Deduction using the ProofWeb system

Cezary Kaliszyk
Femke van Raamsdonk
Freek Wiedijk
Hanno Wupper
Maxim Hendriks
Roel de Vrijer

Abstract

This is the manual of the ProofWeb system that was implemented at the Radboud University Nijmegen and the Free University Amsterdam in the SURF education innovation project *Web deduction for education in formal thinking*.

Contents

1 Introduction

This document is the manual of the ProofWeb system. If you are just interested in using the system you can best skip this introduction and directly go to Chapter 2 on page 5.

This document does *not* explain mathematical logic in detail, but is intended as a companion to a logic textbook when using the ProofWeb system. (For instance, one can read [6] for Gentzen style 'tree' proofs, or [3] for Fitch style 'box' proofs.) However, the appendices of these notes can be used as a reference for natural deduction.

ProofWeb is a system for practising natural deduction on a computer. It is meant to be used in introductory courses in mathematical logic. The logics that ProofWeb has been specifically designed for are the usual basic ones:

– first order propositional logic
– first order predicate logic
– first order predicate logic with equality

ProofWeb is designed for the classical variants of these logics, but it is easy to work in ProofWeb in the constructive fragments of these logics as well.

ProofWeb combines the following features:

- ProofWeb has been designed to be as close as possible in look and feel to the way mathematical logic is taught in textbooks. For Fitch style 'box' proofs it specifically matches the conventions of *Logic in Computer Science: Modelling and Reasoning about Systems* by Michael Huth and Mark Ryan [3], published by Cambridge University Press.

- When using ProofWeb one is using the Coq proof assistant (developed at the INRIA institute in France) without any restrictions. This means that with ProofWeb one is using a system that has the power to build computer proofs like the one of the Four Color Theorem [2], or a proof of the correctness of an optimizing C compiler [4].

- ProofWeb is used through the web. One does not need to install any special software to be able to use the system. A web browser (without any ProofWeb specific plug-ins and even without Java) is all that is needed. Students will be able to work on their exercises from any computer connected to the Internet (like for instance a computer in an Internet café), without first having to copy their files or to install any special software.

- With ProofWeb all files are on a central server. Teachers will at all times be able to see the status of their students' work, including their solutions to the exercises in full. Also, the students' work can be easily backed up.

- ProofWeb comes with free course notes explaining the system. (This is the text that you are currently reading.)

- ProofWeb comes with a database of around two hundred basic exercises.
- ProofWeb will automatically check whether solutions to exercises are correct or not, and present this information both to the student and the teacher in a clear tabular format.

If you are a *student* there are two ways to work with ProofWeb:

1. Just go to the ProofWeb server provided by the Radboud University Nijmegen

   `http://proofweb.cs.ru.nl/`

   and use the free guest account. There is no need to register or to provide a password. (In this mode the full set of exercises is not available. This way of working with ProofWeb is just for trying it out.)

2. If you follow a course in mathematical logic that uses ProofWeb for the practical work, then go to the ProofWeb server of that course (which often also will be `proofweb.cs.ru.nl`), and use the account there that your teacher will have made for you. Ask your teacher for the username and password of that account.

If you are a *teacher* and want to use ProofWeb in your course, then there also are two ways for this:

1. Either you can host your course on the server of the Radboud University Nijmegen. To do this send mail to

   `proofweb@cs.ru.nl`

   and provide details about the course that you want to host. This service is currently provided for free.

2. Alternatively, you can host a ProofWeb server on a machine of your own. To do this go to

   `http://proofweb.cs.ru.nl/`

   follow the link about installing the ProofWeb system, and follow the installation instructions that will tell you what files to download to install the ProofWeb system and what to do with them. (To follow these installation instructions on how to install ProofWeb, you will need a machine running Linux.)

2 Using the system

To use ProofWeb you need a web browser on a machine connected to the Internet. Unfortunately not all web browsers properly support the technologies that ProofWeb uses. For this reason Internet Explorer cannot be used. Browsers that have been tested to work are recent versions of:

- Gecko based browsers, like Mozilla, Firefox, Galeon, Epiphany and Netscape.
- Webkit based browsers, like Safari and Konqueror.
- The Opera browser.

The ProofWeb system has been developed using Firefox.

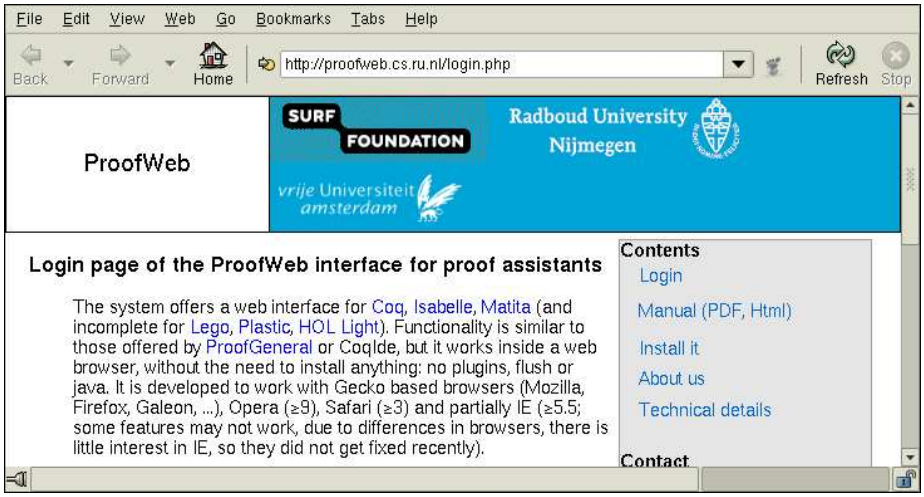
2.1 Logging in

We will now explain the system by walking through a ProofWeb session.

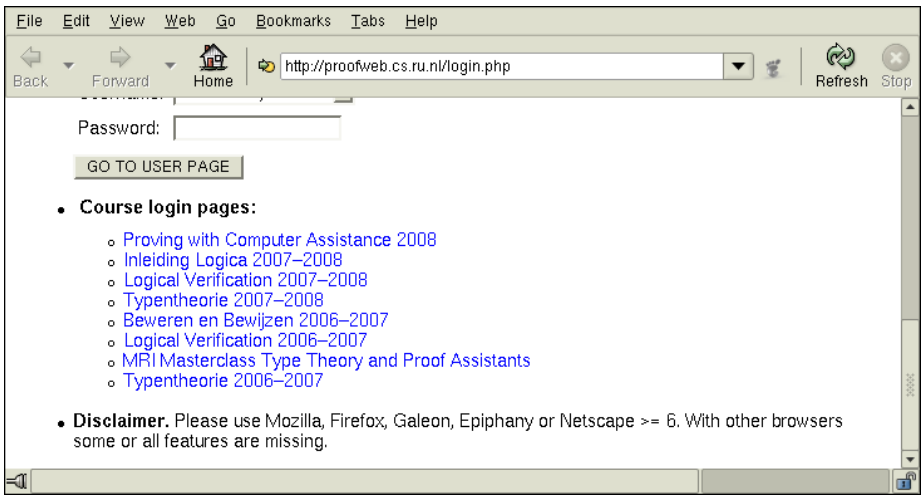
We go to the URL of our ProofWeb server. In the example that we are showing here this is

`http://proofweb.cs.ru.nl/`.

This shows us the main ProofWeb web page:



We scroll down to the list of courses and click on the link of our course:

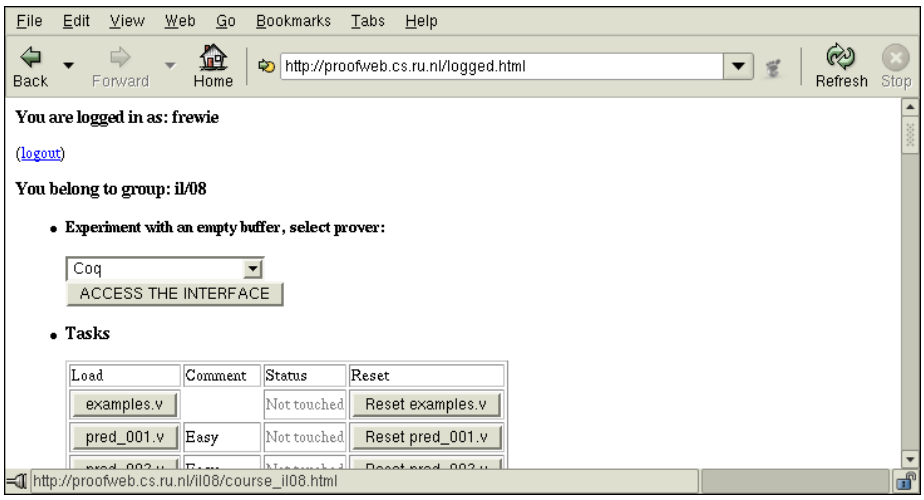


In this example we click the link '`Inleiding Logica 2007-2008`' (which is Dutch for 'Introduction to Logic'.) This will get us to the login page for the course:

We enter our username and password and click '`GO TO USER PAGE`'. This logs us in to the system for the course that we are following.

2.2 The list of exercises

We now see 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.



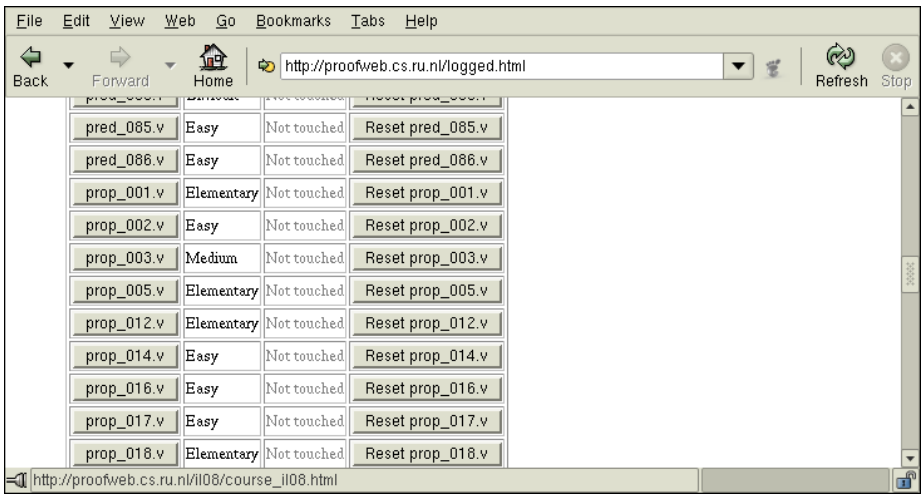
The four possibilities for the status of an exercise are:

- Not touched
- Incomplete
- Correct
- Solved

The colors are meant to resemble the colors of a traffic light. (The challenge of ProofWeb is to have your list of exercises be all green.)

We will explain the meaning of these statuses in detail below. (The difference between 'Correct' and 'Solved' is that according to the Coq system in both cases the solution to the exercises has no errors, but that only in the latter case is the solution restricted to deduction rules that are allowed for the course.)

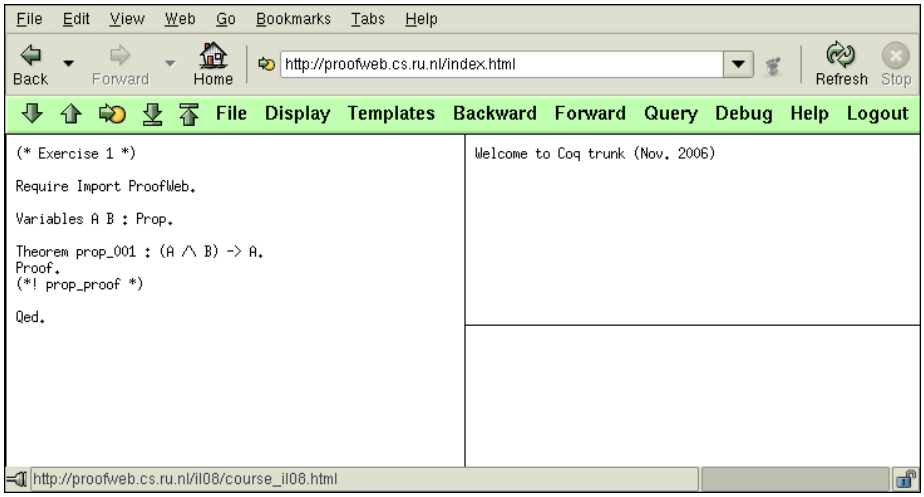
Now we scroll down in the list of exercises to the first exercise about propositional logic called prop_001.v:



We click the button with the name of this file on the left. This will open the exercise in the prover interface.

2.3 The prover interface

We are now looking at the prover interface of ProofWeb:



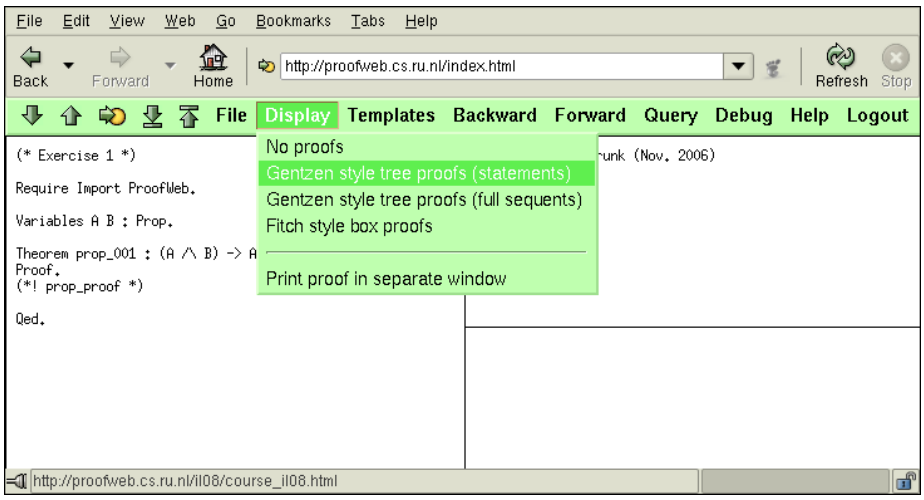
It consists of a green menu bar at the top of our browser window, and three *panes*:

- On the left there is the pane in which we are editing a *proof script*.
- On the top right there is the pane that will show the *proof state* of our proof. We will explain below what this means. The proof state in this pane is shown in the style that is standard when working with the Coq proof assistant.

  Here also will be shown the messages of the Coq system that we are communicating with on the ProofWeb server.
- On the bottom right there is the pane that shows the *proof* that we are working on. The proof in this pane will be shown in the style that is customary in mathematical logic teaching.

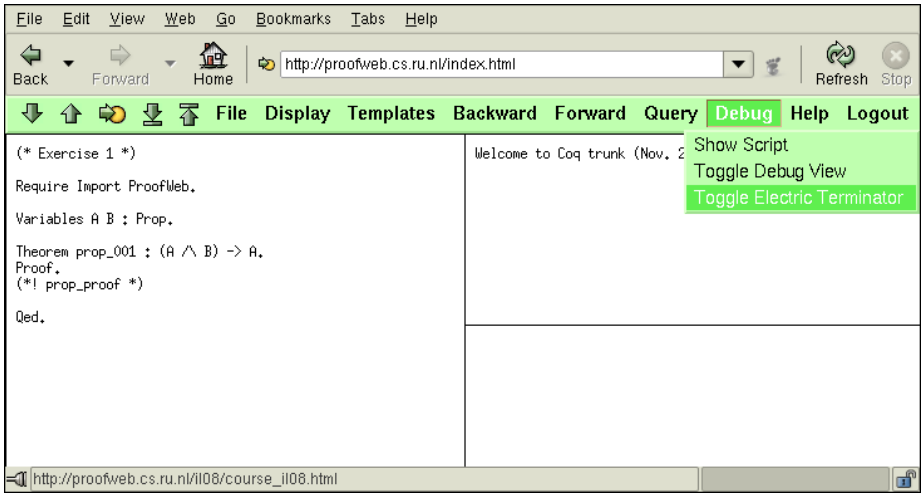
ProofWeb can show the proof in the bottom right pane in three different ways. Depending on the course that you are following one of these proof display styles already may be active.

To change the style of proof display one uses the Display menu. Let us first put the system in the mode in which it displays proofs in Gentzen 'tree' style:



Another setting that one might like to change when starting working with ProofWeb is the 'electric terminator'. When this feature is active, typing a period ('.') automatically executes the command that it terminates. (Some people like this convenience, while others prefer to separate typing their script from having the system execute their commands.)

To activate the electric terminator select Toggle Electric Terminator from the Debug menu:



We are now ready to start working on the exercise. Currently the text on the left (the solution to the exercise that we are editing) has not yet been executed. This text is a *script*, consisting of commands that are all terminated by a period character ('.'). One can execute those commands by clicking the downward arrow in the green menu bar. Alternatively one can execute the next command

by holding down the control key while simultaneously typing the down-arrow key on the keyboard.

We now execute the commands (by clicking the down arrow) until we have executed the

```
Proof.
```

command. (At this point we are working on a proof, and an incomplete proof will now be displayed on the lower right.) Then we remove the text

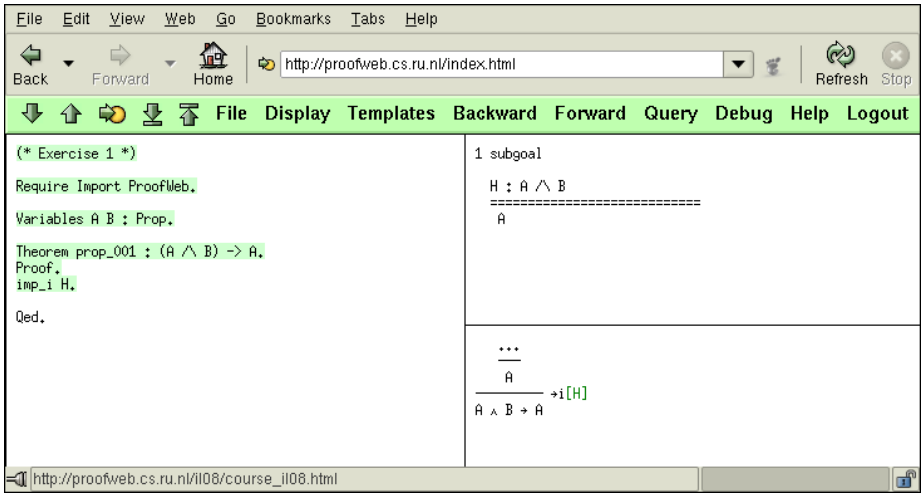
```
(*! prop_proof *)
```

from the script. It should be replaced by a series of commands that create a proof of the statement. We type the first command:

```
imp_i H.
```

and execute it. (The details of the commands that are available and what they mean will be given below.)

We now have:



On the bottom right the incomplete proof after this command now is:

$$\dfrac{\begin{array}{c}\vdots\\ A\end{array}}{A \wedge B \to A}\ {\to}\mathrm{i}\ [\mathrm{H}]$$

The dots are the part of the proof that is not yet completed. It has to be replaced by a valid proof of A by issuing more commands in the proof script.

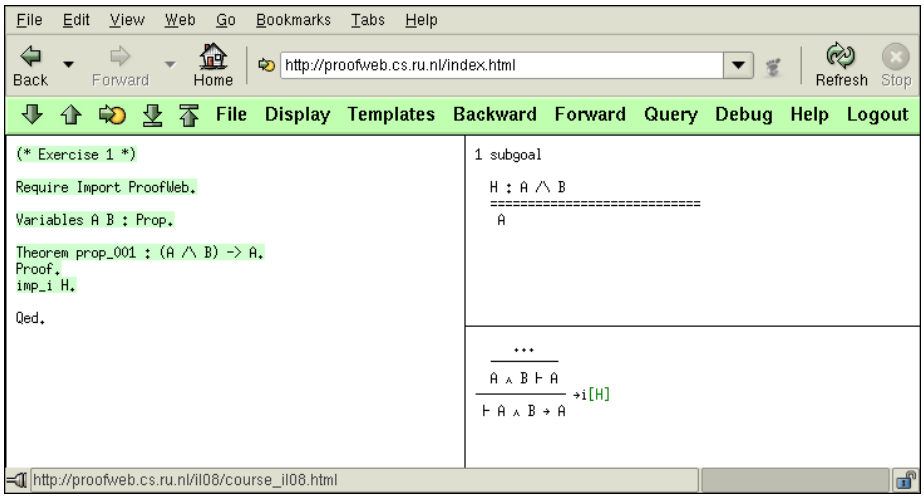
On the top right we see the *proof state* of the Coq proof assistant that is checking our proof. It looks like:

```
1 subgoal

  H : A /\ B
  ============================
   A
```

It means that our first *subgoal* (out of only one) is to prove the statement below the line, A, where we are allowed to use the *assumptions* above the line. In this case there is only one assumption, $A \wedge B$. The *label* of this assumption is `H`, which is how we should refer to the assumption in the script. This 'subgoal' corresponds to the triple dots in the proof on the lower right.

We now look at what the proof looks like in the two other display styles. First here is what it looks like when instead of just statements we ask the system to display *sequents*:



In this case the allowed assumptions are showed to the left of the 'turn-style' symbol ('$\vdash$'). Instead of just

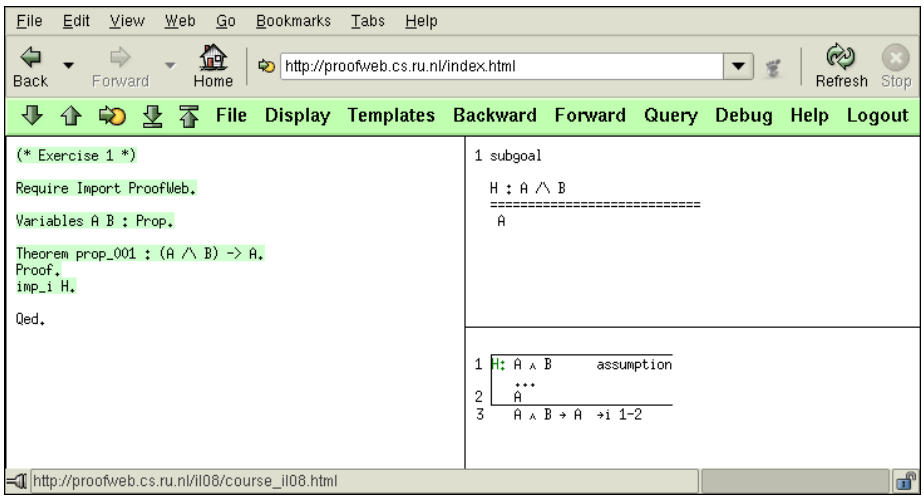
$$A$$

as the middle line of the proof we now have

$$A \wedge B \vdash A$$

Despite the change in presentation the proof has stayed exactly the same.

The third style is Fitch's 'box style':



In this style the (incomplete) proof looks like:

1	H:	$A \wedge B$	assumption
		$\ldots$	
2		A	
3		$A \wedge B \rightarrow A$	$\rightarrow$i 1—2

Again the dots have to be filled in to complete the proof.

We will now finish the exercise. We add two more commands to the script:

```
con_e1 (B).
exact H.
```

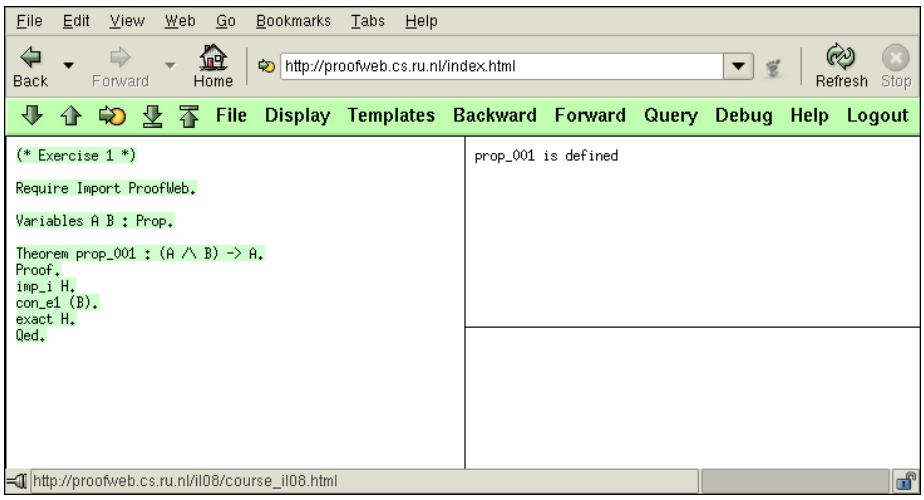
execute them, and then also execute the final

```
Qed.
```

This leads to the message

```
prop_001 is defined
```

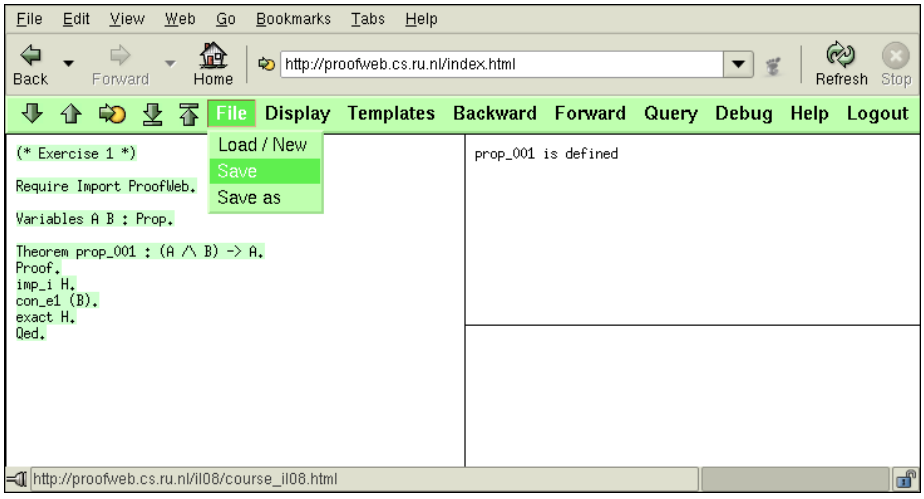
at the top right where Coq shows its output:



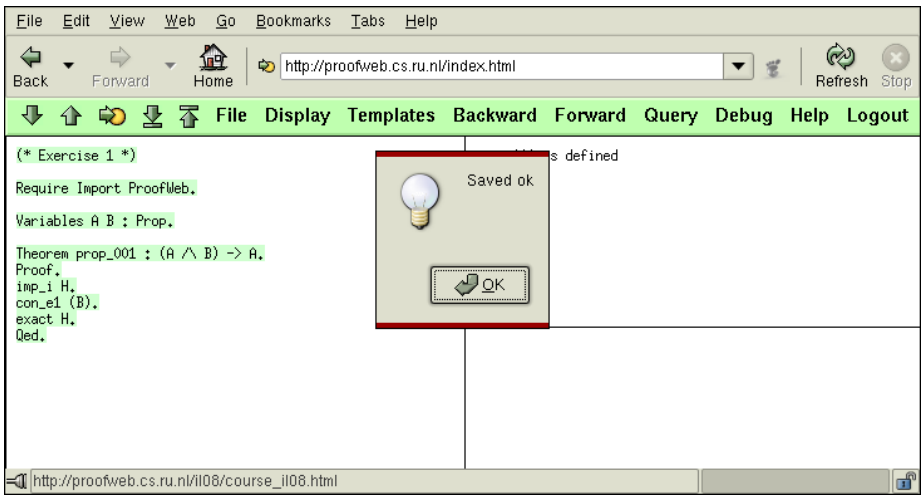
It means that the proof script is complete, which means that it has generated a correct proof of the statement.

2.4 Saving exercises

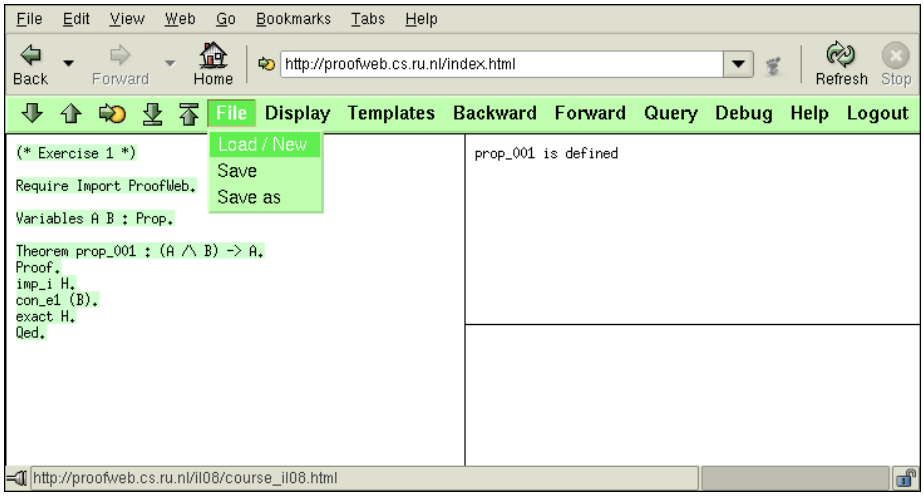
We are now done with exercise (the script on the left is what it should be). We save the file on the server by selecting Save from the File menu:



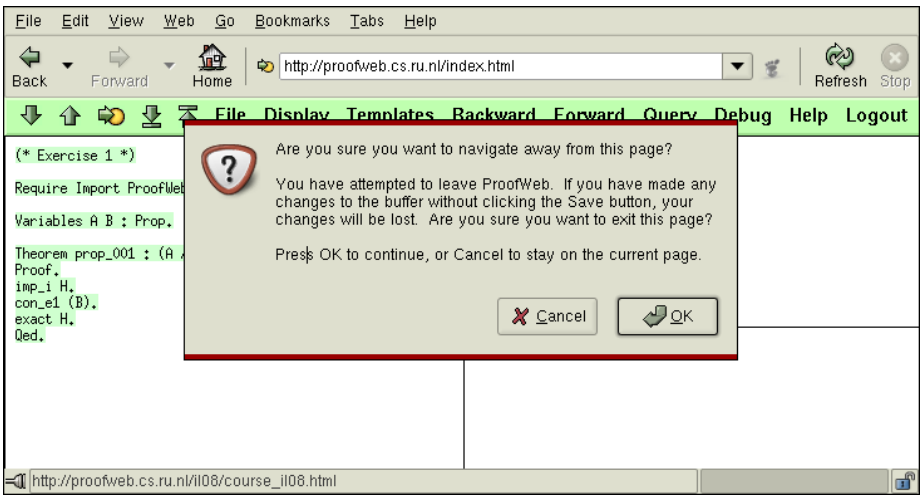
This saves the script as the file prop_001.v in the student's directory on the ProofWeb server:



Next, we select Load / New from the File menu to get back to the list with exercises:



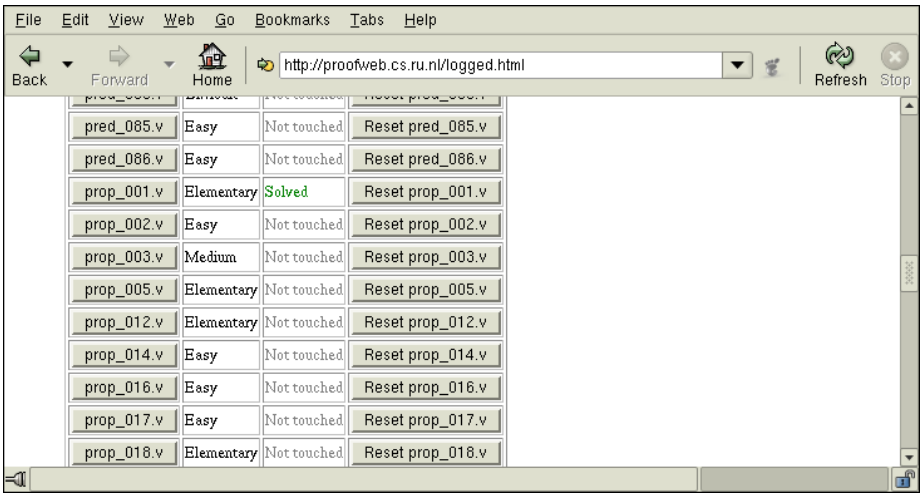
The system will warn us that we are leaving the page with the script (it does not realize that it just has been saved):



Generally it is a good thing that ProofWeb will warn you about this, but in this case we can of course ignore this message and click OK.

2.5 The status of the exercises

We now again are looking at the list of exercises:

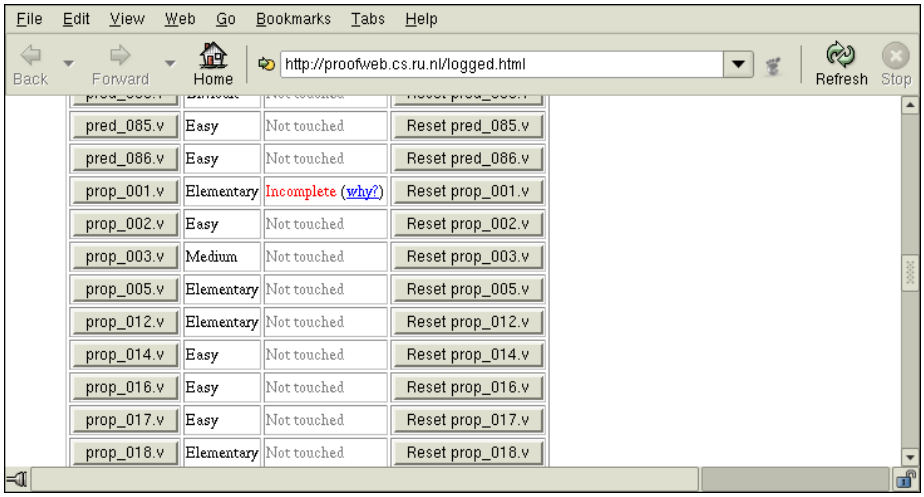


The status of prop_001.v has changed from a gray Not touched to a green Solved because we saved a correct solution.

Instead of going for the next exercises, let us look what happens when we save an incorrect solution. We go back to the exercise by clicking the prop_001.v button once more, remove the lines

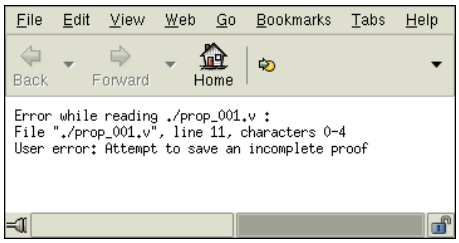
```
con_e1 (B).
exact H.
```

from the proof script, save again, and go once more to the list of exercises:



The status now will be a red Incomplete. This also will be the status when the file is not so much incomplete as well just *wrong*.

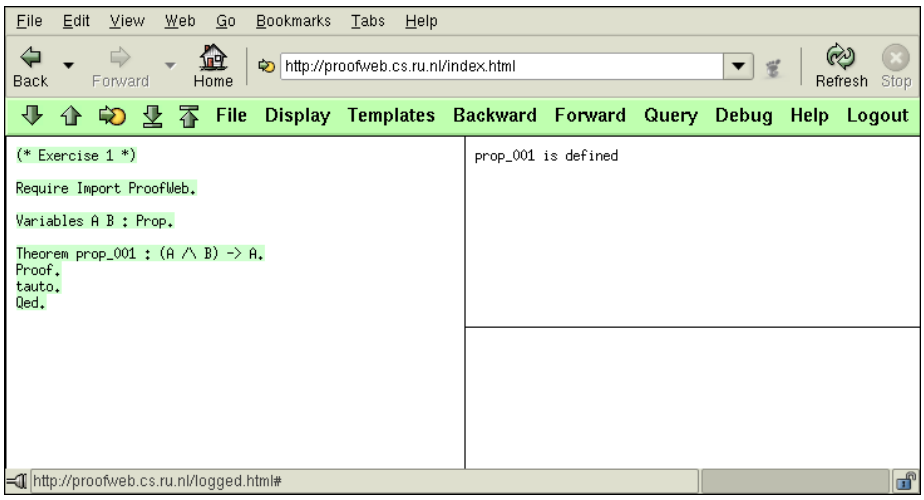
If you want to know why ProofWeb thinks there is an error in your solution, you can click the why? link next to the status. You will get a new window that shows the error message of Coq:



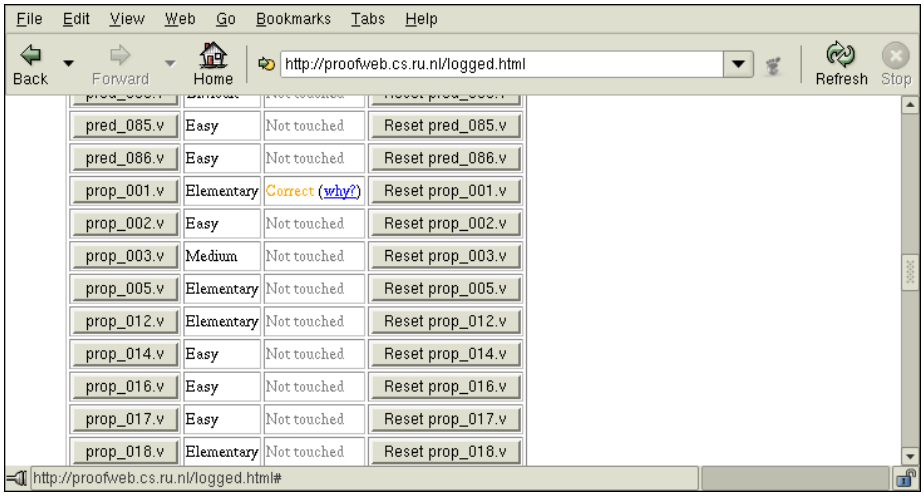
ProofWeb also has an orange status that is in between the green and the red ones saying Correct. You get this when the file is correct Coq, but you modified it in a way that is not allowed for the course. For instance, you might ask Coq to do the proof for you, by using the

```
tauto.
```

command, which makes Coq run an automated propositional theorem prover:



This will prove the statement, but of course this is not an allowed solution to the exercise. Hence the status will become Correct instead of Solved:



The Correct status also will be given if you just delete all the text in the file, as the empty file is a correct Coq file too. Of course that is not an allowed solution to the exercise either.

Again, when the status is Correct, you can click on why? to find out what you did that ProofWeb thinks is not allowed in the course. (The word why? is a bit strange in this context, as Coq will not so much explain why the thing is *correct*, but instead why it is *not* 'solved'.) These error messages generally are a bit cryptic as well:

This ends our example of a ProofWeb session. We will now discuss the system in detail. First we will explain how to write formulas in the ProofWeb system.

3 Writing formulas

The ASCII syntax for formulas of first order predicate logic is given by the following table:

$\bot$	`False`
$\top$	`True`
$\neg A$	`~` A
$A \wedge B$	A `/\` B
$A \vee B$	A `\/` B
$A \to B$	A `->` B
$A \leftrightarrow B$	A `<->` B
$\forall x\, A$	`all` x`,` A
$\exists x\, A$	`exi` x`,` A

For example the formula

$$\exists x(P(x) \vee \neg Q(a)) \to Q(a) \to \exists x\, P(x)$$

is written in ASCII syntax as

```
exi x, (P(x) \/ ~Q(a)) -> Q(a) -> exi x, P(x)
```

Note that in ProofWeb one puts a comma after a quantifier.

Another slight deviation of the ProofWeb formula syntax from standard syntax is that if a predicate has just a single variable as the argument, one is allowed to omit the brackets. (It is not obligatory, though.) Therefore the formula also might be written as:

```
exi x, (P x \/ ~Q a) -> Q a -> exi x, P x
```

In fact this is how Coq prints this formula back at the user. (In the exercise files the formulas are always written with the brackets present though.)

A note for people who know the Coq system: In Coq a different notation for the quantifiers is customary:

$\forall x\, A$	`forall` x `:` D`,` A
$\exists x\, A$	`exists` x `:` D`,` A

In this D is the domain over which one quantifies.

There are two differences between the ProofWeb quantifiers and the standard Coq quantifiers. First, the ProofWeb quantifiers have a priority that is customary in logic textbooks: they bind more strongly than the propositional connectives. The Coq quantifiers bind weakly, which is the standard in proof assistants. For instance the formula

```
all x, A \/ B
```

is read as

$$(\forall x\, A) \vee B$$

which is the reading of

$$\forall x\, A \vee B$$

in logic textbooks. However

```
forall x : D, A \/ B
```

is read as

$$\forall x\, (A \vee B)$$

Second, with the Coq quantifiers one indicates the domain. The ProofWeb quantifiers do not need this, as they use a fixed domain `D`. This domain is defined in the file `ProofWeb.v` which is loaded at the start of the ProofWeb exercises.

The Coq quantifiers can be useful for people who want to use ProofWeb for complicated examples, in which different variables have different 'types'. In that case they can just use the Coq versions of the quantifiers. The ProofWeb tactics, which are the subject of the next two chapters have been programmed to work with either quantifier style.

4 Gentzen style tree proofs

This chapter explains how to build natural deduction proofs using ProofWeb. It treats a style of natural deduction introduced by Gerhard Gentzen in 1935 in his dissertation at the University of Göttingen. We will here call that style 'tree proofs', to distinguish it from the 'box proofs' which will be the subject of the next chapter.

In this chapter we will not explain natural deduction proofs in detail. You will need to read a textbook for that, like Dirk van Dalen's *Logic and Structure* [6]. Here we will just explain how to do this kind of proof in the ProofWeb system.

4.1 Example on paper

Here is an example of a natural deduction proof in predicate logic, in the way that it is written on paper:

$$\dfrac{\dfrac{[\exists x(P(x) \vee \neg Q(a))] \qquad \dfrac{\dfrac{[P(b) \vee \neg Q(a)] \qquad \dfrac{\dfrac{[P(b)]}{\exists x\, P(x)}\,\exists\text{i} \qquad \dfrac{\dfrac{[\neg Q(a)] \qquad [Q(a)]}{\bot}\,\neg\text{e}}{\exists x\, P(x)}\,\bot\text{e}}{\exists x\, P(x)}\,\vee\text{e}}{\exists x\, P(x)}\,\exists\text{e}}{Q(a) \to \exists x\, P(x)}\,\to\text{i}}{\exists x(P(x) \vee \neg Q(a)) \to Q(a) \to \exists x\, P(x)}\,\to\text{i}$$

In different textbooks the precise notation for this kind of proof will vary, but whatever the details are, natural deduction proofs in the 'tree' style have a shape that is close to this.

Differences between textbooks might be that the order of the connective and the 'i' or 'e' can be the other way around, in which case it is i→ instead of →i, or the 'i' or 'e' has to be capitalized, in which case it is →I instead of →i. Another difference is that often the assumptions at the 'leaves' of the proof tree are not put in square brackets, but instead are crossed out with a diagonal slash. (Crossing out an assumption in such a way while doing a proof can give a very satisfactory feeling. However, on a computer screen it is a bit hard to implement. The convention of the square brackets we took from [6].)

In natural deduction proofs the rules and assumptions generally are annotated to indicate which assumption originates from which rule invocation. In ProofWeb we do this in green in the following style:

$$\dfrac{\dfrac{[\exists x(P(x) \vee \neg Q(a))]^{\texttt{H1}} \qquad \dfrac{\dfrac{[P(b) \vee \neg Q(a)]^{\texttt{H3}} \qquad \dfrac{\dfrac{[P(b)]^{\texttt{H4}}}{\exists x\, P(x)}\,\exists\text{i} \qquad \dfrac{\dfrac{[\neg Q(a)]^{\texttt{H5}} \qquad [Q(a)]^{\texttt{H2}}}{\bot}\,\neg\text{e}}{\exists x\, P(x)}\,\bot\text{e}}{\exists x\, P(x)}\,\vee\text{e}\ [\texttt{H4},\texttt{H5}]}{\exists x\, P(x)}\,\exists\text{e}\ [\texttt{H3}]}{Q(a) \to \exists x\, P(x)}\,\to\text{i}\ [\texttt{H2}]}{\exists x(P(x) \vee \neg Q(a)) \to Q(a) \to \exists x\, P(x)}\,\to\text{i}\ [\texttt{H1}]$$

Again, the way this annotation is done differs between textbooks. Often the annotations are subscripts or superscripts of the rules, and often the annotations are numbers. (Here we use the labels that one gets in the Coq system, which are not numeric.)

Now here is the way this same proof is displayed in the ProofWeb window. As you can see it is very similar to the mathematical style that we showed. The main difference is that the lines are a bit longer.

$$\dfrac{\dfrac{[\exists x,\ (P\ x \vee \neg Q\ a)]^{H1} \qquad \dfrac{[P\ b \vee \neg Q\ a]^{H3} \qquad \dfrac{[P\ b]^{H4}}{\exists x,\ P\ x}\exists i \qquad \dfrac{\dfrac{[\neg Q\ a]^{H5} \quad [Q\ a]^{H2}}{\bot}\neg e}{\exists x,\ P\ x}\bot e}{\exists x,\ P\ x}\vee e[H4,H5]}{\exists x,\ P\ x}\exists e[H3]}{Q\ a \to \exists x,\ P\ x}\to i[H2]}{\exists x,\ (P\ x \vee \neg Q\ a) \to Q\ a \to \exists x,\ P\ x}\to i[H1]$$

This example is quite involved. For this reason we will use a simpler example in the next section (which explains the commands that one uses to build a proof in ProofWeb):

$$\dfrac{\dfrac{[A \wedge B]^{H}}{A}\wedge e_1}{A \wedge B \to A}\to i\,[H]$$

This example is the same that was also used in Chapter 2. In ProofWeb it is displayed like:

$$\dfrac{\dfrac{[A\ \wedge\ B]^{H}}{A}\wedge e_1}{A\ \wedge\ B\ \to\ A}\to i[H]$$

Next, we will explain how to write a ProofWeb script that generates this proof.

4.2 Example in ProofWeb

First let us show the exercise for this example, which is generally the starting point when doing a proof in ProofWeb:

```
Require Import ProofWeb.
Variable A B : Prop.
Theorem prop_001 : (A /\ B) -> A.
Proof.
(*! prop_proof *)
Qed.
```

It will be clear that the exercise here is to create a natural deduction proof of the statement

$$A \wedge B \to A$$

The comment that tells where the proof has to be filled in is in green. (Note that Coq will not be able to process this file as it is. It will not want to process the '`Qed.`' line, as the proof will not be finished at that point.)

Here is a solution to this exercise that Coq accepts, and that generates the proof that we just showed:

```
Require Import ProofWeb.
Variable A B : Prop.
Theorem prop_001 : (A /\ B) -> A.
Proof.
imp_i H.
con_e1 (B).
exact H.
Qed.
```

We will now explain this script line by line. The two most interesting lines in the script are the *tactics* that actually build the proof. They are shown in green.

The first line of the script

```
Require Import ProofWeb.
```

tells Coq to load the definitions needed for ProofWeb. (These are in a file `ProofWeb.v`, but the contents of that file is not interesting for users of the system.)

The second line of the script

```
Variable A B : Prop.
```

tells Coq that we want to use propositional variables A and B. In Coq everything needs to be *declared* before it can be used. For example to declare a unary predicate P and a binary predicate Q we need to write:

```
Variable P : D -> Prop.
Variable Q : D * D -> Prop.
```

Or if we want a unary function f and a binary function g:

```
Variable f : D -> D.
Variable g : D * D -> D.
```

Even for variables of the predicate logic we need this kind of declaration. Here is the declaration needed to have a proof use *free* variables x and y:

```
Variable x y : D.
```

(Variables introduced by introduction or elimination rules do not need a declaration like this.)

The domain of the logic called `D` is declared in the `ProofWeb.v` file. In that file also a single constant in this domain is declared which is called `_d`. (This constant is needed to make some of the proof commands work, but one can use it in a proof too, if one likes.)

Now the third non-empty line of the script

```
Theorem prop_001 : (A /\ B) -> A.
```

tells the system that we will be going to prove a statement. If we manage to finish this proof the name of the theorem will be `prop_001`.

The fourth non-empty line is

```
Proof.
```

It does nothing. It visually complements the '`Qed.`' at the end, that is all. If it is left out everything works just as well. (But if you remove it from the script you will not get a Solved status. ProofWeb compares a solution to the original exercise file, and if a line suddenly is missing it does not like it.)

Now we get to the lines that build the actual proof. The command in these lines are called *tactics*. This proof just contains two tactics, but ProofWeb has many. For the full list see Appendix A on page 41.

Before the first tactic is executed, the incomplete proof is just the statement that has to be proved with dots where the proof should go:

$$\begin{array}{c}\vdots\\ A \wedge B \to A\end{array}$$

We will 'grow' the proof upward by executing tactics. Clearly this statement should be proved by implication introduction. So we execute the tactic for that which is called '`imp_i`':

```
imp_i H.
```

Most of the ProofWeb tactic names consist of three letters that identify the connective according to the following table:

$\bot$	`fls`
$\top$	`tru`
$\neg$	`not`
$\wedge$	`con`
$\vee$	`dis`
$\to$	`imp`
$\forall$	`all`
$\exists$	`exi`
$=$	`equ`

then an underscore character, then an '`i`' or '`e`' to distinguish between introduction and elimination rules, and finally an optional '`1`' or '`2`' to distinguish between left and right rules, or very occasionally a prime character '`'`' for a special tactic variant. These tactics also can be selected from the Backward menu in the ProofWeb menu bar, in which case a template for the tactic will be inserted in the text by the system.

After executing the `imp_i` tactic, the incomplete proof becomes:

$$\dfrac{\begin{array}{c}\vdots\\ A\end{array}}{A \wedge B \to A}\ {\to}\mathrm{i}\ {\scriptstyle [\mathrm{H}]}$$

As you can see, the `H` argument to the tactic gave the label for the assumption that is introduced by this rule. We will be able to use this assumption in our proof as

$$[A \wedge B]^{\texttt{H}}$$

Tactics generally have *arguments*, which are labels, formulas and terms. Important!

> *Formulas and terms that are tactic arguments will need to be put in brackets.*

Else the command will give a syntax error. The templates inserted by the Backward menu already has these brackets in them.

We now insert a left conjunction elimination rule in our incomplete proof by issuing the '`con_e1`' tactic:

```
con_e1 (B).
```

This turns the incomplete proof into:

$$\dfrac{\dfrac{\vdots}{A \wedge B}}{A}\,\wedge\mathrm{e}_1 \qquad \dfrac{\dfrac{\vdots}{A \wedge B}\,\wedge\mathrm{e}_1}{A \wedge B \to A}\,{\to}\mathrm{i}\,{\scriptstyle[\texttt{H}]}$$

Now to finish our proof, we will need to tell ProofWeb that the $A \wedge B$ is one of the assumptions (the one labeled `H`). This is done by the command:

```
exact H.
```

After this tactic the proof that we built is finished, and looks like:

$$\dfrac{\dfrac{[A \wedge B]^{\texttt{H}}}{A}\,\wedge\mathrm{e}_1}{A \wedge B \to A}\,{\to}\mathrm{i}\,{\scriptstyle[\texttt{H}]}$$

We now issue the command

```
Qed.
```

to close the proof, and our script is done.

The larger proof on page 20 can be generated by a very similar script. We will not walk through that script in detail, but we will just present it here in its entirety. Try to relate the tactics in this script to the inference rules that are used in the proof:

```
Require Import ProofWeb.
Variable P Q : D -> Prop.
```

```
Variable a : D.
Theorem example :
  exi x, (P(x) \/ ~Q(a)) -> Q(a) -> exi x, P(x).
Proof.
imp_i H1.
imp_i H2.
exi_e (exi x, (P(x) \/ ~Q(a))) b H3.
exact H1.
dis_e (P(b) \/ ~Q(a)) H4 H5.
exact H3.
exi_i b.
exact H4.
fls_e.
neg_e (Q(a)).
exact H5.
exact H2.
Qed.
```

This finishes our description on how to build a natural deduction proof in Gentzen's 'tree' style using ProofWeb. We have not discussed all natural deduction tactics in detail, as they all behave in a very similar way. See Appendix A on page 41 for the full list, with for each tactic a description of their behavior.

In the ProofWeb menus there also is a menu Forward next to the Backward menu. This tactics in this menu will behave reasonably when building a Gentzen style 'tree' proof, but they are primarily intended for making it easier to build Fitch style 'box' proofs, which is the subject of the next chapter.

5 Fitch style box proofs

This chapter explains how to build natural deduction proofs using ProofWeb. It treats a style of natural deduction introduced by Frederic Fitch. We will call that style 'box proofs', to distinguish it from the 'tree proofs' which were discussed in the previous chapter. We will not explain these natural deduction proofs in detail. You will need to read a textbook for that first. ProofWeb was designed to be used next to Michael Huth & Mark Ryan's *Logic in Computer Science: Modelling and Reasoning about Systems* [3], so that is the book that we recommend to fully understand Fitch style 'box' proofs.

However, apart from minor notational differences ProofWeb can be used with any textbook teaching natural deduction in Fitch's 'box' style.

5.1 Example on paper

Here is an example of a natural deduction proof in predicate logic, in the way that it is written on paper:

1		$\exists x(P(x) \vee \neg Q(a))$	assumption
2		$Q(a)$	assumption
3	b	$P(b) \vee \neg Q(a)$	assumption
4		$P(b)$	assumption
5		$\exists x\, P(x)$	$\exists$i 4
6		$\neg Q(a)$	assumption
7		$\bot$	$\neg$e 6,2
8		$\exists x\, P(x)$	$\bot$e 7
9		$\exists x\, P(x)$	$\vee$e 3,4—5,6—8
10		$\exists x\, P(x)$	$\exists$e 1,3—9
11		$Q(a) \to \exists x\, P(x)$	$\to$i 2—10
12		$\exists x(P(x) \vee \neg Q(a)) \to Q(a) \to \exists x\, P(x)$	$\to$i 1—11

And here is the way this same proof is displayed in the ProofWeb system:

1	H1:	∃x, (P x ∨ ¬Q a)	assumption
2	H2:	Q a	assumption
	b		
3	H3:	P b ∨ ¬Q a	assumption
4	H4:	P b	assumption
5		∃x, P x	∃i 4
6	H5:	¬Q a	assumption
7		⊥	¬e 6,2
8		∃x, P x	⊥e 7
9		∃x, P x	∨e 3,4-5,6-8
10		∃x, P x	∃e 1,3-9
11		Q a → ∃x, P x	→i 2-10
12		∃x, (P x ∨ ¬Q a) → Q a → ∃x, P x	→i 1-11

As you can see, some labels have been added in green, the boxes are not closed on the right anymore (to save some space), and the two sub-boxes for the $\vee$e rule have grown together. Apart from that it is quite the same.

This example is quite involved. For this reason we will now give a simpler example for the next section (which explains the commands that one uses to build a proof in ProofWeb):

1	$A \wedge B$	assumption
2	A	$\wedge e_1$ 1
3	$A \wedge B \to A$	$\to$i 1—2

In ProofWeb it is displayed like:

1	H:	A ∧ B	assumption
2		A	∧e$_1$ 1
3		A ∧ B → A	→i 1-2

Now in different textbooks the notation for this kind of proof will vary, but whatever the precise details are, natural deduction proofs in the 'box' style will have a structure that is very similar to this.

Differences between textbooks might be that the notation for the rules might deviate in the details, and in particular that the 'subproofs' often will not be indicated by boxes but by other shapes. In [3], which we follow in ProofWeb, the subproofs are enclosed in boxes:

1		$A \wedge B$	assumption
2		A	$\wedge e_1$ 1
3		$A \wedge B \rightarrow A$	$\rightarrow$i 1—2

However, in other books just a vertical line to left is used, with a short horizontal line indicating where the assumption is separated from the rest of the subproof:

1	|	$A \wedge B$	assumption
2	|	A	$\wedge e_1$ 1
3		$A \wedge B \rightarrow A$	$\rightarrow$i 1—2

Also occasionally a third variant is used, in which only the assumption is enclosed in a box. This way of writing Fitch proofs is called 'flag style', as there now is a shape like a waving flag:

1		$A \wedge B$	assumption
2	|	A	$\wedge e_1$ 1
3		$A \wedge B \rightarrow A$	$\rightarrow$i 1—2

Currently ProofWeb only supports the first of these three variants on the notation for Fitch-style proofs.

We will now explain how to write a ProofWeb script that generates this proof.

5.2 Example in ProofWeb

First let us show the exercise that corresponds to this example, which is generally the starting point when doing a proof in ProofWeb:

```
Require Import ProofWeb.
Variable A B : Prop.
Theorem prop_001 : (A /\ B) -> A.
Proof.
(*! prop_proof *)
Qed.
```

It will be clear that the exercise here is to create a natural deduction proof of the statement

$$A \wedge B \rightarrow A$$

The comment that tells where the proof has to be filled in is colored green. (Note that Coq will not be able to process this file as it is. It will not want to process the '`Qed.`' line at the end, as the proof will not be done at that point without having filled in the proof.)

Here is a solution that Coq accepts, and that generates the proof that we just showed:

```
Require Import ProofWeb.
Variable A B : Prop.
Theorem prop_001 : (A /\ B) -> A.
Proof.
imp_i H.
con_e1 (B).
exact H.
Qed.
```

We will now explain this script line by line. The two most interesting lines in the script are the *tactics* that actually build the proof. They are shown in green.

The first line of the script

```
Require Import ProofWeb.
```

tells Coq to load the definitions needed for ProofWeb. (These are in a file `ProofWeb.v`, but that file is not interesting for users of the system.)

The second line of the script

```
Variable A B : Prop.
```

tells Coq that we want to use propositional variables A and B. In Coq everything needs to be *declared* before it can be used. For example to declare a unary predicate P and a binary predicate Q we need to write:

```
Variable P : D -> Prop.
Variable Q : D * D -> Prop.
```

Or if we want a unary function f and a binary function g:

```
Variable f : D -> D.
Variable g : D * D -> D.
```

Even for variables of the predicate logic we need this kind of declaration. Here is the declaration for being able to use *free* variables x and y in a ProofWeb proof:

```
Variable x y : D.
```

(Please note that variables introduced by introduction or elimination rules do not need a declaration like this.)

The domain of the logic called `D` is declared in the `ProofWeb.v` file. In that file also a single constant in this domain is declared called `_d`. (This constant is needed to make some of the proof commands work, but you can also make use of it in your proof scripts if you like.)

Now the third non-empty line of the script

```
Theorem prop_001 : (A /\ B) -> A.
```

tells the system that we will be going to prove a statement. The name of this statement will be `prop_001`.

The fourth non-empty line is

```
Proof.
```

It does nothing. It complements the '`Qed.`' at the end. If it is left out everything works just as well. (However, in that case you will not be able to get your exercise a Solved status. ProofWeb compares your solution to the original exercise file, and if a line is missing it does not like it.)

Now we get to the lines that build the actual proof. The command in these lines are called *tactics*. This proof just consists of three tactics, but ProofWeb has many. For the full list see Appendix B on page 46.

Before the first tactic is executed, the incomplete proof is just the statement that has to be proved with dots where the proof should go:

$$\begin{array}{ll} & \ldots \\ 1 & A \wedge B \rightarrow A \end{array}$$

The number 1 to the left is the line number of the single line in this (incomplete) proof. When we work on the proof it will change. (By the time the proof is finished it will have number 3.) On the right there will be the rule that generated this line, but this has not yet been fixed, so currently this space is empty.

Now this line will be proved by implication introduction. So we execute the tactic for that called '`imp_i`':

```
imp_i H.
```

Most of the ProofWeb tactic names consist of three letters that identify the connective according to the following table:

$\bot$	`fls`
$\top$	`tru`
$\neg$	`not`
$\wedge$	`con`
$\vee$	`dis`
$\rightarrow$	`imp`
$\forall$	`all`
$\exists$	`exi`
$=$	`equ`

then an underscore character, then an '`i`' or '`e`' to distinguish between introduction and elimination rules, and finally an optional '`1`' or '`2`' to distinguish between left and right rules, or very occasionally a prime character '`'`' for a special tactic variant. These tactics also can be selected from the Backward menu in the ProofWeb menu bar, in which case a template for the tactic will be inserted in the text.

The `H` argument to the tactic gave the label for the assumption that is introduced by this rule. Tactics generally have *arguments*, which can be labels, formulas and terms. Important!

> *Formulas and terms that are tactic arguments will need to be put in brackets.*

Else the command will give a syntax error. The templates inserted by the Backward menu already has these brackets in them.

After executing the `imp_i` tactic, the incomplete proof becomes:

1	`H:`	$A \wedge B$	assumption
		$\ldots$	
2		A	
3		$A \wedge B \rightarrow A$	$\rightarrow$i 1—2

As you can see the line number of the final line now has changed. In ProofWeb we do not use the line numbers to refer to lines in the tactics. Instead we use *labels*, which in the proofs are displayed in green. In the tactic for implication introduction the label has to be given as an argument. In this case the label of the assumption is `H`. In the ProofWeb script we will not use the number 1 to refer to this line, but the label `H`.

The statement that we next will need to prove (the *subgoal*) is the line numbered 2, directly after the dots. Of course we will use conjunction elimination to prove this. In this section we use the *backward* tactics, which means that we will not be able to use the line that already has $A \wedge B$ on it. Instead we work 'backward' from the A by giving the tactic:

```
con_e1 (B).
```

Look on page 46 to see how the `con_e1` tactic is documented in Appendix B. It needs the argument B because from the subgoal it cannot be deduced what the

other side of the conjunction was. Also this argument is a formula, so it should be put in brackets.

After the `con_e1` tactic the (still incomplete) proof has become:

1	H:	$A \wedge B$	assumption
		...	
2		$A \wedge B$	
3		A	$\wedge e_1$ 2
4		$A \wedge B \rightarrow A$	$\rightarrow$i 1—3

As you can see, the final line now has number 4, and the arguments of the $\rightarrow$i rule on that line have been renumbered appropriately.

We can now finish the proof by noting that the subgoal that we are now facing, $A \wedge B$, actually is the assumption labeled `H`. For this execute the tactic:

```
exact H.
```

You might expect this to change line 3 to a copy line (and in fact the tactic called `exact` has a synonym called `copy`):

1	H:	$A \wedge B$	assumption
2		$A \wedge B$	copy 1
3		A	$\wedge e_1$ 2
4		$A \wedge B \rightarrow A$	$\rightarrow$i 1—3

but in fact it removes the duplication altogether and gives us as the final proof:

1	H:	$A \wedge B$	assumption
2		A	$\wedge e_1$ 1
3		$A \wedge B \rightarrow A$	$\rightarrow$i 1—2

(Copy lines sometimes are retained, but only when necessary.)

We now issue the command

```
Qed.
```

to close the proof, and our script is done.

The larger proof on page 25 can be generated by a very similar script. We will not walk through that script in detail too. We will just present it here in its entirety. Try to relate the tactics for the natural deduction rules in this script to the inference rules that occur in the proof:

```
Require Import ProofWeb.

Variable P Q : D -> Prop.
Variable a : D.
```

```
Theorem example :
  exi x, (P(x) \/ ~Q(a)) -> Q(a) -> exi x, P(x).
Proof.
imp_i H1.
imp_i H2.
exi_e (exi x, (P(x) \/ ~Q(a))) b H3.
exact H1.
dis_e (P(b) \/ ~Q(a)) H4 H5.
exact H3.
exi_i b.
exact H4.
fls_e.
neg_e (Q(a)).
exact H5.
exact H2.
Qed.
```

This finishes our description on how to build a natural deduction proof in Fitch's 'box' style using only the backward tactics. This method is conceptually easy, but often a bit laborious. For this reason we will now look at the forward tactics.

5.3 Forward tactics

In the previous section we solved the exercise using a script that only used backward tactics:

```
imp_i H.
con_e1 (B).
exact H.
```

After the first line of that script the (incomplete) proof looked like:

1	H:	$A \wedge B$	assumption
		$\ldots$	
2		A	
3		$A \wedge B \rightarrow A$	$\rightarrow$i 1—2

If we had been doing the proof by hand we now just would put

$$\wedge e_1\ 1$$

on line 2 and be done with it. This also can be done in ProofWeb with the appropriate forward tactic:

```
f_con_e1 H.
```

Note that the reference to line 1 is given by the label `H`. See for the documentation of the `f_con_e1` tactic page 51 of Appendix B.

The prefix '`f_`' abbreviates 'forward'. In fact the backwards tactics also can be given with the prefix '`b_`' added in front. The forward tactics can be inserted in the script by using the Forward menu in the ProofWeb menu bar.

After the `f_con_e1` tactic the proof will be finished looking like:

1	H:	$A \wedge B$	assumption
2		A	$\wedge e_1$ 1
3		$A \wedge B \rightarrow A$	$\rightarrow$i 1—2

This means that the following shorter script is also a solution to the problem:

```
imp_i H.
f_con_e1 H.
```

In fact that is how any sensible ProofWeb user would solve it.

5.4 The `insert` tactic

There is one final tactic that we need to explain: the `insert` tactic. When one builds a proof using backward tactics, one adds lines to the proof *after* the dots. However, with the forward tactics the pattern generally is to add lines *before* the dots. To be able to do that one needs the `insert` tactic.

Suppose that we have an incomplete proof starting like:

1	H:	$A \wedge B$	
		...	

(with more lines after the dots that we will not print here.) Now suppose further that directly after line 1 we would like to insert another line

	H1:	A	$\wedge e_1$ 1

because we know that A follows from $A \wedge B$. The way to do this in ProofWeb is to execute *two* tactics, the `insert` tactic followed by a forward tactic:

```
insert H1 (A).
f_con_e1 H.
```

What will happen if you execute these two tactics, is that after the first tactic the proof looks like:

1	H:	$A \wedge B$	
		...	
2	H1:	A	
		...	

and after the second it is:

1	H:	$A \wedge B$	
2	H1:	A	$\wedge e_1$ 1
		...	

In other words: the `insert` tactic *inserts* a new line in the incomplete proof, and then the forward tactic proves it.

Note that the first tactic corresponds to the left part of the new line, while the second tactic corresponds to the right part of the line:

```
insert H1 (A). f_con_e1 H.
```

corresponds to the line

H1:	A	$\wedge e_1$ 1

Of course it is not obligatory to directly prove an 'inserted' line with just a single forward tactic. It is perfectly legal to prove it using a much more involved proof. The `insert` tactic just means: 'now first we will prove this, and then we will go on with the rest of the proof.' (A technical aside: in the logic this is called a *cut* or *detour*.)

As a larger example, the bigger proof, on page 25, can be more efficiently generated by using forward tactics:

```
Require Import ProofWeb.

Variable P Q : D -> Prop.
Variable a : D.

Theorem example :
  exi x, (P(x) \/ ~Q(a)) -> Q(a) -> exi x, P(x).
Proof.
imp_i H1.
imp_i H2.
f_exi_e H1 b H3.
f_dis_e H3 H4 H5.
f_exi_i H4.
fls_e.
f_neg_e H5 H2.
Qed.
```

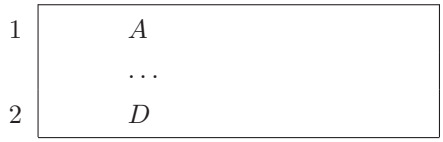
Note that in this script the proof is not generated in a fully forward way: line 7 (the one with the contradiction) is generated in a backward manner using the `fls_e` tactic. (This saves one tactic in the script.)

This finishes our description on how to build a natural deduction proof in Fitch's 'box' style using ProofWeb. We have not discussed all natural deduction tactics in detail here, as they all behave in very similar ways. See Appendix B on page 46 for the full list, with for each tactic a description of their behavior.

5.5 The order of `insert` tactics

ProofWeb has one unfortunate restriction, that is caused by the way it is implemented on top of the Coq system. Sometimes, when '`insert`'ing a line, later that line is not visible in a place where according to the (incomplete) proof it *should* be visible. This is caused by the fact that the line that was inserted only will be visible in the part of the proof that corresponds to the dots where it was inserted.

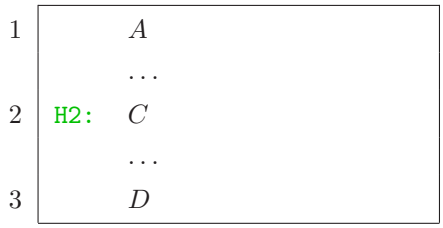
Here is a small example. Suppose we have a proof:



and we want to insert lines B and C in between the A and D. Then we can do:

```
insert H2 (C).
```

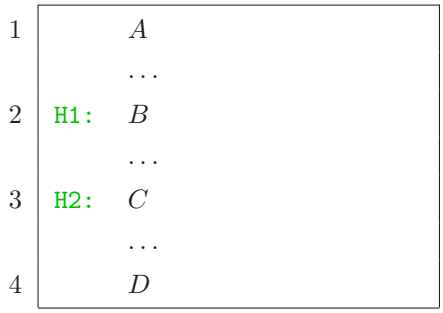
giving:



and subsequently:

```
insert H1 (B).
```

giving:



However, if we do this, in the proof the line labeled `H1` will not be available after the C. It will only be visible in the part of the proof that it was inserted in, which is the part between A and C. So when working on the part of the proof that proves D from C, one will not be able to refer to the B line. This probably will be surprising by that time.

The solution to this problem is to have the `insert` lines in the same order as their corresponding lines occur in the final proof. If one reorders the '`insert`'s:

```
insert H1 (B).
insert H2 (C).
```

there will not be a problem anymore.

Therefore, when using `insert` tactics in ProofWeb proofs, the rule of thumb should be:

> *Put the* `insert` *lines in the script in the order in which the inserted lines will appear in the proof.*

This finishes the chapter on Fitch style 'box' proofs in ProofWeb.

6 ProofWeb versus Coq

In ProofWeb one really is using the Coq system. One is processing a Coq script without any change to the Coq syntax. However, the ProofWeb tactics are not the ones that a Coq user normally would use to prove something. The Coq tactics are much more efficient, but less closely related to the natural deduction rules of first order logic. Also the ProofWeb formula syntax deviates a little bit from the way that a Coq user normally would write formulas. We will now briefly indicate what are the differences between using Coq in the ProofWeb style and using it in the Coq style.

6.1 Formula syntax

In Coq one generally does not use `all` and `exi` for the quantifiers, but instead one uses `forall` and `exists`. The reason that ProofWeb defines its own quantifiers is to get the binding strength the way it is in most logic textbooks. Also in ProofWeb one is working in an *unsorted* logic in which all variables range over the same domain, while in Coq variables have a *type*.

So in ProofWeb one writes

```
all x, P(x)
```

while a Coq user would write

```
forall x : D, P(x)
```

(To make things subtle, in Coq the typing of the variable can be omitted if the system can deduce the type from the way it is being used.) Similarly in ProofWeb one uses

```
exi x, P(x)
```

while a Coq user would write

```
exists x : D, P(x)
```

These are the only differences between ProofWeb formula syntax and the customary Coq formula syntax.

The ProofWeb tactics have been designed to also work with the 'usual' Coq quantifiers. If you prefer to have multiple sorts and weakly binding quantifiers in your formula syntax, it still is perfectly possible to use ProofWeb.

6.2 Tactics

In ProofWeb one uses many weak tactics. There generally are two for every deduction rule, a backward and a forward one. In Coq one uses fewer, but much more efficient tactics. The most important Coq tactics are:

```
intros
apply
elim
```

Also, there are a couple of Coq tactics that really are just synonyms of the ProofWeb tactics (but without the systematic naming):

```
split
left
right
exists
```

We will not explain the customary Coq tactics here. Read the Coq manual [5] or the *Coq'Art* book by Yves Bertot [1] if you want to learn about this. Here we will just present traditional Coq proofs for our two examples:

```
Variable A B : Prop.
Theorem prop_001 : A /\ B -> A.
Proof.
intros H.
elim H.
intros H1 H2.
apply H1.
Qed.
```

and:

```
Variable D : Set.
Variable P Q : D -> Prop.
Variable a : D.
Theorem example :
  (exists x : D, P(x) \/ ~Q(a)) -> Q(a) ->
  exists x : D, P(x).
Proof.
intros H1 H2.
elim H1.
intros b H3.
elim H3.
intros H4.
exists b.
apply H4.
intros H5.
elim H5.
```

```
apply H2.
Qed.
```

Of course an actual Coq user would probably not just solve these proofs using these basic tactics, but instead would try to speed things up by using automated tactics like `auto`, `tauto` or `intuition`. These are tactics that search for these small proofs themselves. In fact these tactics will find the first proof, but not the second. Of course if one uses real heavy tactics like `jprover` (which can prove any formula of first order predicate logic, given enough time and space), it will solve the second one immediately as well.

7 ProofWeb versus Huth & Ryan

ProofWeb was designed to follow as closely as possible the conventions of Michael Huth & Mark Ryan's *Logic in Computer Science: Modelling and Reasoning about Systems* [3]. However, due to the use of the Coq system, and because of the design decision to have the users work with undiluted Coq input (without preprocessing the script in the ProofWeb system first), a few slight deviations crept in. We will now briefly discuss these.

7.1 Formula syntax

The ProofWeb quantifiers have a *comma*. This is not customary in logic textbooks. There either they have nothing, or they have a period.

The formula

$$\forall x\, P(x)$$

is written in ProofWeb syntax as

```
all x, P(x)
```

and the system will print the formula back at you as

```
all x, P x
```

so without the brackets around the variable. Also, in the proofs displayed in the window pane at the lower right the comma will be printed:

```
∀x, P x
```

We could have used a notation without commas, but we had two reasons for retaining the comma. First, we think that

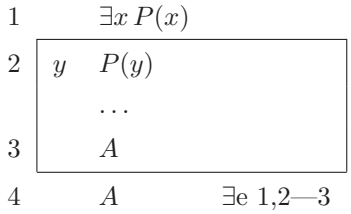
```
all x P x
```

looks confusing (to us it looks like the `P` is a relation symbol between the two `x`'s). Second, having the comma makes it easier for users to switch to the customary Coq notation for quantifiers later, if they decide to go that way.

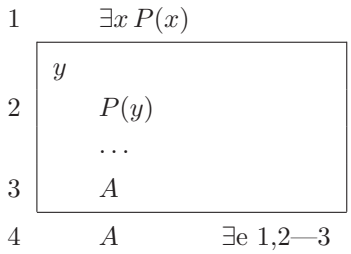
7.2 Proof display

The main difference between the proofs on paper and the proofs on the screen is the presence of Coq labels. We put them there to help writing the proof scripts. To emphasize that they are 'extra', they are in green.

Another slight difference between the proofs on paper and the proofs on the screen is the placing of variables in the ∀i and ∃e rules. We push them one line up. For instance, in the book the ∃e rule is written like:



while in ProofWeb it is:



A third deviation from the book is that we write the rules for the quantifiers in the style

∀i

while the book writes

∀ x i

A final difference between the book and the system is that in the disjunction elimination rule we run the sub-boxes together. (See the example on page 26.)

8 Outlook

With ProofWeb we have created a system for teaching and practicing natural deduction in first order logic, both for propositional and for predicate logic. We hope that it will be a useful tool for people trying to study these logics, and that it will find widespread use all over the world.

Acknowledgements

Thanks to (nobody yet) for helpful comments on these notes.

References

[1] Yves Bertot and Pierre Castéran. *Interactive Theorem Proving and Program Development, Coq'Art: The Calculus of Inductive Constructions*. EATCS Texts in Theoretical Computer Science. Springer Verlag, 2004.

[2] Georges Gonthier. A computer-checked proof of the Four Colour Theorem. URL: `http://research.microsoft.com/~gonthier/4colproof.pdf`, 2004.

[3] Michael Huth and Mark Ryan. *Logic in Computer Science: Modelling and Reasoning about Systems*. Cambridge University Press, 2nd edition, 2004.

[4] Xavier Leroy. Formal certification of a compiler back-end, or: programming a compiler with a proof assistant. In *33rd symposium Principles of Programming Languages*, pages 42–54. ACM Press, 2006.

[5] The Coq Development Team. *The Coq Proof Assistant Reference Manual*, 2007. ⟨`http://pauillac.inria.fr/coq/doc/main.html`⟩.

[6] Dirk van Dalen. *Logic and Structure*. Springer Verlag, 4th edition, 2004.

A Tactics in Gentzen style

Rules that are not intuitionistically valid are marked with a star. Rules that according to [3] are derived rules are marked with a dagger.

---

conjunction introduction

`con_i`

$$\begin{array}{c}\vdots\\ A \wedge B\end{array} \longrightarrow \frac{\begin{array}{cc}\vdots & \vdots\\ A & B\end{array}}{A \wedge B}\,\wedge\mathrm{i}$$

---

conjunction elimination left

`con_e1` B

$$\begin{array}{c}\vdots\\ A\end{array} \longrightarrow \frac{\begin{array}{c}\vdots\\ A \wedge B\end{array}}{A}\,\wedge\mathrm{e}_1$$

---

conjunction elimination right

`con_e2` A

$$\begin{array}{c}\vdots\\ B\end{array} \longrightarrow \frac{\begin{array}{c}\vdots\\ A \wedge B\end{array}}{B}\,\wedge\mathrm{e}_2$$

---

disjunction introduction left

`dis_i1`

$$\begin{array}{c}\vdots\\ A \vee B\end{array} \longrightarrow \frac{\begin{array}{c}\vdots\\ A\end{array}}{A \vee B}\,\vee\mathrm{i}_1$$

---

disjunction introduction right

`dis_i2`

$$\begin{array}{c}\vdots\\ A \vee B\end{array} \longrightarrow \frac{\begin{array}{c}\vdots\\ B\end{array}}{A \vee B}\,\vee\mathrm{i}_2$$

---

disjunction elimination

`dis_e (A \/ B) H1 H2`

$$\begin{array}{c}\vdots\\ C\end{array} \longrightarrow \dfrac{\begin{array}{ccc}\vdots & [A]^{\texttt{H1}} & [B]^{\texttt{H2}}\\ A \vee B & \vdots & \vdots\\ & C & C\end{array}}{C}\ \vee\text{e}\ {\scriptstyle [\texttt{H1},\texttt{H2}]}$$

implication introduction

`imp_i H`

$$\begin{array}{c}\vdots\\ A \to B\end{array} \longrightarrow \dfrac{\begin{array}{c}[A]^{\texttt{H}}\\ \vdots\\ B\end{array}}{A \to B}\ \to\text{i}\ {\scriptstyle [\texttt{H}]}$$

implication elimination

`imp_e A`

$$\begin{array}{c}\vdots\\ B\end{array} \longrightarrow \dfrac{\begin{array}{cc}\vdots & \vdots\\ A \to B & A\end{array}}{B}\ \to\text{e}$$

negation introduction

`neg_i H`

$$\begin{array}{c}\vdots\\ \neg A\end{array} \longrightarrow \dfrac{\begin{array}{c}[A]^{\texttt{H}}\\ \vdots\\ \bot\end{array}}{\neg A}\ \neg\text{i}\ {\scriptstyle [\texttt{H}]}$$

negation elimination

`neg_e A`

$$\begin{array}{c}\vdots\\ \bot\end{array} \longrightarrow \dfrac{\begin{array}{cc}\vdots & \vdots\\ \neg A & A\end{array}}{B}\ \neg\text{e}$$

falsum elimination

`fls_e`

$$\begin{array}{c}\vdots\\ A\end{array} \longrightarrow \dfrac{\begin{array}{c}\vdots\\ \bot\end{array}}{A}\ \bot\text{e}$$

truth introduction

`tru_i`

$$\begin{array}{c}\vdots\\ \top\end{array} \longrightarrow \frac{}{\top}\,\top\mathrm{i}$$

double negation introduction†

`negneg_i`

$$\begin{array}{c}\vdots\\ \neg\neg A\end{array} \longrightarrow \frac{\begin{array}{c}\vdots\\ A\end{array}}{\neg\neg A}\,\neg\neg\mathrm{i}$$

*double negation elimination**

`negneg_e`

$$\begin{array}{c}\vdots\\ A\end{array} \longrightarrow \frac{\begin{array}{c}\vdots\\ \neg\neg A\end{array}}{A}\,\neg\neg\mathrm{e}$$

*law of excluded middle**†

`LEM`

$$\begin{array}{c}\vdots\\ A \vee \neg A\end{array} \longrightarrow \frac{}{A \vee \neg A}\,\mathrm{LEM}$$

*proof by contradiction**†

`PBC H`

$$\begin{array}{c}\vdots\\ A\end{array} \longrightarrow \frac{\begin{array}{c}[\neg A]^{\mathtt{H}}\\ \vdots\\ \bot\end{array}}{A}\,\mathrm{PBC}\,{}_{[\mathtt{H}]}$$

modus tollens†

`MT` B

$$\begin{array}{c}\vdots\\ \neg A\end{array} \longrightarrow \frac{\begin{array}{cc}\vdots & \vdots\\ A \to B & \neg B\end{array}}{\neg A}\,\mathrm{MT}$$

universal introduction

`all_i` y

$$\begin{array}{c}\vdots\\ \forall x\, A\end{array} \longrightarrow \frac{\begin{array}{c}\vdots\\ A[y/x]\end{array}}{\forall x\, A}\,\forall\mathrm{i}$$

universal elimination

`all_e (all x,` A`)`

$$\begin{array}{c}\vdots\\ A[t/x]\end{array} \longrightarrow \frac{\begin{array}{c}\vdots\\ \forall x\, A\end{array}}{A[t/x]}\,\forall\mathrm{e}$$

existential introduction

`exi_i` t

$$\begin{array}{c}\vdots\\ \exists x\, A\end{array} \longrightarrow \frac{\begin{array}{c}\vdots\\ A[t/x]\end{array}}{\exists x\, A}\,\exists\mathrm{i}$$

existential elimination

`exi_e (exi x,` A`)` y `H`

$$\begin{array}{c}\vdots\\ B\end{array} \longrightarrow \frac{\begin{array}{c}\vdots\\ \exists x\, A\end{array} \qquad \begin{array}{c}[A[y/x]]^{\mathtt{H}}\\ \vdots\\ B\end{array}}{B}\,\exists\mathrm{e}\,[\mathtt{H}]$$

equality introduction

`equ_i`

$$\begin{array}{c}\vdots\\ t = t\end{array} \longrightarrow \frac{}{t = t}\,=\!\mathrm{i}$$

equality elimination, simple version

`equ_e (`t_1 `=` t_2`)`

$$\begin{array}{c}\vdots\\ A[t_2/x]\end{array} \longrightarrow \frac{\begin{array}{c}\vdots\\ t_1 = t_2\end{array} \qquad \begin{array}{c}\vdots\\ A[t_1/x]\end{array}}{A[t_2/x]}\,=\!\mathrm{e}$$

equality elimination, general version (t_2 *may occur in* A)

`equ_e' (`t_1 `=` t_2`) (fun x =>` A`)`

$$\begin{array}{c} \vdots \\ A[t_2/x] \end{array} \quad \longrightarrow \quad \dfrac{\begin{array}{cc} \vdots & \vdots \\ t_1 = t_2 & A[t_1/x] \end{array}}{A[t_2/x]} =\mathrm{e}$$

B Tactics in Fitch style

The green '`H:`' labels that occur in these descriptions are not part of the way proofs are written in Huth & Ryan, but are necessary for working in ProofWeb. They are the symbolic equivalents (which stay the same throughout the proof process) of the line numbers (which change all the time).

B.1 Structural tactics

`exact H`

m	`H:`	A		m	`H:`	A	
		$\dots$					
n		A	$\longrightarrow$	n		A	copy m

`insert H` B

						$\dots$	
				n	`H:`	B	
		$\dots$				$\dots$	
n		A	$\longrightarrow$	$n+1$		A	

B.2 Backward tactics

The tactic names may be prefixed with `b_...` to contrast them to the corresponding forward tactics.

Rules that are not intuitionistically valid are marked with a star. Rules that according to Huth & Ryan are derived rules are marked with a dagger.

---

conjunction introduction

`con_i`

				$\dots$	
			n	A	
				$\dots$	
	$\dots$		$n+1$	B	
n	$A \wedge B$	$\longrightarrow$	$n+2$	$A \wedge B$	$\wedge\mathrm{i}\ n, (n+1)$

---

conjunction elimination left

`con_e1` B

				$\dots$	
	$\dots$		n	$A \wedge B$	
n	A	$\longrightarrow$	$n+1$	A	$\wedge\mathrm{e}_1\ n$

---

conjunction elimination right

`con_e2` A

				$\dots$	
	$\dots$		n	$A \wedge B$	
n	B	$\longrightarrow$	$n+1$	B	$\wedge\mathrm{e}_2\ n$

disjunction introduction left

`dis_i1`

				$\dots$	
	$\dots$		n	A	
n	$A \vee B$	$\longrightarrow$	$n+1$	$A \vee B$	$\vee\mathrm{i}_1\ n$

disjunction introduction right

`dis_i2`

				$\dots$	
	$\dots$		n	B	
n	$A \vee B$	$\longrightarrow$	$n+1$	$A \vee B$	$\vee\mathrm{i}_1\ n$

disjunction elimination

`dis_e (`A `\/` B`) H1 H2`

					$\dots$	
			n		$A \vee B$	
			$n+1$	`H1:`	A	assumption
					$\dots$	
			$n+2$		C	
			$n+3$	`H2:`	B	assumption
					$\dots$	
	$\dots$		$n+4$		C	
n	C	$\longrightarrow$	$n+5$		C	$\vee\mathrm{e}\ n,(n+1)\text{—}(n+2),(n+3)\text{—}(n+4)$

implication introduction

`imp_i H`

			n	`H:`	A	assumption
					$\dots$	
	$\dots$		$n+1$		B	
n	$A \to B$	$\longrightarrow$	$n+2$		$A \to B$	$\to\mathrm{i}\ n\text{—}(n+1)$

implication elimination

`imp_e` A

				$\vdots$	
			n	$A \to B$	
				$\vdots$	
	$\ldots$		$n+1$	A	
n	B	$\longrightarrow$	$n+2$	B	$\to$e $n, (n+1)$

negation introduction

`neg_i H`

			n	`H:`	A	assumption
					$\vdots$	
	$\ldots$		$n+1$		$\bot$	
n	$\neg A$	$\longrightarrow$	$n+2$		$\neg A$	$\neg$i n—$(n+1)$

negation elimination

`neg_e` A

				$\vdots$	
			n	$\neg A$	
				$\vdots$	
	$\ldots$		$n+1$	A	
n	$\bot$	$\longrightarrow$	$n+2$	$\bot$	$\neg$e $n, (n+1)$

falsum elimination

`fls_e`

				$\vdots$	
	$\ldots$		n	$\bot$	
n	A	$\longrightarrow$	$n+1$	A	$\bot$e n

truth introduction

`tru_i`

	$\ldots$				
n	$\top$	$\longrightarrow$	n	$\top$	$\top$i n

double negation introduction[†]

`negneg_i`

				$\vdots$	
	$\ldots$		n	A	
n	$\neg\neg A$	$\longrightarrow$	$n+1$	$\neg\neg A$	$\neg\neg$i n

*double negation elimination**

`negneg_e`

				$\vdots$	
	$\dots$		n	$\neg\neg A$	
n	A	$\longrightarrow$	$n+1$	A	$\neg\neg$e n

*law of excluded middle**†

`LEM`

	$\dots$				
n	$A \vee \neg A$	$\longrightarrow$	n	$A \vee \neg A$	LEM

*proof by contradiction**†

`PBC H`

			n	`H:`	$\neg A$	assumption
					$\vdots$	
	$\dots$		$n+1$		$\bot$	
n	A	$\longrightarrow$	$n+2$		A	PBC n—$(n+1)$

modus tollens†

`MT` B

				$\vdots$	
			n	$A \to B$	
				$\vdots$	
	$\dots$		$n+1$	$\neg B$	
n	$\neg A$	$\longrightarrow$	$n+2$	$\neg A$	MT $n, (n+1)$

universal introduction

`all_i` y

				y		
					$\vdots$	
	$\dots$		n		$A[y/x]$	
n	$\forall x\, A$	$\longrightarrow$	$n+1$		$\forall x\, A$	$\forall$i n—n

universal elimination

`all_e (all x,` A`)`

				$\vdots$	
	$\dots$		n	$\forall x\, A$	
n	$A[t/x]$	$\longrightarrow$	$n+1$	$A[t/x]$	$\forall$e n

existential introduction

`exi_i` t

				$\dots$	
	$\dots$		n	$A[t/x]$	
n	$\exists x\, A$	$\longrightarrow$	$n+1$	$\exists x\, A$	$\exists$i n

existential elimination

`exi_e (exi x,` A`)` y `H`

				$\dots$	
			n	$\exists x\, A$	
				y	
			$n+1$	`H:` $A[y/x]$	
				$\dots$	
	$\dots$		$n+2$	B	
n	B	$\longrightarrow$	$n+3$	B	$\exists$e n,$(n+1)$—$(n+2)$

equality introduction

`equ_i`

	$\dots$				
n	$t = t$	$\longrightarrow$	n	$t = t$	=i

equality elimination, simple version

`equ_e (`t_1 `=` t_2`)`

				$\dots$	
			n	$t_1 = t_2$	
				$\dots$	
	$\dots$		$n+1$	$A[t_1/x]$	
n	$A[t_2/x]$	$\longrightarrow$	$n+2$	$A[t_2/x]$	=e $n, (n+1)$

equality elimination, general version (t_2 may occur in A)

`equ_e' (`t_1 `=` t_2`) (fun x =>` A`)`

				$\dots$	
	$\dots$		n	$t_1 = t_2$	
				$\dots$	
	$\dots$		$n+1$	$A[t_1/x]$	
n	$A[t_2/x]$	$\longrightarrow$	$n+2$	$A[t_2/x]$	=e $n, (n+1)$

B.3 Forward versus backward tactics

conjunction introduction

`f_con_i H1 H2`

m_1	H1:	A		m_1	H1:	A	
m_2	H2:	B		m_2	H2:	B	
		...					
n		$A \wedge B$	$\longrightarrow$	n		$A \wedge B$	$\wedge\mathrm{i}\ m_1, m_2$

conjunction elimination left

`f_con_e1 H`

m	H:	$A \wedge B$		m	H:	$A \wedge B$	
		...					
n		A	$\longrightarrow$	n		A	$\wedge\mathrm{e}_1\ m$

conjunction elimination right

`f_con_e2 H`

m	H:	$A \wedge B$		m	H:	$A \wedge B$	
		...					
n		B	$\longrightarrow$	n		B	$\wedge\mathrm{e}_2\ m$

disjunction introduction left

`f_dis_i1 H`

m	H:	A		m	H:	A	
		...					
n		$A \vee B$	$\longrightarrow$	n		$A \vee B$	$\vee\mathrm{i}_1\ m$

disjunction introduction right

`f_dis_i2 H`

m	H:	B		m	H:	B	
		...					
n		$A \vee B$	$\longrightarrow$	n		$A \vee B$	$\vee\mathrm{i}_2\ m$

disjunction elimination

`f_dis_e H H1 H2`

m	H:	$A \vee B$		m	H:	$A \vee B$		
				n	H1:	A	assumption	
						$\dots$		
				$n+1$		C		
				$n+2$	H2:	B	assumption	
						$\dots$		
		$\dots$		$n+3$		C		
n		C	$\longrightarrow$	$n+4$		C	$\vee$e m,n—$(n+1)$,$(n+2)$—$(n+3)$	

implication elimination

`f_imp_e H1 H2`

m_1	H1:	$A \to B$		m_1	H1:	$A \to B$	
m_2	H2:	A		m_2	H2:	A	
		$\dots$					
n		B	$\longrightarrow$	n		B	$\to$e m_1, m_2

negation elimination

`f_neg_e H1 H2`

m_1	H1:	$\neg A$		m_1	H1:	$\neg A$	
m_2	H2:	A		m_2	H2:	A	
		$\dots$					
n		$\bot$	$\longrightarrow$	n		$\bot$	$\neg$e m_1, m_2

falsum elimination

`f_fls_e H`

m	H:	$\bot$		m	H:	$\bot$	
		$\dots$					
n		A	$\longrightarrow$	n		A	$\bot$e m

truth introduction

`f_tru_i`

	...				
n	$\top$	$\longrightarrow$	n	$\top$	$\top$i n

double negation introduction†

`f_negneg_i H`

m	`H:`	A		m	`H:`	A	
		...					
n		$\neg\neg A$	$\longrightarrow$	n		$\neg\neg A$	$\neg\neg$i m

*double negation elimination**

`f_negneg_e H`

m	`H:`	$\neg\neg A$		m	`H:`	$\neg\neg A$	
		...					
n		A	$\longrightarrow$	n		A	$\neg\neg$e m

*law of excluded middle**†

`f_LEM`

	...				
n	$A \vee \neg A$	$\longrightarrow$	n	$A \vee \neg A$	LEM

modus tollens†

`f_MT H1 H2`

m_1	`H1:`	$A \to B$		m_1	`H1:`	$A \to B$	
m_2	`H2:`	$\neg B$		m_2	`H2:`	$\neg B$	
		...					
n		$\neg A$	$\longrightarrow$	n		$\neg A$	MT m_1, m_2

universal elimination

`f_all_e H`

m	`H:`	$\forall x\, A$		m	`H:`	$\forall x\, A$	
		...					
n		$A[t/x]$	$\longrightarrow$	n		$A[t/x]$	$\forall$e m

existential introduction

`f_exi_i H`

m	H:	$A[t/x]$		m	H:	$A[t/x]$	
		$\dots$					
n		$\exists x\, A$	$\longrightarrow$	n		$\exists x\, A$	$\exists$i m

existential elimination

`f_exi_e H` y `H1`

m	H:	$\exists x\, A$		m	H:	$\exists x\, A$	
					y		
				n	H1:	$A[y/x]$	
						$\dots$	
		$\dots$		$n+1$		B	
n		B	$\longrightarrow$	$n+2$		B	$\exists$e m,n—$(n+1)$

equality introduction

`f_equ_i`

		$\dots$					
n		$t = t$	$\longrightarrow$	n		$t = t$	=i

equality elimination

`f_equ_e H1 H2`

m_1	H1:	$t_1 = t_2$		m_1	H1:	$t_1 = t_2$	
m_2	H2:	$A[t_1/x]$		m_2	H2:	$A[t_1/x]$	
		$\dots$					
n		$A[t_2/x]$	$\longrightarrow$	n		$A[t_2/x]$	=e m_1, m_2