1 Introduction

In these labs, you will use Marcel, a program written by me, to prove statements in propositional and quantifier logic. Marcel implements a style of reasoning very similar to the style I have been describing in the notes (there are minor differences which I will point out). Marcel will not prove your theorems for you, but it will not allow you to make mistakes: if you enter Marcel commands you will correctly execute proof strategies of the kind I am teaching.

In both the paper and computerized logical reasoning segments of this course, I am also learning from you about how best to teach this kind of material. If you have ideas from your perspective about how things might be done better, they might be of interest to me: in particular, I have made several changes to the software directly motivated by student experiences.

The Lab I problems are due next Friday at midnight; you turn in your work by saving log files and emailing them to me. I am not writing paper homework for this Friday; a paper assignment might appear next Monday or Wednesday due the following Monday or Wednesday, though.

2 Setting Up

If you want to install Marcel on your own computer, it is possible and I have assisted students to do it. Not a few students have managed to do it on their own. You need to install Moscow ML (there is a link on the Marcel page, to which there is a link at the top of my web page), then put the Marcel source files in the right place relative to your Moscow ML installation. It is dead
easy to do this on Windows, not too bad on Linux, and it is possible on a Mac (using the underlying Linux).

Marcel is a program written in the computer language Standard ML, of which Moscow ML is an implementation, and it runs in the Moscow ML interpreter window.

Setting up to do this on the MG104 machines involves the following steps. You are most welcome to try this before the lab on Friday and see if you can get set up, but we will be spending time on this in lab as well.

open a terminal: Click the background and select Open Terminal.

copy the Marcel source into your home directory: In the terminal window type the following command:

    cp /home/public/holmes/marcel.sml .

The standalone period at the end is not a typo! This copies the source file for Marcel into your home directory.

start the Moscow ML interpreter: Type mosml in your terminal window. You should get messages on your screen about the Moscow ML version and a new prompt (a hyphen).

It is useful to be aware that everything you type in the Moscow ML window is a Moscow ML command. Be aware that a command always ends in a semicolon. If you hit return and you haven’t finished typing a command, you can continue typing it (for example, you can supply a missing semicolon). Exception: you cannot hit return in the middle of a string. If you get into an ML error condition, just hit Ctrl-C – you will interrupt out of the error condition, and your Marcel proof will not be damaged.

Everything in ML and Marcel is case sensitive: lowercase and uppercase letters are always different.

compile Marcel: You only need to do this the first time you use Marcel, or if I give you an updated version for some reason. This has been known to happen; I have a hands-on approach to servicing software!

At the hyphen prompt, type

    compile "marcel.sml";
You will get lots of messages. When it quiets down it will have compiled Marcel.

**start Marcel:** You will need to execute these ML commands to start Marcel every time you use it.
At the hyphen prompt, type
```
load "marcel"; open marcel;
```
Notice that there are two commands here; you could issue them separately but you do have to issue them in that order.
A lot of output follows these commands; you can ignore it.
At this point you have the Marcel prover in front of you.

### 3 Marcel Notation

The notation of Marcel is not the same as the notation we use in the notes or on the board. The problem is that we need to use ASCII symbols since we are in a text interface. The symbols used are fairly natural, though.

**propositional letters:** These are of the form $P_1$, $P_2$, $P_3$, and so forth. Other letters will not work. Lowercase $p$ will not work.

**negation:** What is usually written $\neg P$ we will write $\sim P_1$ in Marcel.

**conjunction:** What is usually written $P \land Q$ we will write $P_1 \& P_2$ in Marcel.

**disjunction:** What is usually written $P \lor Q$ is written $P_1 \lor P_2$ in Marcel. The symbol is a lowercase $v$.

**implication:** What is usually written $P \rightarrow Q$ is written $P_1 \rightarrow P_2$ in Marcel. The implication arrow is a hyphen followed by a greater than sign.

**biconditional:** What is usually written $P \leftrightarrow Q$ is written $P_1 == P_2$ in Marcel.

**parentheses and order of operations:** You can use parentheses to make sure your sentence has the correct structure. Marcel uses order of operations (the same as the ones I stated in class) and when it displays
a formula it does not show parentheses that are not needed. I would encourage you to put in as many parentheses as you think are needed.

4 Marcel Commands

A Marcel command is actually a function in the ML programming language. A function with no inputs gets an input anyway (because Marcel insists): the command \texttt{r} for example just means “apply a proof strategy to the conclusion of the current argument” and needs no input, but when it is issued it gets a dummy input (a pair of parentheses) so we type

\texttt{r();}

to issue this command (remember that an ML line always ends with a semicolon).

Inputs to other Marcel commands are sometimes strings (all logical expressions are treated as strings) which need to be enclosed in quotes, and sometimes integers, which are just entered as usual (without quotes).

The very first command we show you is \texttt{start} or \texttt{s}, which takes a single quoted logical expression as input and sets up the environment to prove that expression.

\begin{verbatim}
- s "P1 -> P1";
Line number 1:

|-

1: P1 -> P1

> val it = () : unit
\end{verbatim}

This is how the environment looks when we start proving the theorem P1 \rightarrow P1. \texttt{s "P1 -> P1";} is the command we entered; the rest is the computer’s response. The line \texttt{> val it = () : unit} you can ignore completely; it is just ML interpreter chatter.
Each line of the proof shows us a statement we want to prove (our goal) below the symbol |- and a list of statements we are locally allowed to assume (use) above this symbol. Right now we have no assumptions at all.

- r();

**Line number 2:**

1: P1

|-

1: P1

I should mention that we are writing vertically a notation (called sequent notation) which is usually written horizontally: the assumptions would normally be to the left of the |- and the conclusion to the right, and this motivates our nomenclature.

The command r(); which we issue here abbreviates “right”: its effect is to apply a proof strategy to the conclusion we are trying to prove (which is to the right of the |-, which is often called the “turnstile”). The r command is smart: it looks at the form of the statement and applies the correct strategy, in this case the strategy for proving an implication: assume the hypothesis of the implication and make the conclusion of the implication the new goal. You can see what happened here.

You can also see that we are done: we are trying to prove P1 and P1 is one of our assumptions. We tell the computer this.

- Done();

Q. E. D.

The Done(); command asks the computer to check whether the first assumption is the same as the goal. If it is, the program either reports that
we are finished with the classical formula Q. E. D. (as it does here) or else serves up the next thing we need to prove.

The computer does not draw conclusions on its own: you need to point out to it that you are done. It checks and remarks that indeed, you are done (or issues an error message if you aren’t!)

## 5 A Long Example

We are going to prove

\(((P \lor Q) \rightarrow R) \leftrightarrow (P \rightarrow R) \land (Q \rightarrow R)\).

We proved one direction of this in the notes, and you were assigned the proof of the other direction in the homework.

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

Line number 1:

|-  
1:  P1 v P2 -> P3 ==  
   (P1 -> P3) & (P2 -> P3)

Notice that we entered the statement to be proved with lots of parentheses, but the Marcel display function decided that most of them were not needed.

- r();

Line number 2:

1:  P1 v P2 -> P3

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

The \texttt{r} command applies a proof strategy to the goal we are trying to prove. It is a biconditional, so the proof is divided into two parts, proving the implications in each direction. When Marcel has multiple goals to prove, it chooses one to serve up to you; when you are done proving that one it will give you the next one. There are commands which will let you look at the other goals:

- \texttt{sg();}

Line number 3:
1: (P1 \rightarrow P3) & (P2 \rightarrow P3)

\texttt{|-}

1: P1 v P2 \rightarrow P3

- \texttt{sg();}

Line number 2:
1: P1 v P2 \rightarrow P3

\texttt{|-}

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

The \texttt{sg} command switches between subproofs just introduced (as in this case): there is another command \texttt{ng} which will page through all current subproofs. The only reason we did this here is to show that the other proof is “out there”; you will see it reappear at the appropriate moment. We switch back and continue the proof in the usual order (but we could have switched,
done the proof of the other part, and Marcel would then have served up the proof of this part to do next).

We are going to apply a strategy to the conclusion next. To prove a conjunction, we need to prove each part of the conjunction separately, so we expect the proof to break into parts again.

- \( r() \);

Line number 4:

1: \( P_1 \lor P_2 \to P_3 \)

|-

1: \( P_1 \to P_3 \)

The other goal \( P_2 \to P_3 \) will appear again! We have an implication as our goal so of course we will apply \( r \) again.

- \( r() \);

Line number 6:

1: \( P_1 \)

2: \( P_1 \lor P_2 \to P_3 \)

|-

1: \( P_3 \)

We were trying to prove \( P_1 \to P_3 \) on the previous step, so now we add \( P_1 \) to our assumptions and adopt \( P_3 \) as our new goal.

Now we have the problem that we can’t do anything with our conclusion or with the first assumption (which is what the \( l() \); (“left”) command for
handling assumptions works on). So we use a command to bring the second assumption into the first position.

- gl 2;

Line number 6:
1: P1 v P2 -> P3
2: P1

|-
1: P3

- gl abbreviates “get left” (there is a gr command but it is only used in a different mode of the prover where we may have more than one conclusion).

Now we can make our first application of the 1(); command (apply a proof strategy to the first assumption). The first assumption is an implication here: Marcel’s strategy here is related to modus ponens. Marcel first attempts to prove the hypothesis of the implication as a new goal using the other assumptions (and also using the negation of the original goal, a technical thing); then it attempts to prove the original goal using the conclusion of the implication as a new hypothesis. The left rule for implication is probably the proof strategy of Marcel that causes more head scratching than any other, though I actually have talked about it (when we have an implication as a hypothesis, we are tempted to prove its hypothesis as a goal).

- 1();

Line number 7:
1: P1
2: ¬P3
We are now trying to prove the hypothesis $P_1 \lor P_2$ of the implication (so that we can introduce the conclusion of the implication as an assumption by *modus ponens* in the next part). The fact that we assume the negation of the original goal might be surprising; do notice that when $C$ is your goal it is actually always safe to assume $\neg C$ (take a pessimistic view) because $(\neg C \rightarrow C) \rightarrow C$ is true. In any case we will usually not use the additional assumption.

- `r();`

Line number 9:

1: $P_1$
2: $\neg P_3$
3: $\neg P_2$

1: $P_1$

When Marcel sees a goal of the form $P \lor Q$, it always assumes $\neq Q$ and adopts $P$ as the new goal. Notice that we are done with this subproof: our goal is the same as one of our assumptions.

- `Done();`
When we type `Done();` (and the computer agrees we are right) we will either be done with the proof or jump to some unfinished business. It is early days yet in this proof and we have lots of unfinished business. This is the second part of the application of the left rule to \((P1 \lor P2) \rightarrow P3\) above: in the second part we get to introduce the conclusion \(P3\) of the implication as a new assumption (basically because we have proved it by *modus ponens*), and in this case we are immediately done with this subproof because \(P3\) is also our original goal!

We return to some older unfinished business (the other half of the proof of the conjunction in line 2). The proof here is going to look very much like the proof of the first part of the conjunction: I’m going to include it here but without as much blow-by-blow commentary.
- r();

Line number 10:
1: P2
2: P1 v P2 -> P3
|- 
1: P3
- gl 2;

Line number 10:
1: P1 v P2 -> P3
2: P2
|- 
1: P3
- l();

Line number 11:
1: P2
2: ~P3
|- 
1: P1 v P2
- r();
Something different happens here because Marcel insists on one form of the proof for a disjunction: instead of getting P2 in both assumptions and conclusion, we get contradictory assumptions of P2 and its negation. This is valid of course, as a contradiction implies anything, but will take a different strategy.
We bring the negative assumption to the front, then apply the left rule for a negation, which is a form of the contrapositive: in the presence of the other assumptions, instead of trying to prove \( P_1 \) from the negation of \( P_2 \), we are now trying to prove \( P_2 \) from the negation of \( P_1 \).

- Done();

Line number 12:

1: \( P_3 \)
2: \( P_2 \)
|- 
1: \( P_3 \)
- Done();

Line number 3:

1: \((P_1 \to P_3) \& (P_2 \to P_3)\)
|-
We are now done with the proof of the first implication in the main biconditional and Marcel serves us the second one. We will begin by obvious unpacking: an application of \( r \) to the implication in the conclusion and of \( l \) to the conjunction which is the first assumption will have obvious effects, identical to what we would do on paper.

- \( r() \);

Line number 15:

1: \( P_1 \lor P_2 \rightarrow P_3 \)

2: \( (P_1 \rightarrow P_3) \land (P_2 \rightarrow P_3) \)

\([\leadsto]\)

1: \( P_3 \)

- \( gl \ 2; \)

Line number 15:

1: \( (P_1 \rightarrow P_3) \land (P_2 \rightarrow P_3) \)

2: \( P_1 \lor P_2 \)

\([\leadsto]\)

1: \( P_3 \)

- \( l(); \)

Line number 16:
I remark that I was wondering when I did this whether I wanted to do the left rule or the right rule first. I made the wrong choice; I would not have needed `gl` commands if I had done them in the other order. But the proof still works fine.

We will now apply the left rule for a disjunction, which is always proof by cases in Marcel.

- `l();`

I remark that I was wondering when I did this whether I wanted to do the left rule or the right rule first. I made the wrong choice; I would not have needed `gl` commands if I had done them in the other order. But the proof still works fine.

We will now apply the left rule for a disjunction, which is always proof by cases in Marcel.

- `l();`

I remark that I was wondering when I did this whether I wanted to do the left rule or the right rule first. I made the wrong choice; I would not have needed `gl` commands if I had done them in the other order. But the proof still works fine.

We will now apply the left rule for a disjunction, which is always proof by cases in Marcel.

- `l();`

I remark that I was wondering when I did this whether I wanted to do the left rule or the right rule first. I made the wrong choice; I would not have needed `gl` commands if I had done them in the other order. But the proof still works fine.

We will now apply the left rule for a disjunction, which is always proof by cases in Marcel.

- `l();`
The assumption \( P_1 \lor P_2 \) is replaced here by \( P_1 \); later we will see another goal just like this but with \( P_1 \lor P_2 \) replaced by \( P_2 \).

Now we are really looking at application of **modus ponens**, and we can see how our left rule for implication supports it. On paper we would deduce the goal \( P_3 \) from assumption 1 and assumption 2 by modus ponens, so we will apply the left rule of implication to assumption 2.

```
- gl 2; l();
```

**Line number 17:**

1: \( P_1 \rightarrow P_3 \)
2: \( P_2 \rightarrow P_3 \)
3: \( P_1 \)

\[-\]

1: \( P_3 \)

**Line number 19:**

1: \( P_2 \rightarrow P_3 \)
2: \( P_1 \)
As we remarked earlier, we are allowed to type two commands on a line. Our aim in the first part of the left rule of implication is to prove the hypothesis of the implication as a new goal, but we already have it as an assumption (we are really in the *modus ponens* situation!)

- gl 2; Done();

Line number 19:

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

Line number 20:

1: P3
2: P2 -> P3
3: P1
The conclusion of the implication is (as we already knew) our goal, so we are done with this bit.

- Done();

We are now in the second part of the proof by cases. It is very much like the first; we display it but without comments.

- gl 3; l();

We are now in the second part of the proof by cases. It is very much like the first; we display it but without comments.

- gl 3; l();
\[-
1: P3
\]

Line number 21:

1: P2
2: P1 \rightarrow P3
3: \neg P3

\[-
1: P2
- \text{Done();}
\]

Line number 22:

1: P3
2: P2
3: P1 \rightarrow P3

\[-
1: P3
- \text{Done();}
\]

Q. E. D.
6 Saving Your Work

Before you start working, type `startlogging "<filename>";`. This opens a log file named `<filename>.mlg` (don’t really use this name) into which your Marcel session will write the commands you type and other useful information.

When you are done proving a theorem, type `LogTheProof();` and it will save a somewhat readable English proof as a comment in your log file. You can see this same proof of a theorem on the screen by typing `showall();` and hitting Enter repeatedly.

When you are done working, type `stoplogging();` and it will close your log file.

You will turn in your assignments by mailing me the log files.

7 Lab I Problems

1. Prove both forms of deMorgan’s law using Marcel.

   I will make an observation during lab time which will help orient you in proofs of negations: Marcel’s strategy for proving \( \neg P \) is not to assume \( P \) and prove a contradiction, but to assume \( P \) and prove \( \neg P \); this does work! (use Marcel to prove that \((P \rightarrow \neg P) \rightarrow \neg P \)).

2. Prove problem 6 from the first homework using Marcel.

3. Prove

   \[
   (P \rightarrow (Q \rightarrow R)) \leftrightarrow ((P \land Q) \rightarrow R)
   \]

   using Marcel. This is a familiar theorem (our first example) and you should be able to see how the proof exactly follows what we did.

4. Prove

   \[
   ((P \rightarrow Q) \rightarrow R) \leftrightarrow ((\neg R \rightarrow P) \land (Q \rightarrow R))
   \]

   using Marcel.

   This is a verification of Marcel’s left rule for implication; you can see the strange assumption \( \neg R \) in the first part. It isn’t important to understand this to do the problem, just do the Marcel proof.