Swansea University
Department of Computer Science
February, 2022

CSC375/CSCM75 Logic for Computer Science

User Manual

Instructions for using a proof editor for natural deduction proofs in propositional logic and for solving the lab assignment tasks.

The system is implemented in Haskell and designed to run with interactive Haskell, ghci, which is included in the distribution of GHC (Glasgow Haskell Compiler).

In the following it is assumed that one works under Linux, and Haskell as well as the text editor Emacs are available. It is also assumed that the prover files are in the directory `prover`. How to achieve this is described in the file `README`.

Contents

1 Getting started

Move into the directory prover

```
cd prover
```

Open the emacs editor:

```
emacs&
```

(or `{xemacs&`). An Emacs reference card can be found at

```
http://www.cs.swan.ac.uk/~csulrich/tutorials/emacs-refcard.html
```

Open a shell in Emacs:

```
M-x shell
```

(`M` is the Meta-key or the `ESC`-key). Now you can run Haskell from Emacs which gives you a better input interface than the terminal.

Start interactive Haskell:

```
ghci
```

Load the file `Prover.hs`:

```
:l Prover.hs
```

Run the function main:

```
main
```

Now you can start proving:

First enter the formula you want to prove, then apply proof commands until all goals are solved.

Each command applies a proof rule of natural deduction in a backwards fashion by replacing the current goal, which must match the conclusion of the selected proof rule, by its premises which are then the new goals to be solved.

- How to enter formulas is described in Section .
- How to apply proof commands is described in Section .
- Other commands are described in section .
- How to save and view your work is described in Section .
- How to finish a session is described in Section .

2 Assessment

To assess your lab work we will ask you to show your work (see Section 6) and possibly demonstrate that you can carry out selected proofs.

Before you start a proof, make sure you have your *logic course notes* at hand.

Hint: ctrl-uparrow repeats the last user input.

When you finished a proof, the proof tree is written in the file `pproof.tex` as a latex document that uses the macro package `bussproof.sty` by Sam Buss.

You may submit your proof as solution to Task 1 by typing

```
submit 1
```

Similarly, for the other tasks.

More details about this in Section 6.

3 Syntax of formulas

Short (and incomplete but usually sufficient) overview

The usual bracketing rules for formulas apply.

For example, `A -> (B -> C)` can be written `A -> B -> C`.

Conjunction	`A and B`	or	`A & B`
Disjunction	`A or B`	or	`A | B`
Implication	`A -> B`		
Falsity	`bot`	or	`F`
Negation	`not A`	or	`A -> bot`

`not A -> B` is the same as `(not A) -> B` and the same as `(A -> bot) -> B`.

Detailed description

	Input syntax	Examples	Latex display
Atomic formulas	any letter except `F`, possibly primed and indexed	`A`, `B`, `C`, `A'`, `B2`	A, B, C, A', $B2$
Falsity	`bot`, `F`		$\bot$
And	`and`, `&`	`A and B`, `A & B`	$A \wedge B$
Or	`or`, `|`	`A or B`, `A | B`	$A \vee B$
Implies	`->`	`A -> B`	$A \rightarrow B$
Negation	`not`	`not A`, `A -> bot`	$A \rightarrow \bot$

`not A` is interpreted as `A -> bot`. Binding priorities (from strong to weak): `not`, `and`, `or`, `->`.

Implication, conjunction and disjunction associate to the right. For example,

`D -> not B -> A or B and C -> E` is the same as

`D -> ((not B) -> ((A or (B and C)) -> E))`.

4 Proof commands

Some of the commands below require an argument (for example the command `use` requires a label as argument, and the command `andel` requires a formula as argument). Such commands can also be entered without argument, in which case you will be prompted to enter it with an indication of what should be entered.

Rule	Input syntax	Explanation
Assumption	`use u`	use an available assumption labelled `u` any lower case letter can be a label
$\wedge^+$	`andi`	And-introduction backwards: $\dfrac{A \qquad B}{A \wedge B}\,\wedge^+$ The current goal must be of the form $A \wedge B$. Two new goals, A and B, are created.
$\wedge^-_{\mathrm{l}}$	`andel B`	And-elimination left backwards: $\dfrac{A \wedge B}{A}\,\wedge^-_{\mathrm{l}}$ If the goal was A, the new goal will be $A \wedge B$. The formula B is the missing part of the conjunction.
$\wedge^-_{\mathrm{r}}$	`ander A`	And-elimination right backwards: $\dfrac{A \wedge B}{B}\,\wedge^-_{\mathrm{r}}$ If the goal was B, the new goal will be $A \wedge B$.
$\to^+ u$	`impi u`	Implication-introduction backwards $\dfrac{B}{A \to B}\,\to^+ u$ The current goal must be of the form $A \to B$. The new goals is B which has the extra assumption A labelled by u (u must not have been used before)
$\to^-$	`impe A`	Implication-elimination right backwards: $\dfrac{A \to B \qquad A}{B}\,\to^-$ If the goal was B, there will be the two new goals, $A \to B$, and A. The formula A is the missing part of the implication.

Rule	Input syntax	Explanation
$\vee_l^+$	`oril`	Or-introduction left backwards: $\dfrac{A}{A \vee B}\,\vee_l^+$ The current goal must be of the form $A \vee B$. The new goals is A.
$\vee_r^+$	`orir`	Or-introduction right backwards: $\dfrac{B}{A \vee B}\,\vee_r^+$ The current goal must be of the form $A \vee B$. The new goals is B.
$\vee^-$	`ore A or B`	Or-elimination backwards: $\dfrac{A \vee B \qquad A \to C \qquad B \to C}{C}\,\vee^-$ If the goal was C, there will be the three new goals, $A \vee B$, $A \to C$, and $B \to C$. The formula $A \vee B$ is the missing disjunction.
efq	`efq`	Ex-falso-quodlibet backwards $\dfrac{\perp}{A}\,\text{efq}$ The goal can be any formula. The new goal is $\perp$.
raa	`raa`	Reductio-ad-absurdum backwards $\dfrac{(A \to \perp) \to \perp}{A}\,\text{raa}$ The goal can be any formula. The new goal is $(A \to \perp) \to \perp$, that is, the double negation of A.

5 Control commands

Input syntax	Explanation	Example
`undo`	Undo a proof step	
`quit`	Leave the prover	
`new`	Start a new proof (without saving the current proof)	
`submit <i>`	Save your proof as a solution to Task `<i>` as described in Section 6.	`submit 1`
`delete <i>`	Delete the solution to Task `<i>`	`delete 1`
?	Brief explanation of available commands	

6 Saving and viewing your work

When you finished a proof, that is, when you see the message

```
Current goal: No goal
Proof complete.
```

you can save it by entering

```
submit <i>
```

where `<i>` is the number of the task you just solved (for example 1 or 2).

To view your solution(s), go to a terminal and enter (in the directory where the prover files are)

```
pdflatex assignment
```

Then, go to the filemanager and click on the file

```
assignment.pdf
```

After finishing the next proof, you only need to repeat the `submit` command and run `pdflatex` again. The pdf will be updated automatically.

When you are working on a proof, you can view the current state of your (incomplete) proof at any time by entering in the terminal

```
pdflatex pproof
```

The file `pproof.pdf` will then contain the current state of your proof showing all open goals.

7 Finishing a session

To finish a session, type

```
quit
```

This will terminate the prover program. To leave Haskell, type

```
:q
```

8 Example of a session

The session below creates proofs of the formulas $A \to A$, $A \wedge B \to B \wedge A$, and $A \vee B \to B \vee A$. Some errors have been inserted to demonstrate how they can be corrected.

```
uli@uli-Inspiron-5502:~/teach/lectures/logic/prover22/prover$ ghci
GHCi, version 8.8.3: https://www.haskell.org/ghc/  :? for help
Loaded package environment from /home/uli/.ghc/x86_64-linux-8.8.3/environments/default
Prelude> :l Prover
[ 1 of 11] Compiling MapAux           ( MapAux.hs, interpreted )
[ 2 of 11] Compiling Parser           ( Parser.hs, interpreted )
[ 3 of 11] Compiling Formula          ( Formula.hs, interpreted )
[ 4 of 11] Compiling Occ              ( Occ.hs, interpreted )
[ 5 of 11] Compiling Perhaps          ( Perhaps.hs, interpreted )
[ 6 of 11] Compiling Proof            ( Proof.hs, interpreted )
[ 7 of 11] Compiling Buss             ( Buss.hs, interpreted )
[ 8 of 11] Compiling Step             ( Step.hs, interpreted )
[ 9 of 11] Compiling ReadShow         ( ReadShow.hs, interpreted )
[10 of 11] Compiling SystemL          ( SystemL.hs, interpreted )
[11 of 11] Compiling Prover           ( Prover.hs, interpreted )
Ok, 11 modules loaded.
*Prover> main
Enter goal formula X > A -> A

 --------------------------
Current goal: A -> A
Enter command> impi u

 --------------------------
Available assumptions
u : A
Current goal: A
Enter command> use u

 --------------------------
Current goal: No goal
```

```
Proof complete.
Enter quit, submit <i>, delete <i>, new, or ?> submit 1

 ---------------------------
Current goal: No goal
Proof complete.
Enter quit, submit <i>, delete <i>, new, or ?> new
Enter goal formula X > A & B -> B & A

 ---------------------------
Current goal: (A & B) -> (B & A)
Enter command> impi u

 ---------------------------
Available assumptions
u : A & B
Current goal: B & A
Enter command> andi

 ---------------------------
Available assumptions
u : A & B
Current goal: B
Enter command> ander

 ---------------------------
Available assumptions
u : A & B
Current goal: B
Enter missing formula X> A

 ---------------------------
Available assumptions
u : A & B
Current goal: A & B
Enter command> use u

 ---------------------------
Available assumptions
u : A & B
Current goal: A
Enter command> andel B

 ---------------------------
Available assumptions
u : A & B
Current goal: A & B
Enter command> use u
```

```
 ---------------------------
Current goal: No goal
Proof complete.
Enter quit, submit <i>, delete <i>, new, or ?> submit 2

 ---------------------------
Current goal: No goal
Proof complete.
Enter quit, submit <i>, delete <i>, new, or ?> new
Enter goal formula X > A or B -> B or A

 ---------------------------
Current goal: (A or B) -> (B or A)
Enter command> impi u

 ---------------------------
Available assumptions
u : A or B
Current goal: B or A
Enter command> ore

 ---------------------------
Available assumptions
u : A or B
Current goal: B or A
Enter missing formula of the form  X or Y> A or B

 ---------------------------
Available assumptions
u : A or B
Current goal: A or B
Enter command> use u

 ---------------------------
Available assumptions
u : A or B
Current goal: A -> (B or A)
Enter command> impi

 ---------------------------
Available assumptions
u : A or B
Current goal: A -> (B or A)
Enter assumption variable> v

 ---------------------------
Available assumptions
```

```
v : A
u : A or B
Current goal: B or A
Enter command> orir

 ---------------------------
Available assumptions
v : A
u : A or B
Current goal: A
Enter command> use u
Assumption doesn't fit or doesn't exist, try again> use v

 ---------------------------
Available assumptions
u : A or B
Current goal: B -> (B or A)
Enter command> impi w

 ---------------------------
Available assumptions
w : B
u : A or B
Current goal: B or A
Enter command> andi
Rule not applicable, try again> orir

 ---------------------------
Available assumptions
w : B
u : A or B
Current goal: A
Enter command> undo
Rule not applicable, try again> undo

 ---------------------------
Available assumptions
w : B
u : A or B
Current goal: B or A
Enter command> oril

 ---------------------------
Available assumptions
w : B
u : A or B
Current goal: B
Enter command> use w
```

```
 ---------------------------
Current goal: No goal
Proof complete.
Enter quit, submit <i>, delete <i>, new, or ?> submit 3

 ---------------------------
Current goal: No goal
Proof complete.
Enter quit, submit <i>, delete <i>, new, or ?> quit
*Prover> :q
Leaving GHCi.
uli@uli-Inspiron-5502:~/teach/lectures/logic/prover22/prover$
```

Since the proofs were commited as solutions to Tasks 1, 2, and 3, the file `assignment.pdf` contains (after running `pdflatex assignment`) all three proofs rendered as trees.

9 Contact

To report errors or make suggestions for improvements please email

`u.berger@swansea.ac.uk`