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 that is currently being developed in Nijmegen and Amsterdam for teaching logic to undergraduate computer science students. This 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 will be a large database of logic problems. This database will also hold the solutions of the students. This means that 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 will be 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 a type theory course, and is currently being further developed in the first year logic course of computer science in Nijmegen.

Keywords: Logic Education, Proof Assistants, Coq, Web Interface, AJAX, DOM, Natural Deduction, Gentzen, Fitch

1 Introduction

1.1 Motivation

At every university, 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

[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'.

is called 'Inleiding Logica' ('Introduction to Logic'). Almost all computer science curricula will have similar undergraduate courses.

For learning this kind of elementary mathematical logic it is crucial to make many exercises. Those exercises can of course be made 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.

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.

1.2 Our contribution

This paper describes a system, currently named ProofWeb, that is in development at the RU in Nijmegen and at the VU in Amsterdam. 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 the dialect of the ML programming language called

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 [9].

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 (presently over 200 exercises) so 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. The course notes are still under development.

1.3 Related work

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 [10]. 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).

1.4 Contents

In the rest of the paper we present both our project and the current state of the system that we are building. We start with a short description of our project in Section 2, and discuss our experiences so far in Section 3. Next, in Section 4 we present the architecture of the interface. Section 5 is 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 and work that is currently in progress.

2 Structure of the project

The project of developing ProofWeb is financed by the SURF foundation [12] (the Dutch organization for computers in academic teaching) and runs in the period fall 2006 till fall 2007 (three semesters). Cezary Kaliszyk is employed for a full year at half time to program the system, while Maxim Hendriks is employed for half a year at full time to develop the educational materials (the database of problems and the course notes), as well as to evaluate the educational success of the project.

We identified the following nine sub-tasks, called 'work packages':

(i) the database of the system,

(ii) Coq tactics that exactly correspond to the rules the logic,

(iii) graphical representations for the proofs,

(iv) checking a Coq file against an exercise,

(v) a large set of logic problems,

(vi) course notes that explain the system,

(vii) using the system in actual courses,

(viii) dissemination of the results of the project,

(ix) evaluation of the project.

3 Experience so far

The system ProofWeb is used in the following advanced courses:

(i) In fall 2006: the course 'Logical Verification' [11] 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. Natural deduction is taught in Gentzen style, that is, proofs have a tree-like structure, and grow upward from the conclusion of the proof.

(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.

Moreover, ProofWeb is or will be used in the following introductory courses:

(i) In spring 2007: the course 'Beweren en Bewijzen' [1] at the RU, taught by Hanno Wupper and Erik Barendsen. This is a computer science undergraduate course in logic, with natural deduction in Gentzen style.

(ii) In fall 2007: the course 'Inleiding Logica' [6] 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 7), 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'.

The course 'Logical Verification' at the VU in fall 2006 was a first opportunity to test the system. About 25 students followed and completed the course. They were all mature (graduate) students, very well able to deal with a system that was still in beta. A part of the course consists of learning type theory and Coq via basic (undergraduate) logic exercises, which were done using the system ProofWeb. We learned the following from the use of the system ProofWeb in this course.

Initially we did not have a dedicated server, so it was running on one of the group servers of the research group in Nijmegen on a non-standard port. One of the issues was, that the web-proxy at the VU did not allow the students to access pages running on non-standard ports, so they were required to turn the proxy off.

One of the assignments 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 the twenty-five students were able to use about 400Mb memory and a fraction of a CPU. This might be thanks to the fact that the students were not using tactics that involve automation.

During this course there was not yet support for visualizing proofs. 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 the course is to learn Coq.

The second course in which ProofWeb is used is the course 'Type Theory' in spring 2007 at the RU. The first half of this course is basically an accelerated clone of the 'Logical Verification' course. 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 of 'Logical Verification' together with access to the server. They then would work on their own, with an opportunity to call for help if needed. It turns out that this worked unexpectedly well. The students just studied the lecture notes and did the exercises of the course. And even without much pressure on them in the form of requiring them to meet deadlines, they managed to keep on schedule reasonably well. The only thing that at some point confused them (after which a lecture was organized to make things clear) was the part of the course that did not correspond to Coq work: derivations in Pure Type Systems.

All in all our experience so far is that the system ProofWeb seems to work very well in teaching. Indeed, hardly any students used more traditional Coq interfaces like Proof General or CoqIDE. The courses so far are more advanced ones, so it remains to be seen whether ProofWeb also works well for larger numbers of undergraduate students, but we are optimistic about that. In addition, as of May 2007, the progress on all of the nine work packages seems to be well on target.

4 Architecture of the interface

In this section we shortly describe the architecture of the interface to Coq used in ProofWeb. The interface 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 plugins are required).

To obtain 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 web browsers;
- *Document Object Model* (*DOM*) — a way of referring to subelements of a web page that allows modification of the page on the fly, creating dynamic elements;
- XmlHttp — an API available to client side scripts, that allows requesting information from the web server without reloading the page.

The asynchronous DOM modification 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.

ProofWeb uses an implementation of this architecture that is used to create a web interface for proof assistants. The server stores sessions for all users, and the clients are presented with an interface that is completely contained in a web browser, but resembles and is comparably responsive to a local interface like Proof General or CoqIDE (see Figure 1).

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 the directories of the students of these courses.

An example of the use of the interface in the 'Logical Verification' course can be seen in Figure 2.

5 Natural deduction for first-order logic

This section is concerned with natural deduction proofs for first-order logic in 'Gentzen style', where a proof is a tree.

5.1 Tactics

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. Because in principle the full power of Coq is available, this means that we had to write tactics (in effect, to dumb Coq down) to match the traditional logic rules. What then arose was that, whereas in a Coq proof one can look at a hypothesis and

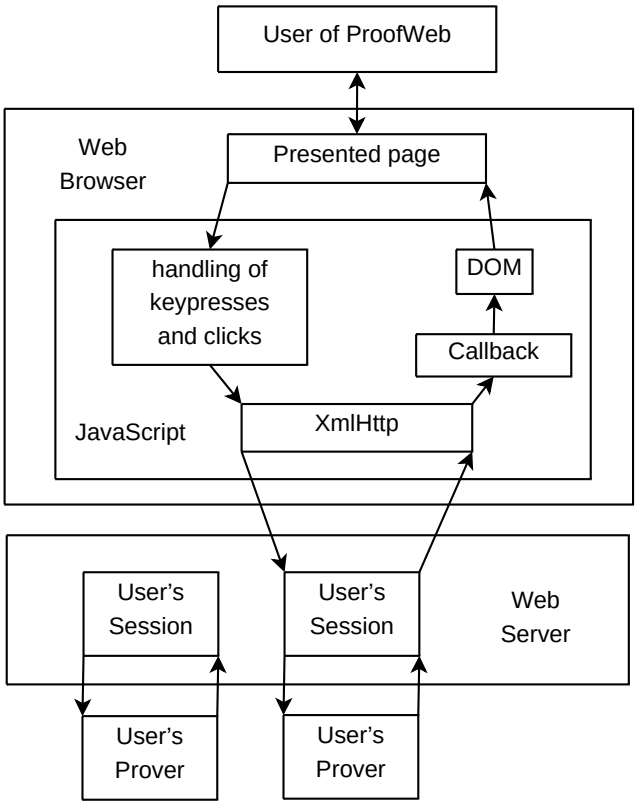


Fig. 1. ProofWeb architecture.

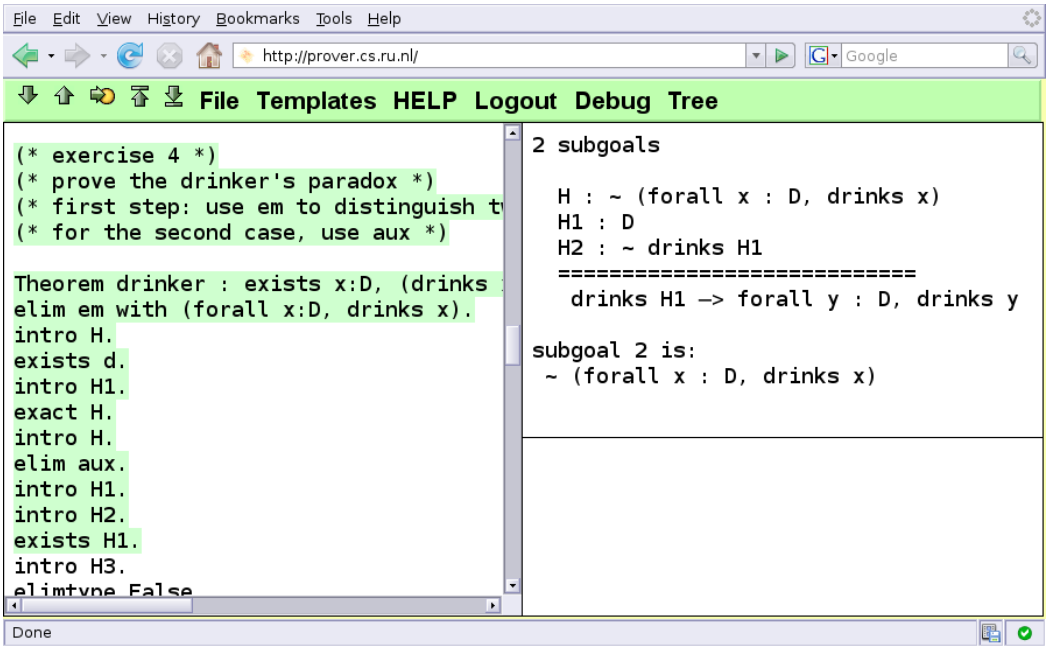


Fig. 2. ProofWeb in the 'Logical Verification' course.

eliminate it, ending up in a new proof state, traditional natural deduction offers no such jumps. So 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 7).

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_el X H1 H2 :=
  match X with
  | ( _ \/ _ ) =>
   assert X;
   [ idtac |
     match goal with
       | x : X |- _ =>
         elim x; [intro H1 | intro H2]; clear x
     end
   ]
  | _ => fail "The first argument is not a disjunction"
end.
```

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

(iii) C, but now with the extra assumption B with name (or proof, if viewed constructively) H.

Also, the tactic gives a nice and understandable error message. All the 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). We give a small example of a proof with our set of tactics, and hope it speaks for itself:

```
Theorem example : ((A \/ B) /\ ~A) -> B.
Proof.
imp_in z.
dis_el (A \/ B).
con_ell (~A).
ass z.
imp_in y.
neg_el A.
con_elr (A \/ B).
ass z.
ass y.
imp_in x.
ass x.
Qed.
```

5.2 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 graph 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. `Show` allows the user to see a non-current goal, `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>
```

We modified ProofWeb to be able to parse XML trees dumped by Coq and generate natural deduction diagrams (see Figure 3). 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.

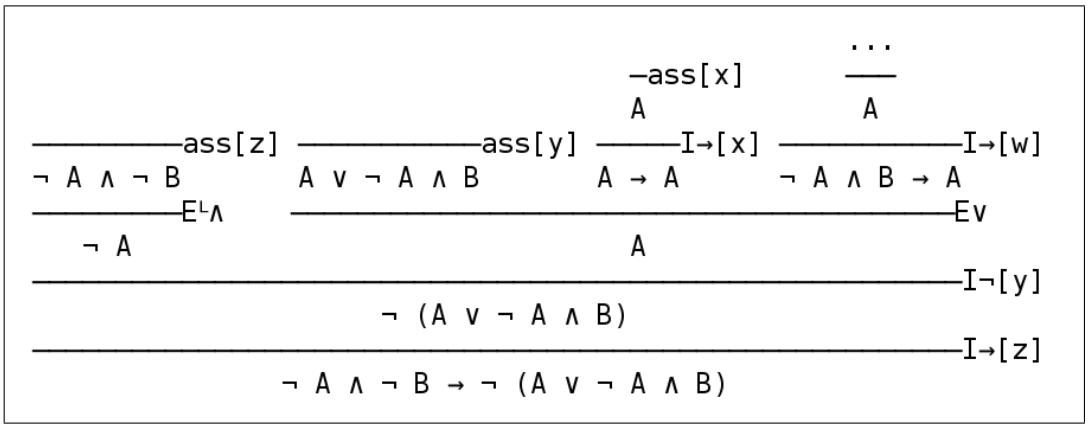


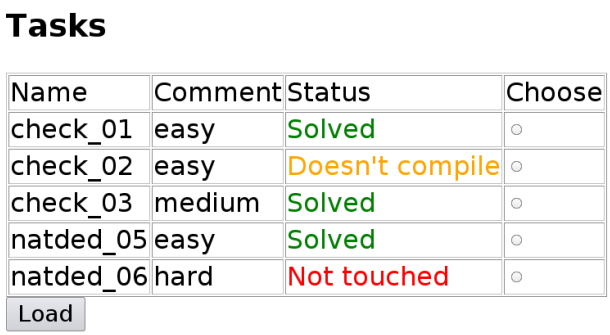
Fig. 3. A natural deduction tree as seen on the webpage (cropped screenshot).

6 The exercise set

Also part of the project is the development of a set of exercises for the students. For a particular course, a number of exercises assigned to the students. It is desirable that ProofWeb can be used as a complete course environment. So 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 (see Figure 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.
- Does not 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.
- Incorrect — 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 incorrect.
- Correct — Passed the verification by our tool.

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.



• **Tasks**

Name	Comment	Status	Choose
check_01	easy	Solved	○
check_02	easy	Doesn't compile	○
check_03	medium	Solved	○
natded_05	easy	Solved	○
natded_06	hard	Not touched	○

Load

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

7 Outlook

At the moment of writing, the project ProofWeb is more or less half way, so this paper reports on work in progress. In this section we first discuss a main issue we are currently working on: adding the possibility of using the system for 'Fitch-style' natural deduction derivations. We then briefly comment on further points of current and future work.

7.1 Fitch-style deductions

The most important improvement is to add 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 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'. We are working on having the system display Fitch-style deductions. A basic version of this has already been implemented (see Figure 5), but needs further development. A number of issues arise. When a tactic creates a number of assumptions, should these be kept in one flag or should the system create multiple flags? Also, Fitch-style deductions may include repetitions of assertions assumed by flags. Most of these repetitions are redundant. However, not repeating assumptions immediately before they are used leads to very unreadable proofs. What should be done? We are currently looking at adapting the approach presented in [4] to Coq tactics.

7.2 Future work

Some of the other issues that currently are being worked on are:

- The course notes that are to accompany the system. These are still in a rudimentary stage.
- There is no separate web interface yet for the teacher to manage the student logins and the set of exercises for the course, nor to inspect the work of the students.

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

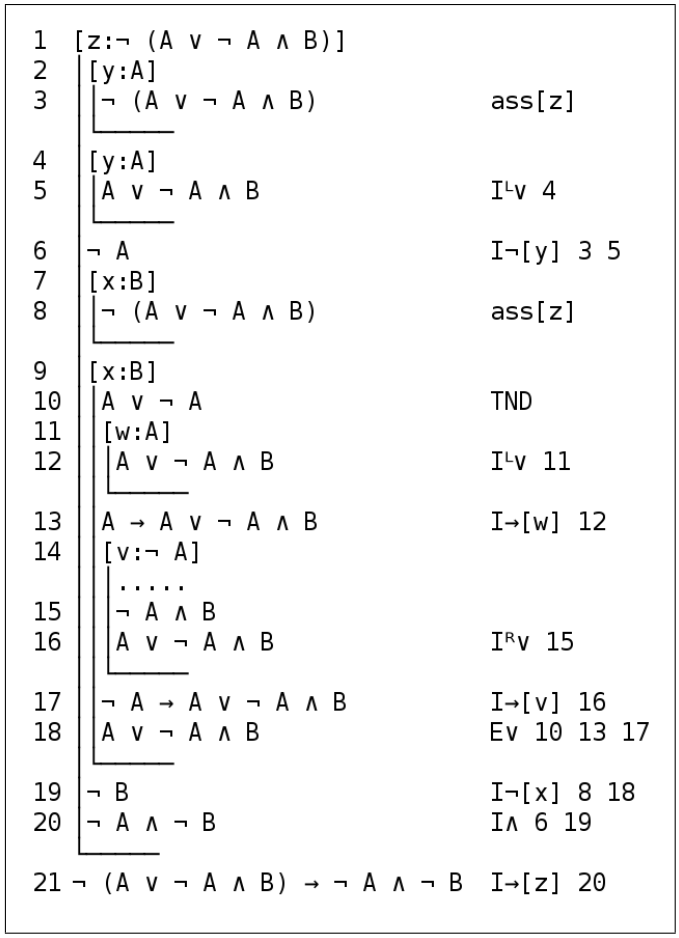


Fig. 5. A Fitch-style deduction as currently displayed by the system (implementation in progress).

At the moment this is 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 in an HTML 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, ...). Also, some effort has been made to make the system work reasonably well with the most common versions of Internet Explorer. However, the compatibility of the system with most common web browsers is something that will need further attention.
- At the moment there is hardly any documentation of how to install and maintain the server. Our server currently is available to everyone who wants to experiment with our system, but there is no good guide available that explains how to install a server of his own. Because the server is still very much in a constant state of

flux, documenting the installation and maintenance processes is at this moment not yet reasonable. However, in the final phase of the project it will be important to also create this kind of documentation.

- With the current version of 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' such a student session (possibly speeded up or slowed down). 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.

7.3 Beyond the project

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 in Nijmegen is the creation of a so-called 'math wiki' [8]. Here, traditional wiki technology is integrated with the same Coq front end that our system is based on.

7.4 Using the system

We think that it is important that our system is experimented with (and hopefully someday frequently used) by as many people as possible. For this reason, we currently offer the use of our system to anyone on the internet, even without any registration. The ProofWeb system can be tried at

http://prover.cs.ru.nl

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 (2005).
URL http://coq.inria.fr/doc-eng.html

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

[7] Kaliszyk, C., *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*, 2006, pp. 53–64, to be published in ENTCS.

[8] Kaliszyk, C. and P. Corbineau, *Cooperative repositories for formal proofs* (2007), to be published in the proceedings of MKM 2007.

[9] 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.

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

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

[12] *SURF foundation.*
URL http://www.surf.nl/