Marcel Theorem Prover Lab

Dr. Holmes

November 30, 2022

1 Introduction to Marcel

This is the lab manual with the first (and probably only) computer lab to be done with the Marcel prover in Math 189, Fall 2022.

Marcel is a computer program designed to manipulate arguments.

An argument is made up of a list of premises and a conclusion. An argument is valid iff any assignment of values to variables which makes all the premises true will also make the conclusion true. Many of our logical rules are presented exactly as small valid arguments.

Marcel is designed to try to prove that arguments are valid. Its basic commands transform a given argument into a single argument or a list of arguments whose validity is equivalent to the validity of the original argument. Marcel then displays each of these arguments in turn and invites the user to show that the argument is valid. When it is done showing that one argument is valid, it will remember and display the next one that needs to be shown to be valid.

The list of commands that we use in the first lab is extremely short:

s: The **s** command has a single string as input, Marcel notation for a theorem to be proved. An argument is constructed and displayed with no premises and the intended theorem as conclusion.

r: The **r** (for **Right**) command takes no inputs: this command transforms the argument we are working on by simplifying the conclusion in a way appropriate to its logical form (possibly producing two arguments to verify instead of one).

l: The **l** (for **Left**) command takes no inputs. This command transforms the argument we are working on by simplifying the first premise in a way appropriate to its logical form (possibly producing two arguments to verify instead of one).

gl: The **gl** (for **Getleft**) command takes a number as its input: the number indicates which of the premises to bring to the front of the premise list (so as to be able to apply **l**).

d: The **d** (for **Done**) command takes no inputs. When the first premise and the conclusion are the same, use of the **d** command causes Marcel to record the current argument as valid, and the next one that we need to verify will be served up (or Marcel will say QED and report that the current theorem is proved!)

For propositional logic proofs, these are the only commands that are needed!

There are some other useful commands. The final command is important for saving your work and sending it to me.

b: back up a step!

L: display the argument you are currently looking at. This might be useful after a Python error (Python errors will generally not hurt your proof at all, but they do fill the display with distracting red stuff).

SA: Display an almost readable proof of what you have done so far. When you have not finished the current theorem, it will display a partial proof.

SL: Use **SL (filename)")** to create a log file. This will be called `(filename)logfile.py` and as you write Marcel commands they will be recorded in the logfile, which can actually be run under Python to reassemble your proof. Use **SL done** to close your log file (this creates a dummy log file called **donelogfile.py** of which we make no use). When you are finished with a proof (when Marcel reports QED) you might want to use the **LP** command, which inserts the proof which would be shown by **SA** into the log file as a comment.

We say something briefly about Marcel notation for propositional logic. This is driven by the fact that we need to use characters found on the typewriter keyboard! Negation is written $\sim$, conjunction (and) is written &, disjunction is written V (that is capital V and Marcel is case sensitive), implication is written `->`, and the biconditional is written `==`. Marcel has built-in order of operations: the operations presented first in the list just given bind more tightly. Implication and the biconditional group to the right, while conjunction and disjunction group to the left. You can always supply extra parentheses if you are not sure how something groups; Marcel will drop ones that it doesn't need. Use single lower case letters followed immediately by a question mark as propositional variables. We will supply completely set up start commands with Marcel notation already written out for most exercises.

I will do extensive examples in class: I am not going to try to embed the examples in this file at this point, though I may add them later.

Do the proofs in the exercises. Make log files of your proofs and send them to me in email. It is also acceptable to paste the contents of your Python window into a text file and mail that to me, but I prefer the log file because I can just run it in Python to see immediately that your work is correct.

2 Exercises for Lab I on propositional logic

1. Here are some examples with commands set up.

```
#Examples for you to try out

#s p?&(p?->q?)->q?

#s ((p?&q?)->r?)==(p?->(q?->r?))

#s p?Vq?== ~(~p?& ~q?)
```

2. Set up the other deMorgan law in Marcel and prove it.

3. Show the validity of the rule of **constructive dilemma**: from $P \vee Q$, $P \to R$ and $Q \to S$, derive $R \vee S$. To do this, you not only need to

translate the notations for the individual propositions, but also write a single larger proposition to prove. Hint: the first example proof above is the sentence that sets up a proof of the validity of the rule of modus ponens: the conjunction of the premises implies the conclusion.

4. Carry out the proof starting with

   `s (p?Vq?)&(r?V~p?)&(s?V~q?)->(r?Vs?)`

   Write out this theorem in standard propositional logic notation and write a paper proof.

5. (optional) `s (a? == b?) == c?) == (a? == (b? == c?))`

3 Remarks on how Marcel works

The ideas that Marcel is based on are closely parallel to the proof strategies in the manual of logical style. What Marcel does is take an argument, and simplify it, presenting one or two arguments whose validity is equivalent to the validity of the original argument.

One comment which can be put here as well as anywhere is that it is the one line notation $P_1, P_2, \ldots, P_n \vdash C$ for an argument which motivates the names of Marcel rules: premises appear on the left and conclusions on the right, and this is why the rules acting on premises are **l** and **gl**, and the rule acting on conclusions is **r**.

3.1 Conjunction (and)

Validity of

```
A & B

P2

P3

.
.
.

Pn

--------

C
```

is equivalent to validity of

```
A

B

P2

P3

.
.
.

Pn

--------

C
```

This corresponds to the strategy, if we have $A \wedge B$, we can deduce A and B, for using a conjunction.

Validity of

```
P1

P2

P3
.
.
.

Pn

----------

A & B
```

is equivalent to validity of both of

```
P1

P2

P3
.
.
.

Pn

----------

A
```

and

```
P1

P2

P3
.
.
.

Pn

----------

B
```

This corresponds to the strategy, to prove $A \wedge B$, first prove A, then prove B.

3.2 Disjunction (or)

The validity of

```
A V B

P2

P3

.
.
.

Pn

----------

C
```

is equivalent to the validity of both

```
A

P2

P3

.
.
.

Pn

----------

C
```

and

```
B

P2

P3

.
.
.

Pn

----------

C
```

This corresponds precisely to the strategy of proof by cases for using the hypothesis $A \vee B$.

The validity of

```
P1

P2

.
.
.

Pn

---------

A V B
```

is equivalent to the validity of

```
P1

P2

.
.
.

Pn

~B  [apply double negation if possible]

---------

A
```

This is equivalent to the strategy of alternative elimination for proving a disjunction $A \vee B$.

3.3 Negation

The validity of

```
~A

P2

P3

.
.
.

Pn

--------

C
```

is equivalent to the validity of

```
P2

P3

.
.
.

Pn

~C [apply double negation if possible]

-------

A
```

This is equivalent to an application of the contrapositive and double negation: it's an indirect proof move converting proving C given $\neg A$ to proving A given $\neg C$. Further, if C were itself a negation $\neg D$, we would get just D rather than $\neg\neg D$. Note the same comment in the rule for manipulating disjunctions as conclusions.

Validity of

```
P1

P2

.
.
.

Pn

------

~A
```

is equivalent to validity of

```
A

P1

P2

.
.
.

Pn

------

_|_
```

This is the usual strategy of negation introduction: to prove $\neg A$ assume A and deduce the absurd $\bot$.

When this appears in Marcel, you will notice that the absurd symbol gets no line number. When one of the rules is applied that moves the negation of the conclusion into the premises, the special conclusion $\bot$ simply disappears (a premise $\neg\ \bot$ has no content, it is simply true without conveying any information).

3.4 Implication (only if)

Validity of

```
P1

P2

.
.
.

Pn

----------

A -> B
```

is equivalent to the validity of

```
A

P1

P2

.
.
.

Pn

----------

B
```

This is equivalent to our direct strategy for proving an implication: to prove $A \to B$, assume A and prove B.

You may notice that we didn't put the rule for using an implication first. That is because the form of this rule in Marcel is a little unexpected.

The validity of

```
A -> B

P2

P3

.
.
.

Pn

---------

C
```

is equivalent to validity of both

```
P2

P3

.
.
.

Pn

~C  [apply double negation if possible]

---------

A
```

and

```
B

P2

P3

.
.
.

Pn

---------

C
```

This is the rule of modus ponens in disguise. The first part proves that either A or C follows from the other premises. The second part shows that

if we assume B and the other premises, we can deduce C. Now do proof by cases on the first part: if C follows from just the other premises of course we are done; otherwise, we deduce A from the other premises, and then deduce B by modus ponens from A and $A \to B$, and then the second part allows us to deduce C.

The reason it is different is that Marcel is committed for the most part to a strategy of manipulating one premise or conclusion at a time, and the usual form of *modus ponens* acts on two premises.

I am not at this point writing out the rules for the biconditional: they will not be surprising to a student of my manual of style.

3.5 How to start, how to finish, and strategy while working

The activity in a Marcel session is to prove that some statement P is a theorem (a tautology). The **s** (start) command (in the context "**s** P" sets up the argument

```
-----------

P
```

This is valid if any situation which makes all the premises true makes the conclusion true...there aren't any premises, so this is really simply saying, the conclusion must be true (is a tautology).

A proof ends (or part of a proof ends) when the conclusion is the same as one of the premises.

```
A

P1

P2

.
.
.

Pn

------

A
```

is obviously valid. Marcel's **d** command recognizes this only if the very first premise is the same as the conclusion, so you may need to use the **gl** command to move a premise identical to the conclusion to the front.

When there is more than one argument to show valid (as in an application of proof of a conjunction, use of a disjunction, use of an implication) Marcel gives you one of them to show valid, and when you show that it is valid, it serves you up the next one. When everything is finished, it will say QED.

During a proof, there are some issues of strategy. Marcel will let you act on either the conclusion or the first premise. If both the conclusion and the first premise are simple (single letters in the conclusion, single letters or negations of single letters in the premises) you should bring some other premise to the front to keep working. Sometimes you may want to bring a different premise to the front even though the current first premise is complex. One special thing to notice is that if you have a premise A and the premise $\neg A$ as well, bring $\neg A$ to the front with **gl**, act on it with **l** and you will be done: from an explicit contradiction, any conclusion follows, and it is easy to show this to Marcel.

When choosing which premise to bring to the front in a complicated conclusion, you might want to consider which premises share letters with the conclusion. A negative premise $\neg A$ can be manipulated so that the conclusion becomes A, which may be advantageous.

It is a general fact that each application of the **r** or **l** rule eliminates a logical operation from a premise or conclusion though it may make two arguments for you to show valid: all of the arguments you end up with after you apply one of these rules are in a measurable sense simpler than the argument you started with, except in the exact case of the first premise being the negation of a single letter, the conclusion being a single letter, and applying **l**. Because of this, it is mathematically provable, and not actually hard to see in practice, that if you keep applying rules you will eventually arrive at a situation where every premise is either a single letter or the negation of a single letter, and the conclusion is either a single letter or the absurd. You don't necessarily need to go this far to prove validity, but you can always get to this point. Then if either the conclusion is the same as one of the premises or one of the premises is the negation of another, you have a valid sequent and you can finish (and go on to do the same to the next argument) and if neither of these things is true you have a counterexample (make all the premises true and the conclusion false and you have shown that the argument is invalid). So, if you start with any proposition P of propositional logic, you will prove that it is a theorem if it is one, and you will be able to present a situation where it is false if is not a theorem.

The reason that this really does terminate, even though some of the rules may generate two arguments from one, is that they always generate two simpler arguments (arguments with fewer logical operations) from one. In the worst case, this can have the same exponential blowup in size of arguments as the method of truth tables (all known methods of proof in propositional logic have worst case behavior the same as that of the method of truth tables) but it is often much more efficient.