Math 502 Computer Information and Lab

Dr. Holmes

May 6, 2008

This document contains information on the computer lab we will have on Tuesday in MG122. I am hoping to do the complete intro and run through all the examples on Tuesday. Please note that there are two subsections with exercises, and the quantifier one is a question on the take-home final (which I hereby implicitly announce).

I will be holding office hours in MG122 in the afternoons or on request while this lab is outstanding. It is due at the end of the course.

1 Access to the Software

We are using Marcel, a theorem prover developed by the instructor. Marcel is a package of functions written in Standard ML, and its user interface is currently the command window for the Moscow ML implementation of this programming language. Moscow ML is free and easy to download and install; there is a link to its site on my Marcel page (link in the first paragraph of my web page to the Marcel page, then to Moscow ML).

I hasten to add that you do not need to understand Moscow ML to use Marcel. You do need to know that Moscow ML commands always end with a semicolon, you can put more than one of them on a line, and any command which does not have an argument (such as the ubiquitous 1(); and r(); commands seen below) gets a bogus “unit” argument (the empty pair of parentheses) anyway.

In MG122, look on the desktop of your machine for an icon called mosmlbatch or “shortcut to mosmlbatch”. Click that icon and you will invoke a Moscow ML command window.

In that window, type
load "marcel"; open marcel;

and the computer will load the Marcel functions.

2 A Propositional Example

At the Marcel prompt (a hyphen), type

- s "(P1->(P2->P3))==((P1&P2)->P3)";

The prover replies with

Line number 1:

|-

1: P1 -> P2 -> P3 == P1 & P2
    -> P3

The prover uses ASCII notation: our ¬, ∧, ∨, →, ≡ are ~, &, v, ->, ==

What the prover is displaying is a sequent. The format of sequents is vertical rather than horizontal: a list of numbered propositions making up the left side of the sequent is given, followed by the "turnstile" |-, followed by a list of numbered propositions on the right side. Here there is nothing on the left and just a propositional logic formula on the right. All propositional variables are of the form Pn where n is a numeral.

Notice that the prover does not display unneeded parentheses. It doesn’t need them in input, either, but of course you should type parentheses as
needed if you are uncertain what the order of operations is (and you should look at the output to see if you have made any mistakes).

In general it is better to write input for the prover in a text file you can edit and paste it into the prover window. The usual Ctrl-V does not seem to work: but you can call up a dropdown menu from the upper left hand corner of the command window and do Edit>Paste.

We exhibit half of interactive proof of this sequent.

\[ -r(); \]

The \( r() \) command invokes a right sequent rule. It automatically selects the rule appropriate to the logical form of the first statement on the “right” side of the turnstile.

**Line number 2:**

1: \( P1 \rightarrow P2 \rightarrow P3 \)

\( \vdash \)

1: \( P1 \& P2 \rightarrow P3 \)

Notice that we now have a nontrivial left side of the sequent. Here we are proving one of the two implications involved in the biconditional: the other one will show up automatically later.

\[ -r(); \]
This does exactly what we should expect. This is the natural way to prove the implication on the right side of the preceding sequent: assume the hypothesis and deduce the conclusion.
We carry out the natural unpacking of the conjunctive hypothesis, then we have a problem. The only proposition we can work on is the one in the third position on the left.

-gl 3;

Line number 5:

1: P1 -> P2 -> P3
2: P1
3: P2
|-
1: P3

There is a similar \texttt{gr n} command for moving propositions on the “right” side of the sequent to the front. The next rule (the left rule for implication) is the first branching rule we encounter. It is useful to note that the prover will immediately serve up just one of the goals created by this rule and keep the other one in the background. It will show up automatically later.
-1();

Line number 6:

1: P1
2: P2
\|--
1: P1
2: P3

This is an axiom. We tell the computer this by invoking the `Done();` command. It immediately serves up the other goal created by the branching rule above.
-Done();

Line number 7:

1: P2 -> P3
2: P1
3: P2
1-  
1: P3

This goal involves another left implication, so we will see a branch again.
-1();

Line number 8:

1:  P1
2:  P2

|- 

1:  P2
2:  P3

-gl 2; Done(); (* This is an axiom but we need to move a proposition to the front for the prover to see it *)
Line number 9:

1: P3
2: P1
3: P2

|-
1: P3

-Done(); (* this is the other goal from the last left implication rule; it is just an axiom *)

Line number 3:

1: P1 & P2 -> P3

|-
1: P1 -> P2 -> P3

The prover now presents you with the second part of the proof of the original biconditional. You have all the tools you need to complete the proof.
Try typing `Showall();` at this point, then keep hitting return until you get the prompt back. This gives you a a record of the proof, which one could in principle read, though it is rather boring.

```plaintext
- startlogging "test";
  > val it = () : unit
- LogTheProof();
  > val it = () : unit
- stoplogging();
  > val it = () : unit
```

This sequence of commands will store the proof you just paged through to the file `test.mlg` (a text file).

### 3 Some Exercises

Complete the proof you started above and save the proof to a file as indicated. Please save the commands you actually issue to the prover in a text file and turn in that and the saved proof file electronically (by email to me).

Prove another logical tautology (a reasonably complicated one) in the same way.

Try proving a statement which is not a tautology. You will get to a sequent you cannot prove: explain how this gives you an assignment of truth values which makes the original statement false.

For one of these proofs, set up the whole proof tree on paper.

### 4 Quantifier Example

As my example I use a rather bizarre statement which is beloved as a logical example but whose proof is rather mysterious, but which does allow me to bring out the main points.

```plaintext
- s "(Ex1.(Ax2.P1(x1)->P1(x2)))";
```
In Marcel, parentheses around quantified statements are mandatory. Bound variables are all of the form $x_n$ (where $n$ is a numeral). Note that the same $P_n$ used as propositional letters here can also be used as predicate letters.

The same commands $l();$; $r();$; $gl\ n,$ $gr\ n,$ and $Done();$; will be used here as above. The left and right commands also know the rules for quantifiers (which correspond closely to our intuitive approach to quantifiers from the first section). One new command will appear as soon as we use commands which require us to supply witnesses (the left universal and right existential quantifier rules).

$-r();$

Line number 2:

$|-$

1: $(Ax3.P1(U1) \rightarrow P1(x3))$

2: $(Ex1.(Ax2.P1(x1) \rightarrow P1(x2)))$
The curious variable $U_1$ requires some explanation. Here we apply the right rule for the existential quantifier, where we are allowed to introduce a witness of our choice to the existential statement. The variable $U_1$ is a placeholder for a witness: we can now or later assign it a value, and we will see this process below. Oddly, the variable $U_1$ will never be assigned a value in this particular proof.

\[-r();\]

Line number 3:

\[\begin{align*}
1: \quad & P_1(U_1) \rightarrow P_1(a_2) \\
2: \quad & (\exists x_1. (\forall x_2. P_1(x_1) \rightarrow P_1(x_2)))
\end{align*}\]

The variable $a_2$ introduced here is the arbitrary object that one expects to be introduced when we plan to prove a universal goal.
Now we are allowed to assign any value we like to $U_1$ (it represents the witness we select for the application of the right rule for the existential quantifier above). If we were to assign $a_2$ we would be done... 

```
- su 1 "a2";
```

Circularity error

...but we can’t! The problem is that $a_2$ is a new object which cannot have been mentioned in the proof before it was introduced – whatever we replace $U_1$ with cannot have any new variable (either $\exists n$'s or $\forall n$'s) which
would not be accessible at the time the variable $U1$ was introduced. This is easily managed by the fact that the variables are numerically indexed.

Line number 4:

1: $P1(U1)$

|-

1: $P1(a2)$

2: $(\exists x1.(\forall x2. P1(x1) \rightarrow P1(x2)))$

The only thing we can do here is try to apply the right existential rule again. And in fact as we will see this works. Admittedly, the structure of this particular proof is rather mysterious...
-gr 2; r();

Line number 5:

1: P1(U1)

\[\]

1: (Ax3.P1(U3) -> P1(x3))

2: (Ex1.(Ax2.P1(x1) -> P1(x2)))

3: P1(a2)

We have introduced another witness U3.

-r();

1: P1(U1)

\[\]

1: P1(U3) -> P1(a4)

2: (Ex1.(Ax2.P1(x1) -> P1(x2)))

3: P1(a2)
-r();

Line number 7:

1: P1(U3)
2: P1(U1)
|- 
1: P1(a4)
2: (Ex1.(Ax2.P1(x1) -> P1(x2)))
3: P1(a2)
(* and now we *can* make an assignment which makes this an axiom! *)

-su 3 "a2";

Line number 7:

1: P1(a2)

2: P1(U1)

|-

1: P1(a4)

2: (Ex3. (Ax4. P1(x3) -> P1(x4)))

3: P1(a2)
Line number 7:

1: P1(a2)
2: P1(U1)

|- 

1: P1(a2)
2: P1(a4)
3: (Ex3.(Ax4.P1(x3) -> P1(x4)))

> val it = () : unit

Q. E. D.

It is worth remarking that the first command, which merely reorders lines does not change the “line number” of the sequent (because these commands actually do not create a new sequent at all, just change its presentation); the Done(); command proves this goal and there are no more left to prove, and the prover simply says Q. E. D: we are done proving the original proposition.
5  More Exercises

Write a paper version of the proof above. Notice that your paper proof will not include any analogue of the Un’s: the substitutions you eventually make for the Un’s will be made immediately when the right existential rule is applied.

Complete the proof which starts as follows. This is related to an error that students can make in discrete math. It might seem that reflexivity of a relation follows from symmetry and transitivity, but it doesn’t, quite... Time permitting, I will run through a few initial steps of this proof in lab.

\[
\neg\ s\ "((Ax1.(Ex2.x1R1x2))\ &\ (Ax1.(Ax2.x1R1x2\rightarrow x2R1x1))
\ &\ (Ax1.(Ax2.(Ax3.x1R1x2\&x2R1x3\rightarrow x1R1x3))))\rightarrow(Ax1.x1R1x1)"
\]

(* ML requires that a quoted expression be all on one line; don’t really put a carriage return in the command above *)

Line number 1:

\[
|-
\]

1:

\[
(Ax1.(Ex2.x1\ R1\ x2))\ &
(Ax1.(Ax2.x1\ R1\ x2\ \rightarrow\ x2\ R1\ x1))
\ &\ (Ax1.(Ax2.(Ax3.x1\ R1\ x2
\ &\ x2\ R1\ x3\ \rightarrow\ x1\ R1\ x3))))\rightarrow(Ax1.x1\ R1\ x1)
\]

As in the first exercises, the commands you issue to the prover should be saved in a text file, and you should save a proof file using the commands I demonstrated above.
This exercise is one of the questions on the take home final. You are permitted to seek assistance from other students (whom you should acknowledge) or from me on the computer component. What you should do on your own is write a proof in English which parallels your computer proof and annotate what operations in your computer proof correspond to what moves in your proof in mathematical English. (To achieve this it might be a better strategy to write an English proof then develop a computer proof following the English proof.)