TEACHING LOGIC USING A STATE-OF-THE-ART PROOF ASSISTANT

Maxim Hendriks, Cezary Kaliszyk, Femke van Raamsdonk, Freek Wiedijk

Abstract. This article describes the system ProofWeb developed for teaching logic to undergraduate computer science students. The system is based on the higher order proof assistant Coq, and is made available to the students through an interactive web interface. Part of this system is a large database of logic problems. This database will also hold the solutions of the students. The students do not need to install anything to be able to use the system (not even a browser plug-in), and the teachers are able to centrally track progress of the students.

The system makes the full power of Coq available to the students, but simultaneously presents the logic problems in a way that is customary in undergraduate logic courses. Both styles of presenting natural deduction proofs (Gentzen-style 'tree view' and Fitch-style 'box view') are supported. Part of the system is a parser that indicates whether the students used the automation of Coq to solve their problems or that they solved it themselves using only the inference rules of the logic. For these inference rules dedicated tactics for Coq have been developed. The system has already been used in type theory courses and logic undergraduate courses. The ProofWeb system can be tried at `http://proofweb.cs.ru.nl/`.

Key words: Logic Education, Proof Assistants, Coq, Web Interface, Natural Deduction

1 Introduction

At most European universities, part of the undergraduate computer science curriculum is an introductory course that teaches the rules of propositional and predicate logic. At the Radboud Universiteit (RU) in Nijmegen this course is taught in the first year and is called 'Beweren en Bewijzen' (Dutch for 'Stating and Proving'). At the Vrije Universiteit (VU) in Amsterdam this course is taught in the second year and is called 'Inleiding Logica' ('Introduction to Logic'). Almost all computer science curricula have similar undergraduate courses.

For learning this kind of elementary mathematical logic it is crucial to work many exercises. Those exercises can of course be done in the traditional way, using pen and paper. The student is completely on his own, and in practice it often happens that proofs that are almost-but-not-completely-right are produced. Alternatively, they can be made using some computer program, which guides the student through the development of a completely correct proof. A disadvantage of the computerized way of practising mathematical logic is that a student often will be able to finish proofs by random experimentation with the commands of the system (accidentally hitting a solution), without really having understood how the proof works. Of course, a combination of the two styles of practicing formal proofs seems to be the best option. In any case, computer assistance for learning to construct derivations in mathematical logic seems desirable. Currently the most popular program that is used for this kind of 'computer-assisted logic teaching' is a logical framework called Jape [3], developed by Surin and Bornat at the University of Oxford.

Besides exercises there is also the issue of examination. It would be good if the student has the opportunity to do at any moment a (part of the) logic exam by logging in to the system and be presented with a set of exercises from a database that have to be solved within a certain time. This may require human supervision to prevent cheating. We did not yet work on this, but just mention it as a possible interesting application of computer-assisted logic teaching.

This article describes our development, named ProofWeb. This system provides functionality much like Jape, but the two main innovations that it offers over other, similar systems are:

Received: . . . , accepted: . . . , published: . . .

- The students work on a central server that is accessed through a web interface. The proof assistant will not run on their computer, but instead will run on the server.

  A first advantage is flexibility. The web interface is light: the student will not need to install anything to be able to use it, not even a plug-in. When designing our system we tried to make it as low-threshold as possible. The student can work from any internet connection at any time.

  A second advantage is that the student does not need to worry about version problems with the software or the exercises. Since everything is on the same server, the students have at any time the right version of the software, exercises, and possibly solutions to exercises available, and moreover the teachers know at any time the current status of the work of the students.

  Thirdly, didactical advantages were thought to be serveral. The students are forced to think when solving a problem with ProofWeb, because it lacks the possibility to construct a proof 'at random' by just a few mouse clicks. Also, its back engine (see below) provides the possibility for customization and extension, making it useful in courses at every level. Thus, the system helps to let the student get used to the full capabilities of a proof assistent by providing stepping stones. For more information, see Section 6.

- The system makes use of a state-of-the-art proof assistant, namely Coq [5], and not a system designed for predicate or propositional logic only.

  Coq has been in development since 1984 at the INRIA institute in France. It is based on a type theoretical system called *the Calculus of Inductive Constructions*. It has been implemented in the OCaml programming language (see [19]), and has been used for the formal verification of many proofs, both from mathematics and from computer science. The most impressive verification using Coq is the verification of the proof of the Four Colour Theorem by Georges Gonthier [10]. Another important verification is the development of a verified C compiler by Xavier Leroy and others [20].

  The choice for a state-of-the-art proof assistant fell on Coq because both at the RU and at the VU it is already used in research and teaching.

  An advantage of using a state-of-the-art proof assistant is again flexibility. The same interface can be used (possibly adapted) for teaching more advanced courses in logic or proof assistants.

We also include in ProofWeb:

- A large collection of logic exercises. The exercises range from very easy to very difficult, and are graded for their difficulty. The exercise set is sufficiently large (presently over 200 exercises) that the student will not soon run out of practice material. More about the exercise set can be found in Section 5.
- Course notes, with a basic presentation of propositional and predicate logic, and a description of how to use the system. We choose the presentation of the proofs in the system to be the same as the presentation of the proofs in textbooks. Therefore we developed both the 'Gentzen-style' and the 'Fitch-style' natural deduction variants.

Related work

There are already numerous systems for doing logic by computer. The Association for Symbolic Logic maintains a relatively comprehensive list [7]. The list contains many systems which are quite similar to each other. Furthermore, many of the systems are even already web-based. We choose to compare our system to the logical framework Jape [3], because it seems to be the most widely used of these systems.

There are also many systems that provide centralized environments for learning mathematics that are not backed up by a proof assistant. A representative may be [21], which is an interactive platform for learning mathematical knowledge.

Many experiments have been done with teaching using proof assistants. Mizar has already been used in teaching logic since 1988 [1]. The use of Coq in the classroom is described on the Coq Wiki [4].

The distinctive features of our system are the use of a serious proof assistant, together with a *centralized* 'web application' architecture. The work of the students remains on the web server. It can be saved and loaded back in, and the profile of the student in terms of progress and solution styles is at all times available both to the student, the teacher and the system. This could, for example, enable a teacher to pinpoint a missing step in a student's understanding. The teacher could then give the student a hint or teach him something in a face-to-face meeting.

Our system has been developed for teaching logic in the natural deduction style. There also exists a school of teaching logic due to Dijkstra and Gries, called Calculational Logic (see [11]), in which reasoning is done through rewriting with equations. The Coq system is powerful enough to support this kind of reasoning as well, but we have not developed this style of logic in our system.

In this article we present both our project and the system. We start by shortly describing the architecture of the interface in Section 2. Then, in Sections 3 and 4 we will discuss the supporting infrastructure of tactics and exercises for Gentzen and Fitch style natural deduction. Section 5 talks about the collection of exercises that was constructed. Next, in Section 6, we describe the experiences we had in the project using the system in class. Finally, in Section 7 we give an outlook on future work and possible improvements to the system.

2 The architecture of the interface

The architecture of the interface to Coq used in ProofWeb is an implementation of an architecture for creating responsive web interfaces for proof assistants [14]. It combines current web development technologies with the functionality of local interfaces for proof assistants to create an interface that behaves like a local one, but is available completely with just a web browser (no Java, Flash or plug-ins are required).

To provide this functionality, it uses the *asynchronous DOM modification* technology [22] (sometimes referred to as *AJAX* or *Web Application*). This technique is a combination of three available web technologies:

- JavaScript: a scripting programming language interpreted by the web browsers,
- *Document Object Model* (*DOM* [18]): a way of referring to sub elements of a web page that allows modification of the page on the fly creating dynamic elements
- XmlHttp [23]: an API available to client side scripts, that allows requesting information from the web server without reloading the page.

The technique consists in creating a web page that captures events on the client side and processes them without reloading the page. Events that require information from the server send the data in asynchronous XmlHttp requests and modify the web page in place. Other events are processed only locally. The server keeps prover sessions for all users and the clients are presented with an interface that is completely available in a web-browser but resembles and is comparably responsive to a local interface.

Using the architecture described in [14] for teaching required the creation of groups of logins for particular courses. The students are allowed to access only their own files via the web interface, and teachers of particular courses have access to students' solutions through the admin interface. What the interface looks like can be seen in Figure 1. The proof is constructed on the left hand side, by typing and possibly by using templates from the drop-down menus at the top of the page. The top-right frame displays what is still to be proved. And the bottom-right frame can display the (partially) constructed proof tree or flag proof created by the system, as will be discussed later on.

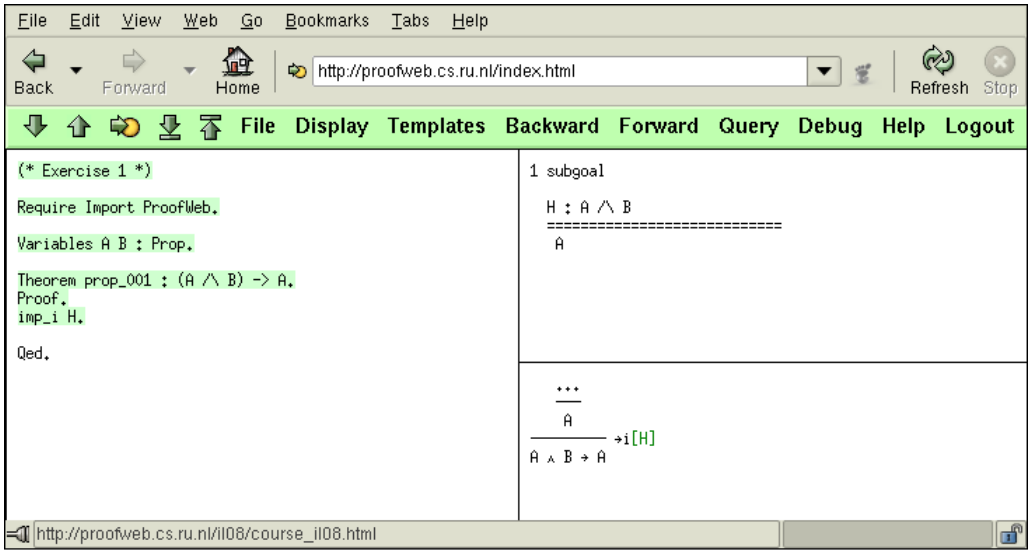

Figure 1: A propositional logic exercise in ProofWeb.

3 Gentzen style natural deduction for first-order logic

A first aim in the development of ProofWeb was to have an exact correspondence between the derivations on paper and the derivations in the system. The student should then work with a set of dedicated tactics, because the standard Coq tactics are too powerful. (For instance, one could solve the exercises in propositional logic using the tactic `tauto` instead of building the actual derivation.)

The standard way of working with Coq is with backward proofs, so we first naturally arrived at a set of backward tactics: every proposition (the current goal) is deduced from another proposition (the new goal) using a deduction rule. The display style that fits most naturally to this kind of proof is a proof tree (flag-style proofs are described in Section 4). This imposes a strict way of working. The proof trees have to be constructed from 'bottom to top'. On the one hand, this makes the construction of a deduction more difficult than on paper, because there is no possibility of building snippets of the proof in a forward way, using what is known from the hypotheses and their consequences. But on the other hand, the method forces the student to ponder the general structure of the proof before deciding by what step he will eventually end up with the current proposition. And the imposed rigidity is congenial with the aim of a logic course to encourage rigorous analytical thinking. Moreover, it becomes very clear where ingenuity comes in, such as with the disjunction elimination rule. The student is supposed to prove some proposition C. It is a creative step to find a disjunction $A \vee B$, prove this, and also prove that C follows from both A and B separately. The same goes for the introduction and elimination of negation.

As an example we present the tactic for disjunction elimination, which gives a good impression of the way additional tactics are implemented (of course, the student never sees this code and can just work with the tactic):

```
Ltac dis_e X H1 H2 :=
  match X with
  | ( _ \/ _ ) =>
   let x := fresh "H" in
   assert (x : X);
   [ idtac | elim x; clear x; [intro H1 | intro H2] ]
```

```
end
|| fail "(the argument is not a disjunction
  or the labels already exist)".
```

If the current goal is C, the tactic `dis_el (A \/ B) G H` will create the following three new goals:

1. $A \vee B$;
2. C, with the extra assumption A with label G;
3. C, with the extra assumption B with label H.

Also, the tactic gives a nice and understandable error message. All natural deduction tactics have been given a name by using three letters of the connective's name and indicating whether the tactic implements an introduction rule or an elimination rule (and if necessary, if that is a left or a right variant or whether it is the forward tactic).

Suppose for example that a student has to prove a simple theorem like `forall A : Prop, A -> A` (the simplest of tautologies, meaning that all propositions imply themselves). This goal will be shown in the top right frame. The student can then type `all_in A.` to say to ProofWeb that this can be proved by using the introduction rule for $\forall$ in the last step of the proof. The system will accept this, and say that the user may now consider `A` as a free variable, and use it to prove `A -> A`. The user can now type `imp_in H.` to say instruct Coq to reach this goal by implication introduction from the hypothesis that `A` holds, which is given the hypothesis name `H`. It now only remains to prove `A`, but since this is now in the list of hypotheses we may make, the user can finish the proof by notifying Coq that this is true by assumption: `ass H.`. A step in this proof can be seen in Figure 2.

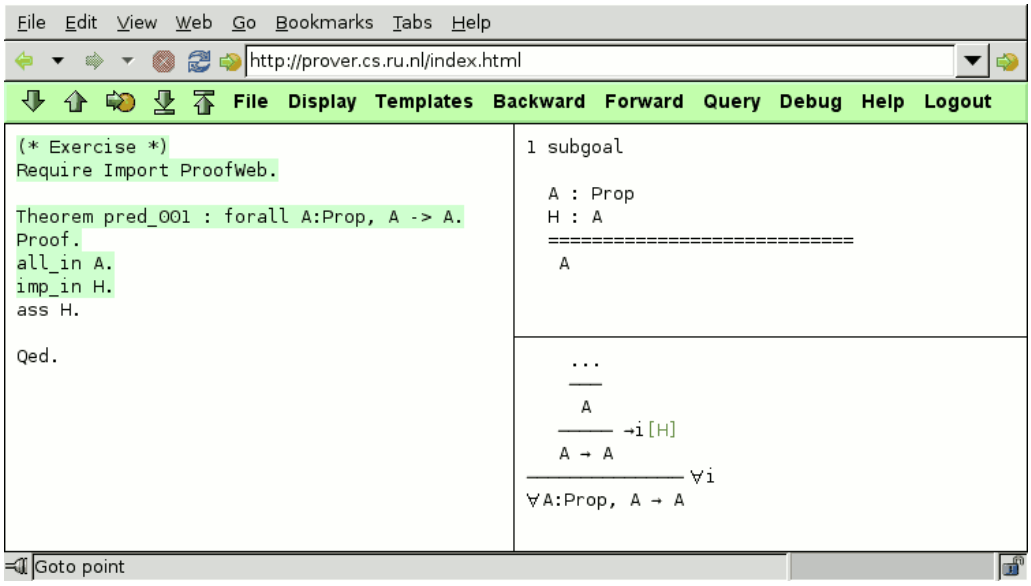

Figure 2: A step in the proof of $A \rightarrow A$.

A second aim is a visual presentation of proofs as proof trees (like in Jape and similar programs). This meant requesting the proof information from Coq and converting it to a graphic format. Coq internally keeps a proof state. This proof state is a recursive OCaml structure, that holds a goal, a rule which allows to obtain this goal from the subgoals, and the subgoals themselves. It is not just a tree structure, since a rule can be a compound rule that contains another proof state. Tactics and tacticals modify the

proof state. Coq includes commands that allow inspecting the proof state. The command `Show` with a number as argument allows the user to see a goal that is not currently being worked on, `Show Tree` shows the succession of conclusions, hypotheses and tactics used to obtain the current goal and `Show Proof` displays the CIC term (possibly with holes).

The output of these commands was not sufficient to build a natural deduction tree for the proof. We added a new command `Dump Tree` to Coq that allows exporting the whole proof state in an XML format. An example of the output of the `Dump Tree` command for a very simple Coq proof:

```
<tree><goal><concl type="A -> A"/></goal>
  <cmpdrule><tactic cmd="intro x"/>
    <tree><goal><concl type="A -> A"/></goal>
      <cmpdrule><tactic cmd="intro x"/>
        <tree><goal><concl type="A -> A"/></goal>
          <rule text="intro x"/>
          <tree><goal><concl type="A"/><hyp id="x" type="A"/>
        </goal></tree></tree>
      </cmpdrule>
      <tree><goal><concl type="A"/><hyp id="x" type="A"/>
    </goal></tree></tree>
  </cmpdrule><tree><goal><concl type="A"/><hyp id="x" type="A"/>
</goal></tree></tree>
```

ProofWeb is able to parse the XML trees dumped by Coq and generate natural deduction diagrams (see Figure 3). The diagrams are rendered as HTML tables with Unicode characters in a fixed width font, so that special signs can be shown, but they can still be transferred over the network very efficiently. Those diagrams may be requested by the user's browser in special query requests. The diagrams are displayed in a separate frame in the interface along with the usual Coq proof state. If the user switches on the display of the diagrams, the client side requests them when no text is being processed.

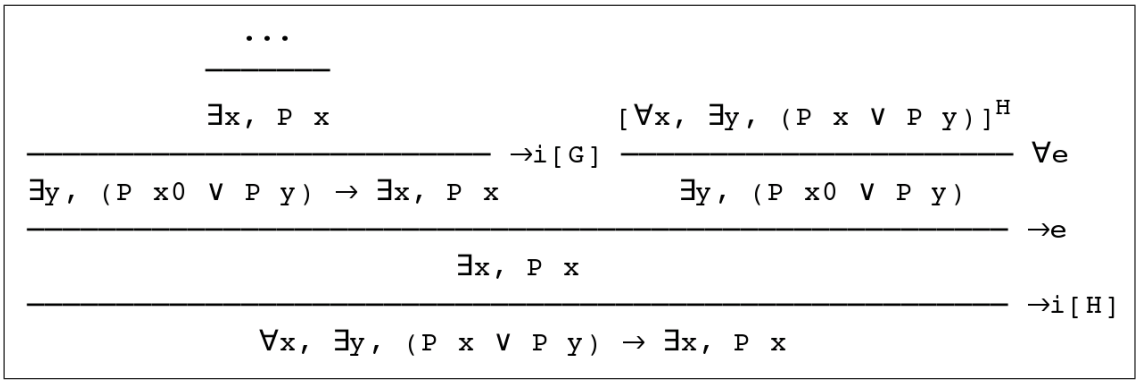

Figure 3: A natural deduction tree as seen on the web page.

Such a diagram is the form in which natural deduction in Gentzen style has traditionally been done on paper. It offers an overview of the line of argument that is more amenable to direct understanding than the Coq script, and will be familiar to the student and other logicians from doing proofs on paper.

4 Fitch style natural deduction for first-order logic

The ProofWeb system also has the possibility to use the system for so-called *Fitch-style* natural deduction proofs. This style of proof was initially developed by Stanisław Jaśkowski in 1934 [13] and

perfected by Frederic Brenton Fitch in 1952 [8]. Fitch-style proofs have the graphical advantage over Gentzen-style proofs [9] of being linear (as opposed to having a branching tree structure), which makes them more convenient to display for large proofs, like the ones constructed by the students for final assignments.

Proofs have a structure of nested boxes, which structure a sequential list of proof steps. Another name for this kind of proofs is 'flag-style proofs' because often the assumptions of a subproof are written in the shape of 'flags'.

The system implements a translation of Gentzen style proofs to a graphical display of the proof in Fitch-style natural deduction, as described in [16]. An example of a Fitch-style deduction rendered by the system is presented in Fig. 4.

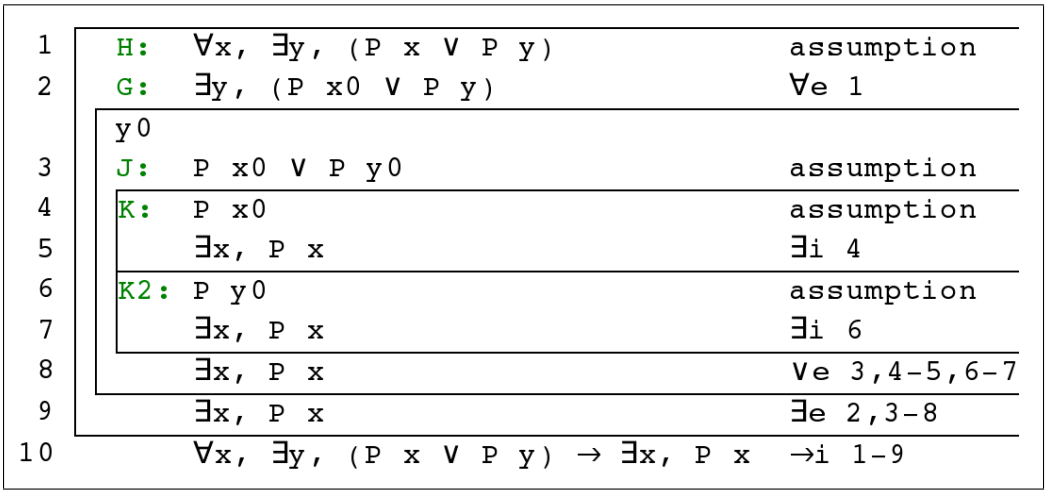

Figure 4: A Fitch-style deduction rendered by the system.

Along with this, a set of special forward tactics has been implemented. Since Coq works goal-oriented, backward tactics are its natural sort, and the mimicry isn't perfect, but it does help to give an introduction to this other system of proving. The forward tactics are denoted with the `f_` prefix

```
Theorem pred_076 : all x, exi y, (P(x) \/ P(y)) -> exi x, P(x).
Proof.
  imp_i H.
  insert G (exi y, (P(x0) \/ P(y))).
  f_all_e H.
  exi_e (exi y, (P(x0) \/ P(y))) y0 J.
  ass G.
  dis_e (P(x0) \/ P(y0)) K K2.
  ass J.
  f_exi_i K.
  f_exi_i K2.
Qed.
```

Now we need a little explanation of this proof, e.g. to help the reader understand some specific step in it.

5 The exercise set

As a part of the project a set of over 200 exercises was developed. If a student logs in via the web interface as a participant to a specific course, he sees the list of exercises for the course (see Figure 5). For each exercise, the system shows its file name, an indication of its difficulty, the current status of the exercise, and a button for resetting the exercise to its initial state.

The four possibilities for the status of an exercise are:

Not touched (in grey)
Incomplete (in red)
Correct (in orange)
Solved (in green)

The colors are meant to resemble the colors of a traffic light.

- The status Not touched means that an exercise has not been opened, or has been opened but has not been saved.
- The status Incomplete means that the file is incomplete or wrong. If one want to know why ProofWeb thinks there is an error in the solution, one can click the 'why?' link next to the status. The system shows a new window that shows the error message of Coq.
- The status Correct means that the file is a correct Coq-file, but that it is not accepted as a solution for the exercise in the course. This happens for instance if more automation is used than intended for the course, for instance: by proving a propositional formula with the tactic `tauto` instead of using the tactics corresponding exactly to the logical rules used in the course. It also occurs when the file is empty. If the status is Correct one can click on the 'why?' link to find out which steps are not allowed in the course.

  This is a feature of ProofWeb that is meant as a service for the teacher, but of course in addition manual verification may be required, for instance if the exercise is to give the definition of a certain object in type theory.
- The status Solved means that the file is correct Coq code and is accepted as a solution in the course.

The verification tool lexes the original task and the student's solution in parallel. The original solution includes placeholders that are valid Coq comments. An example task file on the server looks like:

```
(* Exercise 118 *)

Require Import ProofWeb.

Variables A B C : Prop.

Theorem prop_118 :
  ~~(A \/ B) -> (~~A -> ~~C) -> (~~B -> ~~C) -> ~~C.
(*! prop_proof *)
Qed.
```

The placeholders mark places that need to contain a valid Coq term or a valid proof. There are a number of proof placeholders, which determine the set of allowed tactics. For proofs and terms of given types the automatic verification is enough. However, there are tasks where students are required to give a

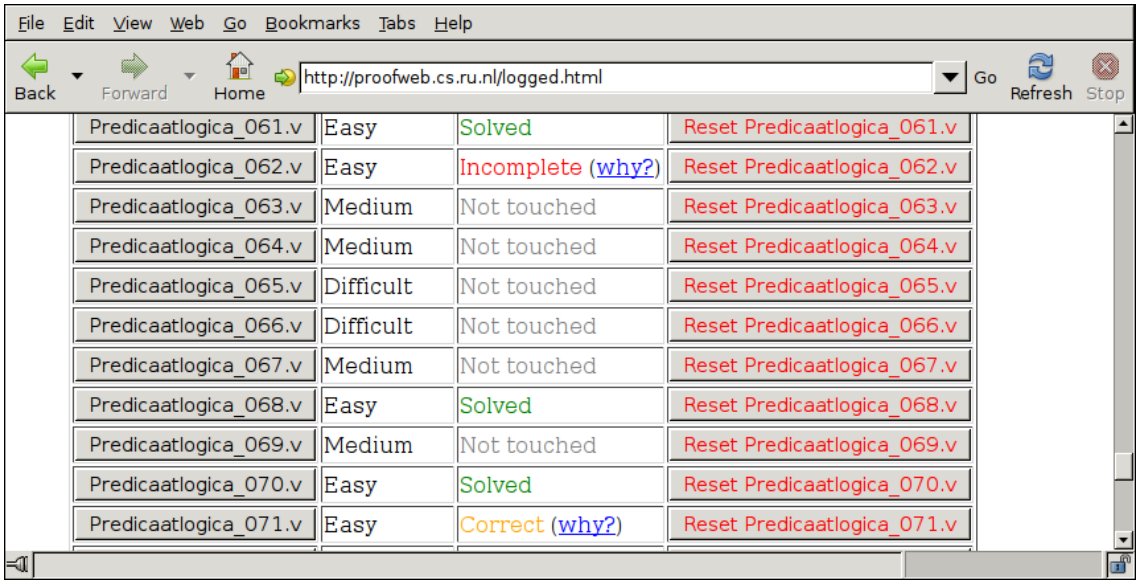

Figure 5: Tasks assigned to students and their status.

definition of a particular object in type theory. For this kind of tasks manual verification by a teaching assistant of a course is required.

There are 5 different sets of tactics allowed in different courses:

- A set of regular Coq tactics for Type theory courses and Coq courses.
- A set of basic logical tactics for classical logic courses. This includes `rewrite` for dealing with equality.
- A set of logical tactics that allow only constructive proofs.
- Forward and backward tactics for dealing with propositional logic.
- Tactics for the additional rules of predicate logic.

The exact names of tactics allows in those settings are provided in [17]. The semantics of the tactics are provided in the the Coq reference manual [5] and in the ProofWeb manual [15]).

6 Teaching experiences during the project

In the beginning of the project, ProofWeb was developed as a web-interface for using Coq on a central server. As such, the system had already been used for three master courses on type theory using Coq. These courses were opportunities to test the interface on more mature students first. During these courses there was no support for visualizing proofs yet. Instead the students had to do their proofs using the customary Coq proof style, which consists of building a tactic script using the standard Coq tactics. This was not problematic, since one of the aims of those courses was to learn Coq.

One of these courses was 'Type Theory', given in spring 2007 at the RU, which quickly went through predicate logic using, concurrently introducing lambda calculus and type systems. As it turned out that initially there were only very few students who wanted to follow this course, it was decided that there

would be no lectures, and that the students just would be given the course notes together with access to the server. They would work on their own, with opportunity to call for help if needed. It turns out that this worked unexpectedly well. The students studied the lecture notes and did the exercises of the course using the system. Even without much pressure on them in the form of requiring them to meet deadlines, they managed to keep on schedule reasonably well. In fact, the only thing that confused them (after which a lecture was organized to make things clear) was a part of the course that did not correspond to Coq work (about derivations in Pure Type Systems).

Then in 2007, ProofWeb was used in two different undergraduate logic courses, the primary target audience. In both courses the students used the special tactics, the display, and the database with exercises to practice natural deduction proofs.

1. In spring 2007: the course 'Beweren en Bewijzen' at the RU [2], a first-year computer science course in logic using Gentzen style 'tree' proofs.
2. In fall 2007: the course 'Inleiding Logica' at the VU [12], a computer science undergraduate course in logic, with natural deduction in Fitch style (see [16]).

For the former course, the first author worked in close contact with the teacher. He gave two short instructions (20 minutes each) explaining how to use the tactics to do proofs, first for propositional logic, then for predicate logic with equality. Each time, this was after the students had already familiarized themselves with the corresponding rules from the course notes, but were not very experienced in using them yet. The students had no noticable difficulty in picking up on the tactics, and the amount of exercises solved (they were not obligatory) showed that enthusiastic students could do a lot.

Later on in the course, the teacher requested a custom extension of the system with some special tactics for (and exercises about) temporal aspects of logic, but done using the predicates $<$, $\leq$, $>$ and $\geq$ and Coq's innate ability to work with numbers. Of course this would have been impossible with a program specifically made for standard first-order logic only.

A questionnaire given to the students after the course gave the following results.

- The students found working with Coq very interesting, one of the better aspects of the course, and felt it should be kept in.
- They said learning to work with the web interface was not very difficult for them, but sometimes mildly annoying due to small bugs that were still in there. They liked the fact that the system worked via the web.
- They did more typing of commands than using the predefined templates from the menu. After having gained a bit of experience, this is faster, a phenomenon also seen with keyboard shortcuts in other programs.
- They found that doing proofs with the system greatly increased their understanding of logic. (Of course, this statement does not compare with anything else. We did not set up a test group to compare marks on the exam.)
- Two of the few students that also tried out Jape – which we mentioned as a good alternative to ProofWeb that they could install on their own computers – preferred Jape above ProofWeb, but gave as motivation that ''Jape makes it so easy to just click together a proof'', which was exactly something we wanted to avoid.

After these two courses, in which the system was developed in full-fledged form, ProofWeb has been used in another 17 courses of varying type in the Netherlands, UK and Brazil. Until now there have been more than 670 active student accounts and 19 teacher accounts. All in all our experience is that the system ProofWeb seems to work very well in teaching.

7 Outlook

We presented the system ProofWeb for teaching logic to undergraduate computer science students. Our experiments show that the students find backtracking and changing your partial proof is much easier with this system and that the teachers can much easier keep track of how the students are doing. The system is accompanied with a final version of the course notes [15].

Some of the issues that can be worked on are the following.

- The teacher web interface allows to manage students logins and inspect the work of the students. Managing the set of exercises is at the moment only possible by logging on to the server through an ssh connection, and then listing and editing files manually. Clearly, a proper web interface for this is necessary.

- The deduction trees are currently rendered as text or HTML in IFrames, and can be optionally opened in a separate browser window to allow easy printing as PostScript or PDF. However students might want to use the trees in their papers, and for that a dedicated TEX or image rendering of the trees could be implemented.

- The interface uses some web technologies that are not implemented in the same way in all browsers. It includes a small layer that is supposed to abstract over incompatible functionalities. Currently this works well with Gecko based browsers (like Mozilla, Firefox, Galeon, Epiphany and Netscape), Webkit based browsers (like Safari and Konqueror), and the Opera browser. Also, some effort has been made to make the system work reasonably well with common versions of Internet Explorer. However, it needs further attention.

- Our server is currently available to everyone who wants to experiment with our system, but a good guide that explains how to install your own ProofWeb server would be useful.

- The system keeps a log of each interaction of each student session. Using these logs, it is possible to develop software for 'replaying' student's sessions. We are currently discussing whether it is useful to develop such an extension of the system.

- The system was designed to be used in standard university courses. It might be useful to create a more complete online environment that would include introductory explanations and adaptive user profiles, therefore allowing students to learn logic without teacher interaction. As mentioned above, experiments in the classroom show this might be feasible.

It is possible to integrate ProofWeb with a system that supports the development of more serious proofs with the Coq system. One of the other projects being pursued is the creation of a so-called 'MathWiki' [6]. Here, traditional wiki technology is integrated with the same proof assistant front end that our system is based on. Combining it with an educational environment might be a long-term goal.

References

[1] M.R. Artalejo. Computerised logic teaching with Mizar. *Computerised Logic Teaching Bulletin*, 1(1), 1988. Scotland.

[2] Beweren en Bewijzen. `http://www.cs.ru.nl/~wupper/B&B/index.html`.

[3] R. Bornat and B. Sufrin. Jape's quiet interface. In N. Merriam, editor, *User Interfaces for Theorem Provers (UITP '96)*, Technical Report, pages 25−−34. University of York, 1996.

[4] The Cocorico Coq wiki. Coq in the classroom, 2010. `http://coq.inria.fr/cocorico/CoqInTheClassroom`.

[5] The Coq Development Team. *The Coq Proof Assistant Reference Manual, version 8.2*. LogiCal project, 2008. Distributed electronically at: `http://coq.inria.fr/doc-eng.html`.

[6] P. Corbineau and C. Kaliszyk. Cooperative repositories for formal proofs. In M. Kauers, M. Kerber, R. Miner, and W. Windsteiger, editors, *Calculemus/MKM*, volume 4573 of *Lecture Notes in Computer Science*, pages 221−−234. Springer, 2007.

[7] H. van Ditmarsch. Logic courseware, 2010. Association for Symbolic Logic. `http://www.ucalgary.ca/aslcle/logic-courseware/`.

[8] F.B. Fitch. *Symbolic Logic: an Introduction.* New York: Ronald Press Company, 1952.

[9] G. Gentzen. Untersuchungen über das logische Schliessen. In M.E. Szabo, editor, *Collected Papers of Gerhard Gentzen*. North-Holland Publishing Company, 1969.

[10] G. Gonthier. A computer-checked proof of the Four Colour Theorem, 2006. `http://research.microsoft.com/~gonthier/4colproof.pdf`.

[11] David Gries and Fred B. Schneider. *A logical approach to discrete math*. Springer-Verlag, New York, USA, 1993.

[12] Inleiding Logica. `http://www.cs.vu.nl/~tcs/il/`.

[13] S. Jaśkowski. On the rules of suppositional formal logic. In Storrs McCall, editor, *Polish Logic 1920-1939*, pages 232−−258. Clarendon Press, Oxford, 1967.

[14] C. Kaliszyk. Web interfaces for proof assistants. In S. Autexier and C. Benzmüller, editors, *Proceedings of the FLoC Workshop on User Interfaces for Theorem Provers (UITP'06), Seattle*, volume 174[2] of *Electr. Notes Theor. Comput. Sci.*, pages 49--61, 2007.

[15] C. Kaliszyk, F. van Raamsdonk, F. Wiedijk, H. Wupper, M. Hendriks, and R. de Vrijer. Deduction using the ProofWeb system. Technical Report ICIS−R08016, Radboud University Nijmegen, September 2008.

[16] C. Kaliszyk and F. Wiedijk. Merging procedural and declarative proof. In S. Berardi, F. Damiani, and U. de'Liguoro, editors, *TYPES*, volume 5497 of *Lecture Notes in Computer Science*, pages 203--219. Springer, 2008.

[17] Cezary Kaliszyk. *Correctness and Availability. Building Computer Algebra on top of Proof Assistants and making Proof Assistants available over the Web.* PhD thesis, Radboud University Nijmegen, 2009.

[18] A. Le Hors, P. Le Hégaret, L. Wood, G. Nicol, J. Robie, M. Champion, and S. Byrne. Document Object Model (DOM) Level 3 Core Specification, Version 1. *W3C Recommendation*, 2004.

[19] X. Leroy. The OCaml programming language, 1998. `http://caml.inria.fr/`.

[20] X. Leroy. Formal certification of a compiler back-end or: programming a compiler with a proof assistant. In *POPL '06: Conference record of the 33rd ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 42−−54. ACM Press, 2006.

[21] E. Melis, J. Büdenbender, G. Goguadze, P. Libbrecht, and C. Ullrich. Knowledge representation and management in activemath. *Ann. Math. Artif. Intell.*, 38(1-3):47--64, 2003.

[22] L. Dailey Paulson. Building Rich Web Applications with Ajax. *Computer*, 38(10):14−−17, 2005.

[23] W3C Web APIs Working Group. The XMLHttpRequest Object. Technical report, W3C, 2008. `http://www.w3.org/TR/XMLHttpRequest/`.

Authors

Maxim Hendriks, Technische Universiteit Eindhoven, The Netherlands, `m.hendriks@tue.nl`
Cezary Kaliszyk, Technische Universität München, Germany, `kaliszyk@in.tum.de`
Femke van Raamsdonk, Vrije Universiteit Amsterdam, The Netherlands, `femke@cs.vu.nl`
Freek Wiedijk, Radboud Universiteit Nijmegen, The Netherlands, `freek@cs.ru.nl`

Acknowledgement

This research was funded by SURF project *'Web-deductie voor het onderwijs in formeel denken'*.