Teaching logic using a state-of-the-art proof assistant

Cezary Kaliszyk Freek Wiedijk

Radboud Universiteit Nijmegen, The Netherlands

Maxim Hendriks Femke van Raamsdonk

Vrije Universiteit Amsterdam, The Netherlands

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, AJAX, Natural Deduction, Gentzen, Fitch

1 Introduction

At every university, part of the undergraduate computer science curriculum is an introductory course that teaches the rules of propositional and predicate

[1] Email: {`cek,freek`}`@cs.ru.nl` {`mhendri,femke`}`@few.vu.nl`

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

This paper is electronically published in
Electronic Notes in Theoretical Computer Science
URL: `www.elsevier.nl/locate/entcs`

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 practicing 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. So computer assistance for learning to construct derivations in mathematical logic is desirable. Currently the most popular program that is used for this kind of 'computer-assisted logic teaching' is a system called Jape [2], developed at the university of Oxford.

This paper describes our development, named ProofWeb. This system is much like Jape (it might be considered to be an 'improved Jape-clone'). The two main innovations that our system offers over other similar systems are:

- The system makes the students work on a centralized server that has to be accessed through a web interface. The proof assistant that the students use will not run on their computer, but instead will run on the server.

  A first advantage is flexibility. The web interface is extremely 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 and non-threatening 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 centralized 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.

- The system makes use of a state-of-the-art proof assistant, namely Coq [3], and not of a 'toy' system.

  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 Objective Caml, 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 [5]. Another important verification is the development of a verified C compiler by Xavier Leroy and others [8].

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 concerning the use of the proof assistant.

The system ProofWeb comes equipped with two more products.

- A large collection of logic exercises. The exercises range from very easy to very difficult, and will be graded for their difficulty. The exercise set is sufficiently large (over 200 exercises) that the student will not soon run out of practice material. More about the exercise set can be found in Section 6.
- Course notes, with a basic presentation of propositional and predicate logic, and a description of how to use the system ProofWeb. We want the presentation of the proofs in the system to be identical to the presentation of the proofs in the textbook. Therefore we develop both the 'Gentzen-style' and the 'Fitch-style' natural deduction variants.

There are already numerous systems for doing logic by computer, of which Jape is the best known. A relatively comprehensive list is maintained by Hans van Ditmarsch [9]. Of course many of these system are quite similar to our system (as well as to each other). For instance, quite a number of these systems are already web-based.

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, can be saved and loaded back in, and the progress of the student is at all times available both to the student, the teacher and the system (i.e., the system has at all times an accurate 'user model' of the abilities of the student).

Our system has been developed for teaching logic in the natural deduction style. There exists a school of teaching logic due to Dijkstra and Gries, called 'Calculational Logic', 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 the rest of the paper we present our experiences so far (Section 2), next we present the architecture of the interface (Section 3). Sections 4 and 5 are concerned with the supporting infrastructure of tactics and exercises, and Section 6 with the presentation of the collection of exercises. Finally, in Section 7 we give an outlook on future work.

2 Experience so far

In the beginning of the project, ProofWeb was developed as a web-interface for using Coq on a centralized server. In this status, the system was already used in three master courses on type theory using Coq:

(i) In fall 2006: the course ‘Logical Verification’ at the VU [10], taught by Femke van Raamsdonk. The course also recapitulates some undergraduate logic, using natural deduction in Gentzen style.

(ii) In spring 2007: the course ‘Type Theory’ at the RU, taught by Freek Wiedijk and Milad Niqui. This course corresponds to the Logical Verification course at the VU.

(iii) In spring 2007: the course ‘Type Theory and Proof Assistants’ in the ‘Master Class Logic 2006-2007’, taught by Herman Geuvers and Bas Spitters. This course is similar to the previous ones, but is aimed at master’s students from all over the Netherlands.

These courses were opportunities to test the interface of ProofWeb on more mature students. The efficiency of the server turned out not to be a problem. At peak times around sixty students use about 2Gb memory and a fraction of a CPU. This might be thanks to the fact that the students are not using tactics that involve automation.

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

(i) In spring 2007: the course ‘Beweren en Bewijzen’ at the RU [1] taught by Hanno Wupper and Erik Barendsen. This is a computer science undergraduate course in logic using Gentzen style ‘tree’ proofs. See Section 4 for a more elaborate discussion about ProofWeb and Gentzen style natural deduction.

(ii) In fall 2007: the course ‘Inleiding Logica’ at the VU [6] taught by Roel de Vrijer. This is a computer science undergraduate course in logic using Fitch style ‘flag’ proofs. See Section 5 for a more elaborate discussion about ProofWeb and Fitch style natural deduction.

3 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 [7]. It combines the 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 it uses the *asynchronous DOM modification* technology (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*) – a way of referring to sub elements of a web page that allows modification of the page on the fly creating dynamic elements and XmlHttp – 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.

The architecture described in [7] was designed as a publicly available web service. Using it 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.

An example of the use of the interface can be seen in Fig. 1.

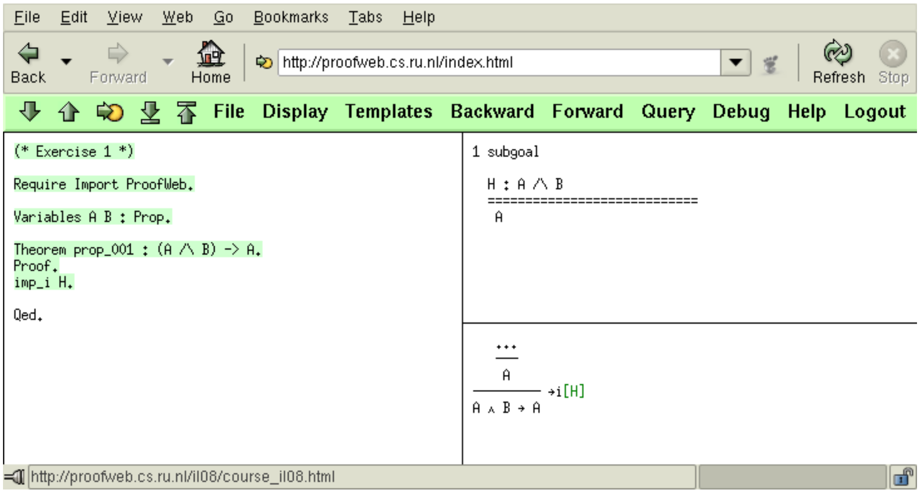

Fig. 1. A propositional logic exercise in ProofWeb.

4 Gentzen style natural deduction for first-order logic

A first aim in the development of ProofWeb was a 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, then one could solve the exercises in propositional logic using the tactic `tauto` instead of building the actual derivation.) We naturally arrived at a set of backward working tactics: every proposition (the current goal) is deduced from another proposition (the new goal) using a deduction rule. This imposes a relatively 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.

As an example we present the tactic for disjunction elimination, which gives a good impression of the way additional tactics are implemented:

```
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:

(i) $A \vee B$;

(ii) C, with the extra assumption A with name (or proof, if viewed constructively) G;

(iii) C, with the extra assumption B with name H.

An example proof with our set of tactics (in all exercises the domain is non-empty):

```
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.
```

4.1 Visualization

A second aim is a visual presentation of proofs as in Jape. 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. The Coq commands that allow inspecting the proof state (`Show`, `Show Tree` and `Show Proof`) were 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 (Fig. 2). The user's browser may request diagrams (when no text is being processed) and displays them in a separate frame in the interface along with the usual Coq proof state.

```
                ...
              ------
              ∃x, P x                                [∀x, ∃y, (P x ∨ P y)]^H
--------------------------------------- →i[G] ---------------------------------- ∀e
∃y, (P x0 ∨ P y) → ∃x, P x                         ∃y, (P x0 ∨ P y)
------------------------------------------------------------------------------------ →e
                                   ∃x, P x
------------------------------------------------------------------------------------ →i[H]
                 ∀x, ∃y, (P x ∨ P y) → ∃x, P x
```

Fig. 2. A natural deduction tree as seen on the web page.

5 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.[3] Fitch-style proofs have the graphical advantage over Gentzen-style proofs of being linear (as opposed to having a branching tree structure), which makes them more convenient to display for

[3] This style of proof was initially developed by Stanisław Jaśkowski in 1934 and perfected by Frederic Brenton Fitch in 1952.

large proofs, like the ones constructed by the students for final assignments. Another name for these kind of proofs is *flag-style proofs*, because the assumptions of a subproof are often written in the shape of 'flags'.

```
 1  | H:   ∀x, ∃y, (P x ∨ P y)               assumption
 2  | G:   ∃y, (P x0 ∨ P y)                  ∀e 1
    | | y0
 3  | | J:   P x0 ∨ P y0                     assumption
 4  | | | K:   P x0                          assumption
 5  | | |      ∃x, P x                       ∃i 4
 6  | | | K2:  P y0                          assumption
 7  | | |      ∃x, P x                       ∃i 6
 8  | |        ∃x, P x                       ∨e 3,4-5,6-7
 9  |          ∃x, P x                       ∃e 2,3-8
10             ∀x, ∃y, (P x ∨ P y) → ∃x, P x  →i 1-9
```

Fig. 3. A Fitch-style deduction rendered by the system.

6 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. It gives for each exercise the name of file that holds the exercise, an indication of the 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 you want to know why ProofWeb thinks there is an error in your solution, you can click the why? link next to the status. You will get a new window that shows the error message of Coq.
- The status Correct means that the file is a correct Coq-file. However, the file is not accepted as a solution for the exercise in the course. This happens for instance if more automation (present in Coq) 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 happens if the file is empty. If the status is Correct

you can click on the why? link to find out what you did that ProofWeb thinks is not allowed in the course.

This is a feature of ProofWeb 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 and moreover is accepted as a solution in the course.

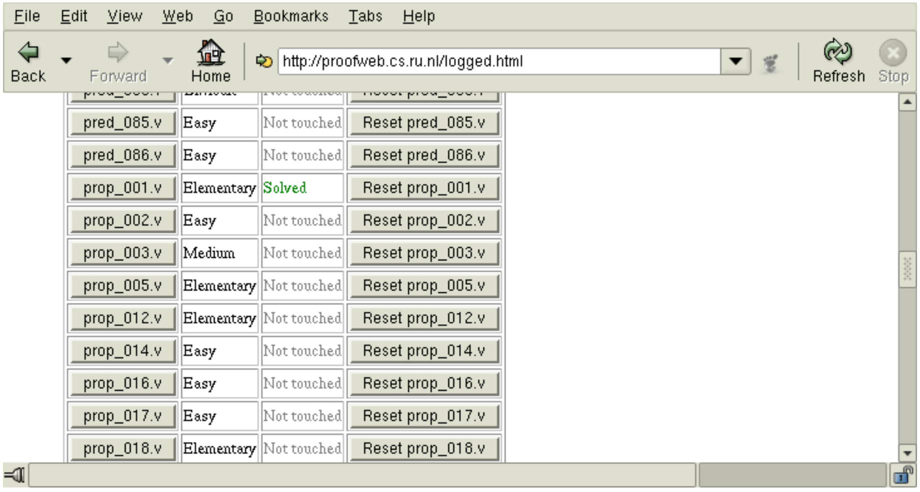

Fig. 4. Tasks assigned to students and their status.

7 Future work

Some of issues that currently are being worked on are the following.

- A first version of the course notes is available via the web-page of the system.
- 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 may need to use the trees in texts, 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.
- The system a log of each interaction of each student session is already stored on the server. Using these logs, it is possible to develop software for

'replaying' student sessions. We are currently discussing whether it is useful to develop such an extension of the system.

- The system was designed in a way 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.

If the development of ProofWeb is finished, a possibility is to integrate it with a system that supports the development of more serious proofs with the Coq system. One of the other projects that currently is being pursued is the creation of a so-called 'math wiki' [4]. Here, traditional wiki technology is integrated with the same proof assistant front end that our system is based on.

References

[1] *Beweren en Bewijzen.*
URL http://www.cs.ru.nl/~wupper/B&B/index.html

[2] Bornat, R. and B. Sufrin, *Jape's quiet interface*, in: N. Merriam, editor, *User Interfaces for Theorem Provers (UITP '96)*, Technical Report (1996), pp. 25–34.

[3] Coq Development Team, "The Coq Proof Assistant Reference Manual Version 8.1," INRIA-Rocquencourt (2006).

[4] Corbineau, P. and C. Kaliszyk, *Cooperative repositories for formal proofs*, in: M. Kauers, M. Kerber, R. Miner and W. Windsteiger, editors, *Calculemus/MKM*, Lecture Notes in Computer Science **4573** (2007), pp. 221–234.

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

[6] *Inleiding Logica.*
URL http://www.cs.vu.nl/~tcs/il/

[7] Kaliszyk, C., *Web interfaces for proof assistants*, Electr. Notes Theor. Comput. Sci. **174** (2007), pp. 49–61.

[8] Leroy, X., *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* (2006), pp. 42–54.

[9] *Logic courseware.*
URL http://www.cs.otago.ac.nz/staffpriv/hans/

[10] *Logical Verification.*
URL http://www.cs.vu.nl/~tcs/lv/