicate logic is that students have enough familiarity with Coq to be able to use it to explore the topic. The payoff is substantial; indeed, the single biggest improvement in comprehension for an individual topic occurred after we introduced mechanized induction.

We were not able to find a textbook that covered structural induction in a way we were happy about; accordingly, we wrote some fairly extensive lecture notes on the subject [HH10, `notes/Induction.pdf`]. By doing so, we were able to develop the paper and mechanized versions of induction with similar notation and in parallel, which allows students to follow along with their own Coq session and experiment.

Another advantage of developing our own materials was that we are able to introduce several related topics that are "off the beaten track". For example, although we do not cover it in full detail, we find it useful to explain coinduction as a contrast to induction. We point out that for both inductive and coinductive types, case analysis forms the basic elimination rules and constructors form the basic introduction rules. Inductive types get a more powerful kind of elimination rule (fixpoints) whereas coinductive types get a more powerful kind of introduction rule (cofixpoints). We also point out the connection to nonterminating vs. terminating computation, a concept which connects back to earlier discussions about intuitionistic logic.

The end result was that most students were able to write extremely clear inductive proofs, even when the induction in question was not straightforward, *e.g.*, when it required a generalization of the induction hypothesis (including the often-confusing situations wherein quantifiers need rearrangement before induction).

Teaching with Coq becomes a bit entwined with teaching Coq. One of the challenges of using Coq as a didactic tool is that Coq is extremely complicated. It is amazing how easily one runs into all kinds of didactically-inconvenient topics at awkward moments. We try to sprinkle in some of these ideas ahead of time, so that when they come up later students already have some context. Moreover, covering the nitty-gritty details further a minor goal, which is to provide the students with a better understanding of Coq, in case they want to use it going forward for another class or a research project—and indeed, several did so. While discussing induction we also cover the ideas of pattern-matching[4], exhaustive/redundant matching, polymorphic types, and implicit arguments.

[4] One detail we have largely avoided discussing is the distinction between computable and incomputable tests for equality—*i.e.*, those that live in `Type` vs. `Prop`. This might be a mistake; one of the advantages of using a mechanical theorem prover is that it is easy to demonstrate the importance of maintaining the computable/incomputable distinction by simply observing that Coq can do much less automation when computability is not maintained.

2.6 Modal Logic: weeks 9 and 10

Introducing modal logic with Coq was a bit challenging. There are two main problems:

1. The semantics of modal logic is usually introduced on paper by defining a finite set of worlds, each of which is a finite set of propositional atoms. The relation between worlds is then a finite set of arrows linking the worlds. Immediately this runs into trouble in Coq—an example of the already mentioned propensity for Coq to force unpleasant didactic issues to the fore, *e.g.*, Coq does not have a simple way to encode finite sets without using library code and explaining the importance of constructive tests for equality (both of which we have avoided in the past).
2. Coq does not have a clean way to carry out natural deduction proofs in modal logic. The best method we have found, a clever encoding by deWind, is still clunky when compared to simple paper proofs [dW01]. Current research in Coq using modal logic tends to prefer semantic methods over natural deduction—that is, modal logic is used to *state properties and goals* rather than *prove theorems*.

In the end, although our initial explanation of modal logic on paper was given in the standard propositional style, on the Coq side we decided to plunge headlong into a higher-order encoding of modal logic. Modal formulae are functions from the parameterized type of worlds into `Prop`, and we lift the usual logical operators (conjunction, etc.) from the metalogic. With judicious use of `Notation`, the formulas in Coq can look pretty close to how we write them on paper. Here is a small sample of our setup:

```
Definition Proposition : Type := world -> Prop.

Definition holds_in (w : world) (phi : Proposition) :=
  phi w.
Notation "w ||- phi" := (holds_in w phi) (at level 50).

Definition And (phi psi : Proposition) : Proposition :=
  fun w => (w ||- phi) /\ (w ||- psi).
Notation "phi && psi" := (And phi psi).
```

We also lift the universal and existential quantifiers from the metalogic, giving the students a first-order (at least) version of modal logic to play with[5]. Even better, if we are careful in how we lift the logical operators then the usual Coq tactics (`split`, etc.) work on modal logic formulas "as one might expect":

```
Goal forall w P Q,
  w ||- P && Q ->
  w ||- Q && P.
Proof.
  intros w P Q PandQholds.
  destruct PandQholds as [Pholds Qholds].
  split; [apply Qholds | apply Pholds].
Qed.
```

[5] In fact, we have given them something much more powerful: the quantification is fully impredicative, although we do not go into such details.

This is extremely useful since the cost of learning a new tactic is quite high to a student.

Since our students already have a grasp of quantification, they can understand when we define the modal box and diamond operators in the standard way (parameterized over some global binary relation between worlds R).

```
Definition Box (phi : Proposition) : Proposition :=
  fun w => forall w', R w w' -> (w' ||- phi).
Notation "[] phi" := (Box phi) (at level 15).

Definition Diamond (phi : Proposition) : Proposition :=
  fun w => exists w', R w w' /\ (w' ||- phi).
Notation "<> phi" := (Diamond phi) (at level 15).
```

To reason about these operators they must be `unfold`ed and then dealt with in the metalogic, but in practice we find that easier than trying to duplicate paper natural deduction proofs. In any event, encoding modal logic in this way allows the students to prove standard modal facts without undue stress, and in addition gives a feel for modal logics with quantifiers. We also introduce multimodal logics—logics with multiple relations between worlds, by parameterizing Box and Diamond:

```
Definition BoxR (R' : world -> world -> Prop)
               (phi : Proposition) : Proposition :=
  fun w => forall w', R' w w' -> (w' ||- phi).
```

We return to this idea when we study the semantics of Hoare logic in week 12 (§2.7).

Multimodal logics also lead into our investigation of correspondence theory—*i.e.*, the connection between the worlds relation R and the modal axioms. Here we are able use our Coq encoding of modal logic to demonstrate some very elegant proofs of some of the standard equivalences (*e.g.*, reflexive with T, transitive with 4) in a way that demonstrates the power of higher-order quantification, giving students a taste of richer logics. For more details see [HH10, `notes/Modal_Logic.pdf`].

2.7 Hoare Logic: weeks 11 and 12

We turn towards Hoare logic as we near the end of the semester. Our Coq integration was not very successful in helping students understand concrete program verifications. The problem seems to be that mechanically verifying even fairly simple programs leads to huge Coq scripts, and often into tedious algebraic manipulations (*e.g.*, $(n+1) \times m = n \times m + m$, where n and m are integers, not naturals). These kinds of goals tend to be obvious on paper, but were either boring or very frustrating for the students to prove in Coq. Accordingly, we did almost all of the program verifications on paper only.

There were two exceptions: first, we required the students to do a handful of extremely short (*e.g.*, two-command) program verifications in Coq, just to get a little taste of what they were like. Secondly, we showed them a verification of the 5-line factorial program given as the standard example of Hoare verification in Huth and Ryan [HR00]. Although the Coq verification was more than 100 lines, it was worth demonstrating, since it found a bug (or at least a woeful underspecification) in the standard textbook proof[6]. This got the key point across: one goes through the incredible hassle of me-

[6] The underspecification comes from not defining how the factorial function (in math, not in code) behaves on negative input, and the bug from not adjusting the verification accordingly.

chanically checking programs because it is the most thorough way to find mistakes; see [HH10, `slides/slides_11_b.color.pdf`, 46–56] for more detail.

Success on the semantic side. We had much better luck integrating Coq into our explanation of the semantics of Hoare logic. This is a topic that several introductory textbooks skip or only cover informally, but we found that Coq allowed us to cover it in considerable detail. In the end, our students were able to mechanically prove the soundness of Hoare logics of both partial and total correctness for a simple language[7]. The difficulty of these tasks were such that we think they demonstrate that our students had reached both a certain familiarity with Coq and a deeper understanding of Hoare logic.

Part of the challenge with providing a formal semantics for Hoare logic is the amount of theoretical machinery we need to introduce (*e.g.*, operational semantics). A second challenge is producing definitions that are simple enough to make sense to the students, while still allowing reasonably succinct proofs of the Hoare axioms.

Finding the right balance was not so easy, but after several attempts we think we have developed a good approach. We use a big-step operational semantics for our language; for most commands this is quite simple. However, the `While` command is a bit trickier; here our step relation recurses inductively, which means that programs that loop forever cannot be evaluated. Our language is simple enough (*e.g.*, no input/output) that this style of operational semantics is defensible, even if it is not completely standard.

Hoare logic as a species of modal logic. We use modal logic to give semantics to the Hoare tuple in the style of dynamic logic [HKT00]. One obvious advantage of such a choice is that *Hoare logic becomes an application for modal logic*—that is, it increases students' appreciation of the utility of the previous topic. This style allows the definitions to work out very beautifully, as follows. Suppose our (big-)step relation, written $c \vdash \rho \rightsquigarrow \rho'$, relates some starting context ρ to some terminal context ρ' after executing the command c. Define the family of context-relations indexed by commands S_c by

$$\rho \; S_c \; \rho' \qquad \equiv \qquad c \vdash \rho \rightsquigarrow \rho'$$

and the multimodal universal $\Box_c^S$ and existential $\Diamond_c^S$ operators as usual over S_c:

$$\begin{array}{lcl} \rho \models \Box_c^S P & \equiv & \forall \rho'.\ (\rho \; S_c \; \rho') \to (\rho' \models P) \\ \rho \models \Diamond_c^S P & \equiv & \exists \rho'.\ (\rho \; S_c \; \rho') \to (\rho' \models P) \end{array}$$

That is, if $\Box_c^S P$ holds on some state ρ, then P will hold on any state reachable after running the command c (recall that only terminating commands can be run); similarly, if $\Diamond_c^S P$ holds on some state ρ, then it is possible to execute the command c, and afterwards P will hold. Now we can give semantics to Hoare tuples as follows[8]:

$$\begin{array}{lcl} \{P\}\ c\ \{Q\} & \equiv & \forall \rho.\ \rho \models (P \Rightarrow \Box_c^S Q) \\ \big[P\big]\ c\ \big[Q\big] & \equiv & \forall \rho.\ \rho \models (P \Rightarrow \Diamond_c^S Q) \end{array}$$

[7] The proof of the `While` rule was extra credit. Several students solved this rule for the logic of partial correctness; to date we have not had any students solve the total correctness variant.

[8] Writing $\Rightarrow$ to mean (lifted) implication, *i.e.*, $\rho \models P \Rightarrow Q \;\equiv\; (\rho \models P) \to (\rho \models Q)$.

Although this style of definition is not suitable for more complicated languages, they work very well here and we find them to be aesthetically pleasing. Moreover, they lead to extremely elegant proofs of the standard Hoare rules. In fact, with the exception of the `While` rule for total correctness, none of the Hoare axioms took us more than about 10 lines of Coq script to prove, which put them within reach of our students' efforts[9]. This allowed us to give the entire soundness proof of the Hoare logic as a (fairly long) homework assignment. For more details, see [HH10, `notes/Hoare.pdf`].

2.8 Other Topics: week 13

The final week of the course is less formal and covers topics of interest to the instructors (*e.g.*, separation logic). Since there is no time to assign homework on topics covered, we do not want to get into huge amounts of detail, and any final exam questions on those topics are by convention fairly simple. In addition, we schedule part of the lecture for students' questions on material covered in the earlier part of the course.

2.9 What We Cut

We added several new topics to the standard topics covered in an introductory logic course (*e.g.*, by Huth and Ryan [HR00]): traditional (Aristotelian) logic, intuitionistic logic, more complex problems in predicate logic and induction, multimodal logic, the semantics of Hoare logic, and most of all the mechanical theorem prover Coq. Although we worked hard to preserve as much of the standard curriculum as we could, there were a few topics that we had to cut for reasons of time. While we covered what a SAT solver is, and explained that the problem was NP-complete, we did not explain any details of the kinds of heuristics involved to get good performance in practice. We also cut most of the material on model checking and temporal logic.

3 Course Format

We found it crucial for the students to acquire familiarity with Coq early in the course. Accordingly, we gave Coq assignments and quizzes. This resulted in a student workload that was significantly above average for comparable courses, since we did not compromise on the number of traditional paper-based assignments. As a result, the assessment components in the latest incarnation of CS3234 (Sem 1 2010/2011) included: 7 paper assignments (at 2% each), 5 Coq assignments (at 2% each), 6 twenty minute Coq quizzes (at 2% each), a one hour paper midterm (10%), and a two hour final with both Coq and paper problems (22% in Coq, 32% on paper). As one might imagine, preparing and grading this many assignments requires a serious commitment on the part of the instructors as well—and in addition, we were preparing course slides, lecture notes, and laboratory exercises. Fortunately, our department was able to allocate two teaching assistants to help giving the tutorials/laboratories and doing some of the grading; we ended up having one of the highest support/student ratios in the department. In the previous year (Sem 1 2009/2010) we did it all ourselves, and we had very little time to do other work. Of course, as we continue to develop and can begin to reuse the course materials, a good part of the labor is reduced.

[9] A useful rule of thumb when setting assignments: if the instructors can solve something in n lines, most of the students can solve the same thing in fewer than $5n$ lines.

When we last taught the graduate version CS5209 (Sem 2 2009/2010), we tried to assign less homework, hoping that graduate students would be able to learn the material without as much supervision. We were mistaken; quite a few of our graduate students had a very hard time with Coq, which was related to the lesser amount of homework. In the future we will assign more work in CS5209. We also tried to give some of the material as a group project; this also turned out to be a bad idea as some of the team members did not put in nearly enough work to do well on the Coq part of the final exam.

Academic honesty. Since the Coq scripts are usually quite short and appear to contain little idiosyncratic information, the temptation to copy solutions from other students seemed to be unusually high. We countered this temptation by conducting systematic cross-checking of scripts, introducing Coq quizzes, which are conducted in computer labs with internet access disabled and submitted at the end of the session, and adding a significant Coq component to the final exam, along with a traditional paper component.

4 Results of Course

It is extremely difficult to be properly scientific when analyzing didactic techniques. We can only run one small "experiment" per year, with numerous changes in curriculum, student quality, topics covered, and instructor experience. The numerous variables make us very cautious in drawing conclusions from quantitative values such as test scores. We are left with subjective opinions of students and instructors on the learning experience.

For our part, we believe that students *that were willing to put in the time required to become familiar with Coq* significantly increased their comprehension of the material. For example, we noticed a definite improvement in pen-and-paper solutions after the students had covered similar problems in Coq. We were also able to give more complex homework problems (*e.g.*, trickier induction and the semantics of Hoare logic) that we would not have been able to cover with pen-and-paper without leaving most of the class behind. We emphasize: both stronger and weaker students benefited from using Coq; the students that seemed to do the worst with the new approach were those that were unwilling to spend the (substantial) time required to become familiar with Coq.

For the students' part, we can do a fair before-and-after comparison of the student feedback for CS3234, because the two incarnations of the module before introduction of Coq were given by the first author in Semester 1 2007/2008 and Semester 1 2008/2009, and the two incarnations after the introduction of Coq were given by both authors in Semester 1 2009/2010 and Semester 1 2010/2011. At the National University of Singapore, students provide their general opinion of the module using scores ranging from 1 (worst) to 5 (best). The students also provide subjective feedback on the difficulty of the module, ranging from 1 (very easy) to 5 (very difficult). The following table includes the average feedback scores in these two categories, as well as the student enrollment and survey respondents in the listed four incarnations:

Semester	Coq Inside	Enrollment	Respondents	Opinion	Difficulty
Sem 1 2007/2008	No	37	24	3.58	3.87
Sem 1 2008/2009	No	33	20	3.55	3.95
Sem 1 2009/2010	Yes	32	17	4.17	4.00
Sem 1 2010/2011	Yes	30	19	3.84	4.05

Students can also give qualitative feedback; here is some of this feedback before Coq:

- "I would like to see more materials from a (real life) application"
- "dry module to me, cant see the link in what is taught and that i'd ever going to apply it. maybe can make it more real life applicable, and talk about how in real programming life would we use such logics. i mean we just learn the logics but dun really know where we will really be making use of it."
- "Quite good.. But everything is too theoretical .."
- "There are very complex ideas which are very difficult to explain."

Here is some of the feedback after the introduction of Coq:

- "Fantastic module. The workload is slightly heavy but that is fine. Learnt a lot."
- "Strengths: help students understand various aspects of logic and how it can be applied in computer science. Weakness: Only the surfaces of some topics. cannot appreciate their usefulness. Homeworks (paper + coq) consume a lot of time"
- "The strength of this module covers various topic on formal proving, giving me a deeper understand on the application of discrete structure that i had taken before. The lecture slides and some of the additional notes are clear and helpful. I like the idea of having Coq lab session, whereby we apply what we learn. However, some of the quiz are very challenging and i think we do need more extra practices (not included in CA marks) on the Coq besides just the homework. The workload is rather heavy and each assignment and homework is just 2%."
- "good module with many labs that can give me a good understanding of COQ"

We received an email from a student of CS5209 that nicely summarizes the benefits and challenges from Coq from the students' perspective: "I would like to thank you for the Automated Theorem Prover (Coq) you taught in CS5209 course. It makes life easy while trying to prove theorem as compared to paper part. In addition to this it saves life of student in Final exam. In the beginning for the course I hated Coq a lot, but slowly I start liking it as I understood the way tactic works and how to use them. Now it has become most favorite and interesting part of mine in this course."

5 Related Work

There has been extensive previous work in using proof assistants to teach formal methods. For example, certain logic courses at Carnegie Mellon have been using the ETPS system (a variant of the TPS proof system developed with a focus on education) since 1983 [ABB+03]. Some of the conclusions from using the ETPS system mirror our own: students have been able to prove more difficult theorems with the aid of a proof assistant, and "students show remarkable creativity in developing surprisingly awkward ways of doing things". However, while there is a considerable amount of material in the literature about the ETPS system as a piece of software, we have not found much in the way of experience reports in terms of how to integrate the system into a curriculum.

More recent work has largely focused on using a proof assistant to teach programming languages (including type theory) as opposed to introductory logic. SASyLF is an LF-based proof assistant designed specifically to enable mechanizing proofs about programming languages simple enough for the classroom [ASS08]. One primary advantage

of a more specialized tool such as SASyLF is that the surface syntax can be much closer to paper proofs. Although we found that SASyLF allows for quite elegant statement of grammars and judgments, we found the actual proof scripts to be a bit verbose and cumbersome. The disadvantage, generally speaking, of specialized educational tools is that they tend to be "broad but shallow"—that is, they trade off expressive power for ease of use; in the case of SASyLF, for example, users are restricted to second-order logic. We wanted our students to have exposure to a software system that would allow them to explore further if they wished to (as, indeed, a number did). A related thread is the development of alternative general-purpose theorem provers with, hopefully, a simpler user experience, such as Matita [ACTZ07] or ACL2 Sedan [ACL].

Pierce and others at the University of Pennsylvania use the proof assistant Coq for teaching programming language semantics [Pie09], and observe several of the same general points that we do, *e.g.*, that teaching with Coq becomes entwined with teaching Coq. We suspect that teaching multiple courses (e.g., both logic and programming languages) with the same proof assistant would yield considerable advantages since the costs of teaching Coq would be spread over two semesters instead of one. We are not aware of any attempt to teach such a sequence.

6 Conclusion

We have outlined a migration of a traditional course on logic for computer science to a format that makes extensive use of the theorem prover Coq. Our approach resulted from teaching the material three times (twice in an undergraduate and once in a graduate setting). Along the way, we have found a number of didactic techniques to be useful:

- Introduction of Aristotelian term logic prior to propositional logic so that we can introduce the basic concepts of logic and Coq more gently.
- Keeping the object- and metalogics separate at the beginning; only transitioning to direct use of Coq's `Prop` once the distinction is clear.
- Delaying formal discussion of induction until after predicate logic, and then covering it in detail once students' familiarity with Coq can provide assistance.
- Presenting a full-powered modal logic in Coq instead of attempting to precisely duplicate the experience on paper; a significant exploration of correspondence theory.
- Giving a semantics for Hoare logic so that students can prove the Hoare axioms.
- Presenting several direct applications of formal systems to computational problems: resource scheduling for propositional logic; network security analysis for predicate logic; and Hoare logic's semantics for modal logic.

Comparing the student feedback from CS3234 before and after the migration, it is clear that the introduction of Coq was well received by the students, as shown by a significant improvement of the overall student opinion of the module, at the cost of a modest increase in module difficulty. Anecdotal evidence suggests that the students appreciated the additional learning opportunities afforded by the use of Coq throughout the courses. Overall, considering the available evidence, we believe that the use of Coq in these courses has improved the students' learning of formal logic considerably. The price to pay was additional time spent on learning Coq, which we consider a worth-while investment in its own right.

The material resulting from the migration (including an extensive collection of Coq assignments, quizzes and exam questions) is available online [HH10] for the benefit of the community of academics involved in teaching logic to computer science students.

References

[ABB+03] Peter B. Andrews, Peter Bishop, Chad E. Brown, Sunil Issar, Frank Pfenning, and Hongwei Xi. ETPS: A system to help students write formal proofs, 2003.

[ACL] The ACL2 Sedan. `http://acl2s.ccs.neu.edu/acl2s/doc`.

[ACTZ07] Andrea Asperti, Claudio Sacerdoti Coen, Enrico Tassi, and Stefano Zacchiroli. User interaction with the matita proof assistant. *Journal of Automated Reasoning*, 39(2):109–139, 2007.

[ASS08] Jonathan Aldrich, Robert J. Simmons, and Key Shin. SASyLF: An educational proof assistant for language theory. In *2008 ACM SIGPLAN Workshop on Functional and Declarative Programming Education (FDPE'08)*, Victoria, BC, Canada, 2008.

[BA01] Mordechai Ben-Ari. *Mathematical Logic for Computer Science*. Springer, 2001.

[Bor06] Donald M. Borchert, editor. *Glossary of Logical Terms*. Encyclopedia of Philosophy. Macmillan, 2nd edition, 2006.

[dW01] Paulien de Wind. Modal logic in Coq. VU University Amsterdam, IR-488, `http://www.cs.vu.nl/~tcs/mt/dewind.ps.gz`, 2001.

[HH10] Martin Henz and Aquinas Hobor. Course materials for cs3234/cs5209. `http://www.comp.nus.edu.sg/~henz/cs3234`, 2010.

[HKT00] David Harel, Dexter Kozen, and Jerzy Tiuryn. *Dynamic Logic*. MIT Press, 2000.

[HR00] Michael R. A. Huth and Mark D. Ryan. *Logic in Computer Science: Modelling and reasoning about systems*. Cambridge University Press, Cambridge, England, 2000.

[NT98] George L. Nemhauser and Michael A. Trick. Scheduling a major college basketball conference. *Operations Research*, 46(1):1–8, 1998.

[OGA05] Xinming Ou, Sudhakar Govindavajhala, and Andrew W. Appel. MulVAL: A logic-based network security analyzer. In *14th USENIX Security Symposium*, 2005.

[PH91] William T. Parry and Edward A. Hacker. *Aristotelian Logic*. State University of New York Press, 1991.

[Pie09] Benjamin C. Pierce. Lambda, the ultimate TA: Using a proof assistant to teach programming language foundations, 2009.

[Zha93] Hantao Zhang. SATO: A decision procedure for propositional logic. *Association of Automated Resasoning Newsletters*, 22, 1993. updated version of November 29, 1997.

[Zha02] Hantao Zhang. Generating college conference basketball schedules by a SAT solver. In *Proceedings of the Fifth International Symposium on Theory and Applications of Satisfiability Testing*, pages 281–291, Cincinnati, Ohio, 2002.