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 that 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://prover.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 [1], 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 [2], 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 has been the development of a verified C compiler by Xavier Leroy and others [7].

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 [8]. 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).

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

The system ProofWeb is used in the following courses:

(i) In fall 2006: the course 'Logical Verification' at the VU, taught by Femke van Raamsdonk. This is a computer science master's course about the

type theory of the Coq system. The course is meant for more mature students but also recapitulates some undergraduate logic. It is therefore suitable for testing a first version of ProofWeb as an alternative to ProofGeneral. Natural deduction is taught in Gentzen style, that is, proofs have a tree-like structure, and grow upward from the conclusion of the proof. In the following year proof trees are used by some students of the course.

(ii) In spring 2007: the course 'Type Theory' at the RU, taught by Freek Wiedijk and Milad Niqui. This course is also a master's level course about the type theory of the Coq system, and 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 one, but is not exclusively aimed at students of the RU but at master's students from all over the Netherlands.

(iv) In spring 2007: the course 'Beweren en Bewijzen' at the RU, taught by Hanno Wupper and Erik Barendsen. This is a computer science undergraduate course in logic, where students use special tactics, the display and the database to do natural deduction in Gentzen style.

(v) In fall 2007: the course 'Inleiding Logica' at the VU, taught by Roel de Vrijer. This is a computer science undergraduate course in logic, with natural deduction in Fitch style (cf. Section 5), that is, 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'.

One of the assignments in the course 'Logical Verification' at the VU involves program extraction. Of course we did not allow running the extracted programs on the server, and therefore a mechanism allowing the students to obtain the extracted program was implemented.

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.

All in all our experience so far is that the system ProofWeb seems to work well in teaching. The students understand the system well and appear to like working on the problems.

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 [6]. 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 plugins 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 [6] 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 in the 'Logical Verification' course can be seen in Fig. 1.

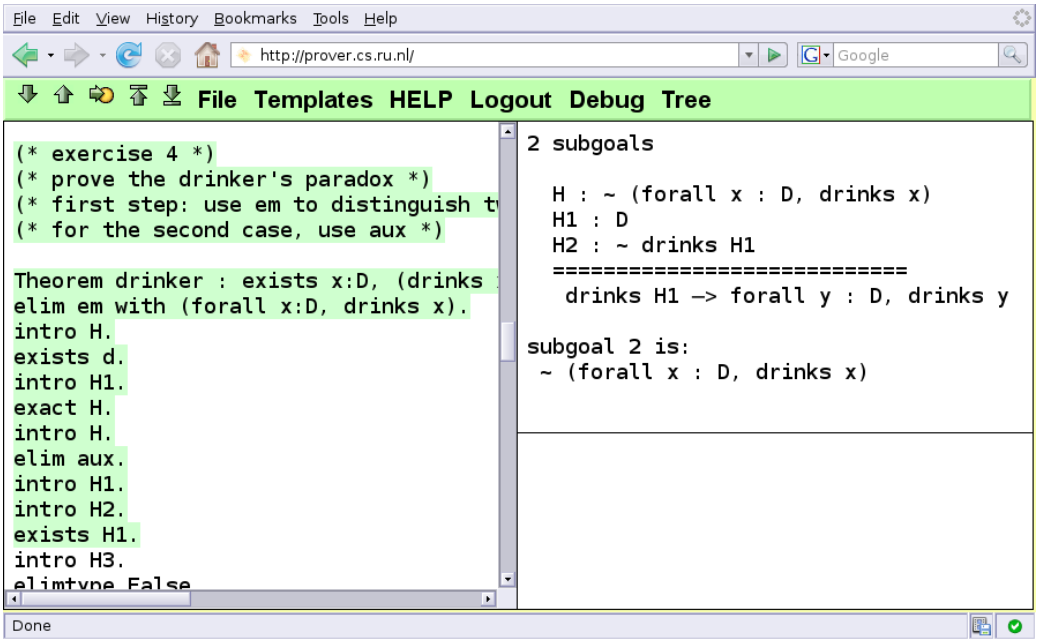

Fig. 1. ProofWeb in the 'Logical Verification' course.

4 Gentzen style natural deduction for first-order logic

A first aim is to enable students of logic courses to construct derivations that correspond exactly to the derivations in the presentation of natural deduction that they use. The traditional Coq tactics are too powerful, so for this reason we had to write tactics that match the traditional logic rules. 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. The display style that fits most naturally to this kind of proof is a proof tree (for flag-style proofs see Section 5).

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. 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:

```
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:

```
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.

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

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

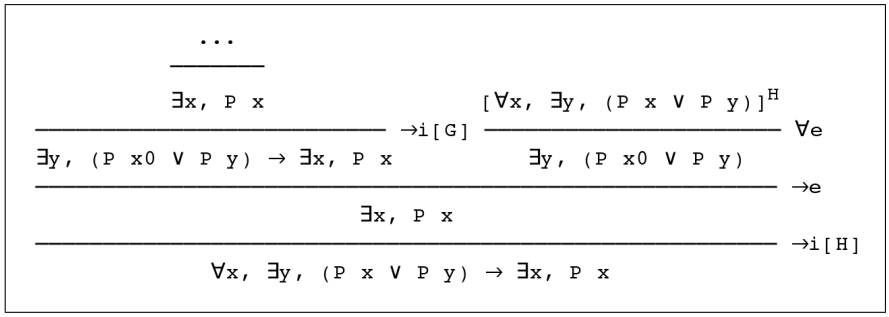

Fig. 2. A natural deduction tree as seen on the webpage.

branching tree structure), which makes them more convenient to display for 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'.

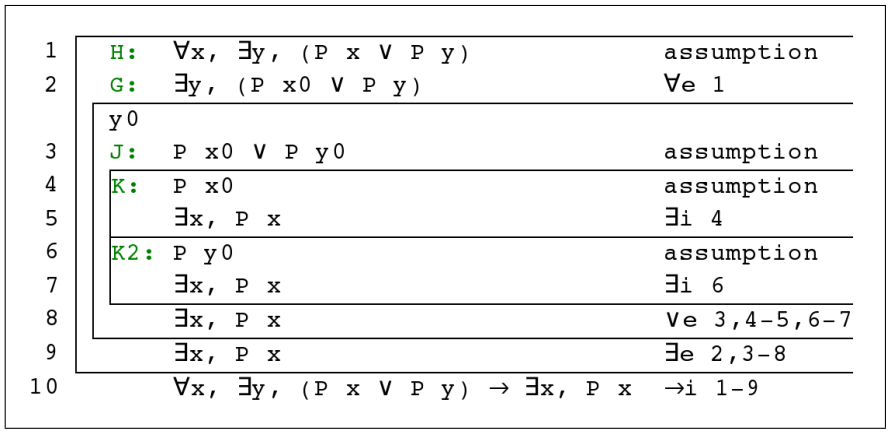

Fig. 3. A Fitch-style deduction rendered by the system.

6 The exercise set

Part of the project was the development of a set of exercises for the students. For a particular course, a number of exercises are assigned to the students. It is desirable that ProofWeb can be used as a complete course environment. When a student logs in via the web interface as participant to a specific course, he is able to see the list of all the assigned tasks (Fig. 4). Every task has a certain status. The status can be one of the following:

- Not touched — When a particular exercise has not been opened, or has been opened but has not been saved.
- Doesn't compile — When the file has been edited and saved, but is not a correct Coq file. It can be because of real errors or because proofs are missing.
- Correct — The students are supposed to modify the given file only in designated places and to use only a set of allowed tactics. If the student uses

a non-allowed too powerful tactic or just removes a task from the file it is marked as 'correct' but not as 'solved'.

- Solved — Passed the verification by our tool.

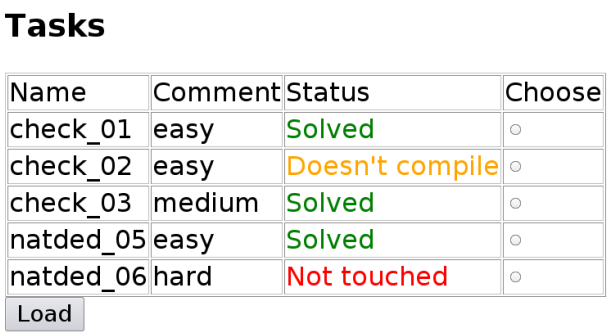

Fig. 4. Tasks assigned to students and their status.

The verification tool lexes the original task and the student's solution in parallel. The original solution includes placeholders that are valid Coq comments. Those placeholders mean that a particular place needs to contain a valid Coq term or a valid proof. For proofs the kind of proof determines 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 definition of a particular object in type theory. For this kind of tasks manual verification by a teaching assistant of a course is required.

7 Future work

Some of issues that currently are being worked on are:

- The course notes that are to accompany the system are not yet finished and the documentation of how to install and maintain the server is quite rudimentary. Our server currently is available to everyone who wants to experiment with our system, but due to privacy concerns it might be necessary for universities to install servers of their own.
- 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 Mozilla compatible browsers (Firefox, Galeon, Epiphany, Netscape, ...), Opera and Safari. 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' [3]. Here, traditional wiki technology is integrated with the same proof assisntant front end that our system is based on.

References

[1] 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.

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

[3] 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.

[4] Geuvers, H. and R. Nederpelt, *Rewriting for Fitch style natural deductions.*, in: V. van Oostrom, editor, *RTA*, Lecture Notes in Computer Science **3091** (2004), pp. 134–154.

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

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

[7] 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.

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