Marcel Theorem Prover Lab

Dr. Holmes

9/9/2021

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 2021.

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 ∼, 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?))`