Using the Marcel Theorem Prover with Its Native Interface (as, on the web page)

M. Randall Holmes

December 20, 2018

Contents

1 Introduction

Marcel is a higher-order theorem prover (meaning it has built-in machinery to handle sets and functions as well as elementary logic). The higher order logic of Marcel is a version of the known-to-be-consistent version NFU of Quine's set theory "New Foundations", a fact which will be entirely invisible

to someone using the system to prove theorems in elementary logic, and even not very visible to those using it to prove theorems in elementary set theory.

This manual is designed to assist a user who is talking to Marcel through the web page version, which invokes Marcel commands using Marcel's native interface instead of the command line in a Python shell.

2 Marcel's internal language (introductory)

In this section we describe a subset of Marcel's internal language for expressing logical and mathematical statements.

Propositional logic in Marcel requires a very small subset of the language.

A lower case letter followed by a question mark is a propositional variable, as `p?`.

The predeclared propositional operators are `~` (not), `&` (and), `V` (or), `->` (only if), and `==` (iff). Notice that `V` must be capitalized. We apologize for the needs of an ASCII terminal here.

The operators have precedence, the operators appearing earlier in that list being more strongly binding. Be sure to leave a space between a logical operator and a following negation symbol. Conjunction and disjunction group to the left, and implication and biconditional group to the right. You can test your understanding of precedence interactively, as you can add as many parentheses as you require to be sure you have said the right thing, and the Marcel display will omit those that Marcel finds unnecessary.

The quantifiers are `A` (for all), and `E` (there exists). Single lower case letters are variables suitable for binding. The equality relation `=` is predeclared. Capital letters after `O` in the alphabet will automatically be declared as unary or binary predicates if used as such. Unary predicates are prefixed to their argument; binary predicates are always infix, as `x R y` . A colon `:` must appear between a quantifier and the expression it binds, as for example in `Ax : Px -> Px`. The colon has the lowest possible operator precedence. Parentheses may be used as desired (and will be omitted if Marcel doesn't need them).

Single letters without any adornment are bound object variables. Single letters linked to numerals by an underscore or dollar sign are different flavors of object free variables or constants, as `x_2`, `a$3`. These constants are usually introduced first by Marcel, but the user may have occasion to type them.

This is a very minimal description of a little bit of the input language, but probably sufficient to start working in elementary logic.

3 The native interface command line (introductory)

In this section we describe enough of the native command interface to get started with proofs in elementary logic.

Each command starts with a sequence of letters and/or numerals and is followed by its argument (if it has one) up to end of line or a semicolon. More than one interface command may be written on a line separated by semicolons. Spaces may separate commands from their arguments, but spaces should not be added between arguments and following semicolons.

We begin with a little file management. The command `SL <filename>` will create a file `<filename>logfile.py` which will contain commands that you enter to Marcel: this command creates log files. To close the log file (so that you can read it) the convention is to issue the command `SL done`; I have not provided an explicit close log command, but opening a new log file closes the previous one. The Trinket web page will show you your log file in a tab in the left pane when you close it; it will be filled with Python commands, and running this Python file in a Python shell would repeat the commands you issued.

To start a proof issue the command `s <theoremtext>` where `<theoremtext>` is the statement of the theorem in the internal language. An example is `s p? -> p?`. Whitespace can be used fairly liberally inside terms of the internal language, but the argument should not be followed by spaces.

The Marcel display in its default state will show you a list of numbered premises above a bar, below which is a conclusion or the absurdity symbol `_|_`. Marcel is currently waiting for you to tell it why this argument is valid (why the given conclusion (or absurdity) follows from the given premises). Typically at any given moment Marcel has several arguments whose validity it needs you to show, but it only shows you one at a time: when you are finished with the current one, it serves up the next one.

The most basic commands manipulating this display are as follows:

1. `r` (no argument): apply an appropriate logical rule (chosen by Marcel) to the conclusion. This cannot be applied if the absurdity symbol

appears below the bar. The menmonic stands for "right" (since the conclusion is to the right of the turnstile in the sequent notation for logical arguments).

2. `l` (no argument): apply an appropriate logical rule (chosen by Marcel) to the first premise. Of course this cannot be applied if there are no premises above the bar. The menmonic stands for "left" (since the conclusion is to the left of the turnstile in the sequent notation for logical arguments).

3. `gl <linenumber>`: move the indicated premise to first position (this version of the command rotates the lines). The mnemonic stands for "get left".

4. `d` (no argument): if the first premise and the conclusion are identical, recognizes this argument as valid and serves up the next argument which you need to verify. The mnemonic stands for "done".

These commands are sufficient for propositional logic proofs. For proofs using quantifiers, one more command is needed:

1. `SU <assignment>`, where `<assignment>` consists of a "dollar-sign" constant (such as `a$3`; such constants are called "instantiables" in our technical description) followed by `:=` followed by a term of the Marcel internal language which is to replace the instantiable in all contexts in which it occurs in the given proof. Instantiables are introduced into proofs by application of logical rules to universal hypotheses and existential conclusions. The mnemonic stands for "set unknown". A dollar sign constant cannot be replaced by a term which contains an underscore or dollar sign context which does not have lower index.

A further file (and proof) management command: when you are done with your proof (when the Marcel display says Q. E. D.) the `LP` command will display and log (as a comment) an account of the proof you have constructed which is in principle readable by a human being (though not very).

Further commands which might be useful for understanding what is happening are `NG` ("next goal") which will take you to the next argument which Marcel has in reserve for you to prove, in an eccentric order which is guaranteed to eventually visit each argument in reserve; an odd point is that it

may revisit the same argument more than once before it cycles through all arguments. The command `SG` ("swap goals") takes you through the collection of goals created by the most recent rule application: this might be most valuable for understanding Marcel's proof strategy.

4 Examples

We present some examples with input to the native interface and the output with which Marcel replies.

4.1 The first example

A theorem expressing the rule of exportation is a favorite of mine.

```
Enter quit to break out of interface:   s (p? & q?) -> r? == (p? -> (q?->r?))
Line 1:


------------------

1:  p? & q? -> r? == p? -> q? -> r?
```

Notice that Marcel doesn't need any of the parentheses supplied by the user.

```
Enter quit to break out of interface:   r
Line 2
proving biconditional
p? & q? -> r? == p? -> q? -> r?
Part I =>
:
```

```
1:  p? & q? -> r?

------------------

1:  p? -> q? -> r?
```

Applying `r` gives two goals to prove, the first of which is exhibited here and the second of which will reappear later when this one has been verified.

```
Enter quit to break out of interface:   r
Line 4
prove
p? -> q? -> r?
by assuming
p?
and deducing
q? -> r?:

1:  p?
2:  p? & q? -> r?

------------------

1:  q? -> r?
```

```
Enter quit to break out of interface:   r
Line 5
prove
q? -> r?
by assuming
q?
and deducing
r?:

1:  q?
2:  p?
3:  p? & q? -> r?

------------------

1:  r?
```

Marcel handles two implication conclusions to be proved in the obvoous way. At this point both the conclusion and the first premise cannot be further analyzed, so `gl` comes into play. We show this on the next page for a better line break experience.

```
Enter quit to break out of interface:   gl 3
Line 5
prove
q? -> r?
by assuming
q?
and deducing
r?:

1:  p? & q? -> r?
2:  q?
3:  p?

------------------

1:  r?
```

```
Enter quit to break out of interface:   l
Line 6
use
p? & q? -> r?
, first part, showing that
p? & q?
or the desired conclusion holds:

1:  q?
2:  p?
3:  ~r?

------------------

1:  p? & q?
```

Marcel's strategy for using implications as premises may be surprising: it is basically doing modus ponens, but it is packaged oddly. It first tries to prove the hypothesis of the implication, then allows itself the conclusion of the implication as a new premise if it succeeds (m.p. then being usable).

```
Enter quit to break out of interface:   r
Line 8
prove
p? & q?
first part:  prove
p?:

1:  q?
2:  p?
3:  ~r?

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

```
1:  p?
```

Notice at the top of the next page the use of two command lines at once, separated by a semicolon.

```
Enter quit to break out of interface:   gl 2; d
Line 8
prove
p? & q?
first part:  prove
p?:

1:  p?
2:  ~r?
3:  q?

------------------

1:  p?



Line 9
prove
p? & q?
second part:  prove
q?:

1:  q?
2:  p?
3:  ~r?

------------------

1:  q?
```

With the `d` command at the top of the next page, we have finished one direction of the top-level implication, and the other is then served up to us. We will not comment further on the proof of the other implication, which involves very similar considerations.

```
Enter quit to break out of interface:   d

Line 7
use
p? & q? -> r?
second part, show that the desired conclusion follows
      from
r?:

1:  r?
2:  q?
3:  p?

------------------

1:  r?
```

```
Enter quit to break out of interface:   d
Line 3
proving biconditional
p? & q? -> r? == p? -> q? -> r?
Part II <=
:

1:  p? -> q? -> r?

------------------

1:  p? & q? -> r?



Enter quit to break out of interface:   r
Line 10
prove
p? & q? -> r?
by assuming
p? & q?
and deducing
r?:

1:  p? & q?
2:  p? -> q? -> r?

------------------

1:  r?
```

```
Enter quit to break out of interface:   l
Line 11
use
p? & q?
by breaking it into its parts
p?
and
q?:

1:  p?
2:  q?
3:  p? -> q? -> r?

------------------

1:  r?
```

```
Enter quit to break out of interface:   gl 3
Line 11
use
p? & q?
by breaking it into its parts
p?
and
q?:

1:  p? -> q? -> r?
2:  p?
3:  q?

------------------

1:  r?
```

```
Enter quit to break out of interface:   l
Line 12
use
p? -> q? -> r?
, first part, showing that
p?
or the desired conclusion holds:

1:  p?
2:  q?
3:  ~r?

------------------

1:  p?
```

```
Enter quit to break out of interface:   d
Line 13
use
p? -> q? -> r?
second part, show that the desired conclusion follows
      from
q? -> r?:

1:  q? -> r?
2:  p?
3:  q?

------------------

1:  r?
```

```
Enter quit to break out of interface:   l
Line 14
use
q? -> r?
, first part, showing that
q?
or the desired conclusion holds:

1:  p?
2:  q?
3:  ~r?

------------------

1:  q?
```

```
Enter quit to break out of interface:   gl 2; d
Line 14
use
q? -> r?
, first part, showing that
q?
or the desired conclusion holds:

1:  q?
2:  ~r?
3:  p?

------------------

1:  q?
```

```
Line 15
use
q? -> r?
second part, show that the desired conclusion follows
      from
r?:

1:  r?
2:  p?
3:  q?

------------------

1:  r?
```

The final **d** completes the proof, signalled by the appearance of Q.E.D and the reiteration of the first line as proved.

```
Enter quit to break out of interface:   d
Q. E. D.

Line 1:


------------------

1:  p? & q? -> r? == p? -> q? -> r?




Enter quit to break out of interface:
```

4.2 The second example

For our second example, we prove the equivalence of an implication with its contrapositive.

In entering this, it is important to separate the negation operator from other operators at least by a space.

```
Enter quit to break out of interface:   s p?->q? == ~q?-> ~p?
Line 1:


------------------

1:  p? -> q? == ~q? -> ~p?
```

```
Enter quit to break out of interface:   r
Line 2
proving biconditional
p? -> q? == ~q? -> ~p?
Part I =>
:

1:  p? -> q?

------------------

1:  ~q? -> ~p?
```

```
Enter quit to break out of interface:   r
Line 4
prove
~q? -> ~p?
by assuming
~q?
and deducing
~p?:

1:  ~q?
2:  p? -> q?

------------------

1:  ~p?
```

```
Enter quit to break out of interface:   r
Line 5
prove
~p?
by assuming
p?
and deducing a contradiction or alternative conclusion
     :

1:  p?
2:  ~q?
3:  p? -> q?

------------------

_|_
```

Notice the appearance of the absurd conclusion here (actually there is no conclusion line to this argument).

```
Enter quit to break out of interface:   gl 2; l
Line 5
prove
~p?
by assuming
p?
and deducing a contradiction or alternative conclusion
     :

1:  ~q?
2:  p? -> q?
3:  p?

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

```
_|_


Line 6
use
~q?
by denying conclusion and proving
q?:

1:  p? -> q?
2:  p?

------------------

1:  q?
```

```
Enter quit to break out of interface:   l
Line 7
use
p? -> q?
, first part, showing that
p?
or the desired conclusion holds:

1:  p?
2:  ~q?

------------------

1:  p?



Enter quit to break out of interface:   d
Line 8
use
p? -> q?
second part, show that the desired conclusion follows
      from
q?:

1:  q?
2:  p?

------------------

1:  q?
```

```
Enter quit to break out of interface:   d
Line 3
proving biconditional
p? -> q? == ~q? -> ~p?
Part II <=
:

1:  ~q? -> ~p?

------------------

1:  p? -> q?



Enter quit to break out of interface:   r
Line 9
prove
p? -> q?
by assuming
p?
and deducing
q?:

1:  p?
2:  ~q? -> ~p?

------------------

1:  q?
```

```
Enter quit to break out of interface:   gl 2; l
Line 9
prove
p? -> q?
by assuming
p?
and deducing
q?:

1:  ~q? -> ~p?
2:  p?

------------------

1:  q?




Line 10
use
~q? -> ~p?
, first part, showing that
~q?
or the desired conclusion holds:

1:  p?
2:  ~q?

------------------

1:  ~q?
```

```
Enter quit to break out of interface:   gl 2
Line 10
use
~q? -> ~p?
, first part, showing that
~q?
or the desired conclusion holds:

1:  ~q?
2:  p?

------------------

1:  ~q?



Enter quit to break out of interface:   d
Line 11
use
~q? -> ~p?
second part, show that the desired conclusion follows
      from
~p?:

1:  ~p?
2:  p?

------------------

1:  q?
```

Notice how we deal with contradictory premises.

```
Enter quit to break out of interface:   l
Line 12
use
~p?
by denying conclusion and proving
p?:

1:  p?
2:  ~q?

------------------

1:  p?




Enter quit to break out of interface:   d
Q. E. D.

Line 1:


------------------

1:  p? -> q? == ~q? -> ~p?




Enter quit to break out of interface:
```

4.3 The third example

Here is an apparently simple but mysterious quantifier proof.

```
Enter quit to break out of interface:   s Ex:Ay:Px -> Py

P:  types-output type then input type(s):  ['prop', 'object']
precedence (even groups right, odd left):          12

Line 1:


------------------

1:  E x : A y : Px -> Py
```

Note that Marcel obligingly automatically declares `P` as a unary predicate. This facility is provided for ease of access for students; explicit declaration commands are to be found in the full command reference!

```
Enter quit to break out of interface:   r
Line 2
prove the existential
E x : A y : Px -> Py
by introducing an instance to be further specified
     then proved:

1:  ~ E x : A y : Px -> Py

------------------

1:  A y : Px$1 -> Py
```

The right rule for the existential quantifier requires us to find a witness `x$1` to the quantified statement. The `SU` command can be used to give such

an "instantiable" a specific value (though we will actually never instantiate this one!).

Note further that the existential conclusion does not disappear, as we may want to instantiate it again (in fact, we will).

```
Enter quit to break out of interface:   r
Line 3
prove the universal
A y : Px$1 -> Py
by proving an arbitrary instance
:

1:  ~ E x : A y : Px -> Py

------------------

1:  Px$1 -> Py_2
```

We prove a universal statement by showing that the statement holds for an arbitrary object (in this case the new `y_2`).

```
Enter quit to break out of interface:   r
Line 4
prove
Px$1 -> Py_2
by assuming
Px$1
and deducing
Py_2:

1:  Px$1
2:  ~ E x : A y : Px -> Py

------------------

1:  Py_2
```

```
Enter quit to break out of interface:   SU x$1 := y_2
x$1:= y_2

substitution fails because term is newer than instantiable


Line 4
prove
Px$1 -> Py_2
by assuming
Px$1
and deducing
Py_2:

1:  Px$1
2:  ~ E x : A y : Px -> Py

------------------

1:  Py_2
```

Notice the restriction on instantiation here. The instantiable `x$1` is standing in for some witness that we could have entered at that point in the proof. And `y_2` could not have been mentioned in that context, being an arbitrary object introduced in a smaller scope.

```
Enter quit to break out of interface:   gl 2
Line 4
prove
Px$1 -> Py_2
by assuming
Px$1
and deducing
Py_2:

1:  ~ E x : A y : Px -> Py
2:  Px$1

------------------

1:  Py_2



Enter quit to break out of interface:   l
Line 5
use
~ E x : A y : Px -> Py
by denying conclusion and proving
E x : A y : Px -> Py:

1:  Px$1
2:  ~ Py_2

------------------

1:  E x : A y : Px -> Py
```

```
Enter quit to break out of interface:   r
Line 6
prove the existential
E x : A y : Px -> Py
by introducing an instance to be further specified
     then proved:

1:  Px$1
2:  ~ Py_2
3:  ~ E x : A y : Px -> Py

------------------

1:  A y : Px$3 -> Py



Enter quit to break out of interface:   r
Line 7
prove the universal
A y : Px$3 -> Py
by proving an arbitrary instance
:

1:  Px$1
2:  ~ Py_2
3:  ~ E x : A y : Px -> Py

------------------

1:  Px$3 -> Py_4
```

```
Enter quit to break out of interface:   r
Line 8
prove
Px$3 -> Py_4
by assuming
Px$3
and deducing
Py_4:

1:  Px$3
2:  Px$1
3:  ~ Py_2
4:  ~ E x : A y : Px -> Py

------------------

1:  Py_4
```

```
Enter quit to break out of interface:   gl 3
Line 8
prove
Px$3 -> Py_4
by assuming
Px$3
and deducing
Py_4:

1:  ~ Py_2
2:  ~ E x : A y : Px -> Py
3:  Px$3
4:  Px$1

------------------

1:  Py_4
```

```
Enter quit to break out of interface:   l
Line 9
use
~ Py_2
by denying conclusion and proving
Py_2:

1:  ~ E x : A y : Px -> Py
2:  Px$3
3:  Px$1
4:  ~ Py_4

------------------

1:  Py_2



Enter quit to break out of interface:   gl 2
Line 9
use
~ Py_2
by denying conclusion and proving
Py_2:

1:  Px$3
2:  Px$1
3:  ~ Py_4
4:  ~ E x : A y : Px -> Py

------------------

1:  Py_2
```

```
Enter quit to break out of interface:   SU x$3 := y_2
x$3:= y_2
Line 9
use
~ Py_2
by denying conclusion and proving
Py_2:

1:  Py_2
2:  Px$1
3:  ~ Py_4
4:  ~ E x : A y : Px -> Py

------------------

1:  Py_2
```

And here we have a successful instantiation.

```
Enter quit to break out of interface:   d
Q. E. D.

Line 1:


------------------

1:  E x : A y : Px -> Py




Enter quit to break out of interface:
```

The exact way that this proof works will be quite mysterious to an undergraduate. It doesn't seem entirely trivial to me!

4.4 The fourth example

The following is the repaired version of a fallacious argument that any symmetric, transitive relation is reflexive.

```
Enter quit to break out of interface:   s (Ax:Ey:x R y) & (Ax:Ay:xRy==yRx) & (A

R:  types-output type then input type(s):  ['prop', 0, 0]
precedence (even groups right, odd left):          12

Line 1:


------------------

1:  (A x : E y : x R y) & (A x : A y : x R y == y R x)
     & (A x : A y : A z : x R y & y R z -> x R z) -> (A x
      : x R x)
```

The input line is too long to see, but the version processed by Marcel is visible. Note the automatic declaration of the relation introduced.

```
Enter quit to break out of interface:   r
Line 2
prove
(A x : E y : x R y) & (A x : A y : x R y == y R x)
     & (A x : A y : A z : x R y & y R z -> x R z) -> (A x
      : x R x)
by assuming
(A x : E y : x R y) & (A x : A y : x R y == y R x)
     & (A x : A y : A z : x R y & y R z -> x R z)
and deducing
A x : x R x:

1:  (A x : E y : x R y) & (A x : A y : x R y == y R x)
     & (A x : A y : A z : x R y & y R z -> x R z)


------------------

1:  A x : x R x
```

```
Enter quit to break out of interface:   l
Line 3
use
(A x : E y : x R y) & (A x : A y : x R y == y R x)
     & (A x : A y : A z : x R y & y R z -> x R z)
by breaking it into its parts
(A x : E y : x R y) & (A x : A y : x R y == y R x)
and
A x : A y : A z : x R y & y R z -> x R z:

1:  (A x : E y : x R y) & (A x : A y : x R y == y R x)

2:  A x : A y : A z : x R y & y R z -> x R z

------------------

1:  A x : x R x
```

```
Enter quit to break out of interface:   l
Line 4
use
(A x : E y : x R y) & (A x : A y : x R y == y R x)
by breaking it into its parts
A x : E y : x R y
and
A x : A y : x R y == y R x:

1:  A x : E y : x R y
2:  A x : A y : x R y == y R x
3:  A x : A y : A z : x R y & y R z -> x R z

------------------

1:  A x : x R x
```

The left grouping of conjunction and disjunction is designed to allow neat unpacking of long lists of hypotheses.

```
Enter quit to break out of interface:   r
Line 5
prove the universal
A x : x R x
by proving an arbitrary instance
:

1:  A x : E y : x R y
2:  A x : A y : x R y == y R x
3:  A x : A y : A z : x R y & y R z -> x R z

------------------

1:  x_1 R x_1
```

We set up the proof of the universal conclusion in the natural way, introducing an arbitrary object which is to be shown to stand in the relation to itself.

```
Enter quit to break out of interface:   l
Line 6
use the universal hypothesis
A x : E y : x R y
by creating an instance to be further specified and
      used:

1:  E y : x$2 R y
2:  A x : E y : x R y
3:  A x : A y : x R y == y R x
4:  A x : A y : A z : x R y & y R z -> x R z

------------------

1:  x_1 R x_1
```

We illustrate instantiation of a universal hypothesis (with an instantiable whose value we will set shortly). Notice that the universal hypothesis lingers: we might want to instantiate it again, though we do not in this proof.

```
Enter quit to break out of interface:   l
Line 7
use the existential hypothesis
E y : x$2 R y
by introducing a new witness:

1:  x$2 R y_3
2:  A x : E y : x R y
3:  A x : A y : x R y == y R x
4:  A x : A y : A z : x R y & y R z -> x R z

------------------

1:  x_1 R x_1
```

We illustrate introduction of an arbitrary witness to an existential hypothesis.

```
Enter quit to break out of interface:   SU x$2 := x_1
x$2:= x_1
Line 7
use the existential hypothesis
E y : x$2 R y
by introducing a new witness:

1:  x_1 R y_3
2:  A x : E y : x R y
3:  A x : A y : x R y == y R x
4:  A x : A y : A z : x R y & y R z -> x R z

------------------

1:  x_1 R x_1
```

Of course the `x$2` we are looking for is `x_1`.

```
Enter quit to break out of interface:
Line 7
use the existential hypothesis
E y : x$2 R y
by introducing a new witness:

1:  x_1 R y_3
2:  A x : E y : x R y
3:  A x : A y : x R y == y R x
4:  A x : A y : A z : x R y & y R z -> x R z

------------------

1:  x_1 R x_1
```

```
Enter quit to break out of interface:   gl 3
Line 7
use the existential hypothesis
E y : x$2 R y
by introducing a new witness:

1:  A x : A y : x R y == y R x
2:  A x : A y : A z : x R y & y R z -> x R z
3:  x_1 R y_3
4:  A x : E y : x R y

------------------

1:  x_1 R x_1
```

```
Enter quit to break out of interface:   l
Line 8
use the universal hypothesis
A x : A y : x R y == y R x
by creating an instance to be further specified and
      used:

1:  A y : x$4 R y == y R x$4
2:  A x : A y : x R y == y R x
3:  A x : A y : A z : x R y & y R z -> x R z
4:  x_1 R y_3
5:  A x : E y : x R y

------------------

1:  x_1 R x_1
```

We start instantiating the symmetric property of the relation with intent which should be obvious.

```
Enter quit to break out of interface:   l
Line 9
use the universal hypothesis
A y : x$4 R y == y R x$4
by creating an instance to be further specified and
      used:

1:  x$4 R y$5 == y$5 R x$4
2:  A y : x$4 R y == y R x$4
3:  A x : A y : x R y == y R x
4:  A x : A y : A z : x R y & y R z -> x R z
5:  x_1 R y_3
6:  A x : E y : x R y

------------------

1:  x_1 R x_1
```

We are done instantiating it.

```
Enter quit to break out of interface:   SU x$4 := x_1
x$4:= x_1
Line 9
use the universal hypothesis
A y : x$4 R y == y R x$4
by creating an instance to be further specified and
      used:

1:  x_1 R y$5 == y$5 R x_1
2:  A y : x_1 R y == y R x_1
3:  A x : A y : x R y == y R x
4:  A x : A y : A z : x R y & y R z -> x R z
5:  x_1 R y_3
6:  A x : E y : x R y

------------------

1:  x_1 R x_1
```

We fill in the first instantiable

```
Enter quit to break out of interface:   SU y$5 := y_3
y$5:= y_3
Line 9
use the universal hypothesis
A y : x$4 R y == y R x$4
by creating an instance to be further specified and
      used:

1:  x_1 R y_3 == y_3 R x_1
2:  A y : x_1 R y == y R x_1
3:  A x : A y : x R y == y R x
4:  A x : A y : A z : x R y & y R z -> x R z
5:  x_1 R y_3
6:  A x : E y : x R y

------------------

1:  x_1 R x_1
```

and the second.

```
Enter quit to break out of interface:   l
Line 10
break biconditional assumption
x_1 R y_3 == y_3 R x_1
into its constituent implications:

1:  x_1 R y_3 -> y_3 R x_1
2:  y_3 R x_1 -> x_1 R y_3
3:  A y : x_1 R y == y R x_1
4:  A x : A y : x R y == y R x
5:  A x : A y : A z : x R y & y R z -> x R z
6:  x_1 R y_3
7:  A x : E y : x R y

------------------

1:  x_1 R x_1
```

Now we do the propositional reasoning to apply symmetry.

```
Enter quit to break out of interface:   l
Line 11
use
x_1 R y_3 -> y_3 R x_1
, first part, showing that
x_1 R y_3
or the desired conclusion holds:

1:  y_3 R x_1 -> x_1 R y_3
2:  A y : x_1 R y == y R x_1
3:  A x : A y : x R y == y R x
4:  A x : A y : A z : x R y & y R z -> x R z
5:  x_1 R y_3
6:  A x : E y : x R y
7:  ~ x_1 R x_1

------------------

1:  x_1 R y_3
```

```
Enter quit to break out of interface:   gl 5; d
Line 11
use
x_1 R y_3 -> y_3 R x_1
, first part, showing that
x_1 R y_3
or the desired conclusion holds:

1:  x_1 R y_3
2:  A x : E y : x R y
3:  ~ x_1 R x_1
4:  y_3 R x_1 -> x_1 R y_3
5:  A y : x_1 R y == y R x_1
6:  A x : A y : x R y == y R x
7:  A x : A y : A z : x R y & y R z -> x R z

------------------

1:  x_1 R y_3
```

```
Line 12
use
x_1 R y_3 -> y_3 R x_1
second part, show that the desired conclusion follows
      from
y_3 R x_1:

1:  y_3 R x_1
2:  y_3 R x_1 -> x_1 R y_3
3:  A y : x_1 R y == y R x_1
4:  A x : A y : x R y == y R x
5:  A x : A y : A z : x R y & y R z -> x R z
6:  x_1 R y_3
7:  A x : E y : x R y

------------------

1:  x_1 R x_1
```

and we have the additional assertion `y_3Rx_1` we have aimed for by applying symmetry. Line 1 and line 6 now suggest that we apply transitivity to prove our desired conclusion.

```
Enter quit to break out of interface:   gl 5
Line 12
use
x_1 R y_3 -> y_3 R x_1
second part, show that the desired conclusion follows
      from
y_3 R x_1:

1:  A x : A y : A z : x R y & y R z -> x R z
2:  x_1 R y_3
3:  A x : E y : x R y
4:  y_3 R x_1
5:  y_3 R x_1 -> x_1 R y_3
6:  A y : x_1 R y == y R x_1
7:  A x : A y : x R y == y R x

------------------

1:  x_1 R x_1
```

```
Enter quit to break out of interface:   l; l; l
Line 13
use the universal hypothesis
A x : A y : A z : x R y & y R z -> x R z
by creating an instance to be further specified and
      used:

1:  A y : A z : x$6 R y & y R z -> x$6 R z
2:  A x : A y : A z : x R y & y R z -> x R z
3:  x_1 R y_3
4:  A x : E y : x R y
5:  y_3 R x_1
6:  y_3 R x_1 -> x_1 R y_3
7:  A y : x_1 R y == y R x_1
8:  A x : A y : x R y == y R x

------------------

1:  x_1 R x_1
```

Instantiating once...

```
Line 14
use the universal hypothesis
A y : A z : x$6 R y & y R z -> x$6 R z
by creating an instance to be further specified and
      used:

1:  A z : x$6 R y$7 & y$7 R z -> x$6 R z
2:  A y : A z : x$6 R y & y R z -> x$6 R z
3:  A x : A y : A z : x R y & y R z -> x R z
4:  x_1 R y_3
5:  A x : E y : x R y
6:  y_3 R x_1
7:  y_3 R x_1 -> x_1 R y_3
8:  A y : x_1 R y == y R x_1
9:  A x : A y : x R y == y R x

------------------

1:  x_1 R x_1
```

Instantiating twice...

```
Line 15
use the universal hypothesis
A z : x$6 R y$7 & y$7 R z -> x$6 R z
by creating an instance to be further specified and
      used:

1:  x$6 R y$7 & y$7 R z$8 -> x$6 R z$8
2:  A z : x$6 R y$7 & y$7 R z -> x$6 R z
3:  A y : A z : x$6 R y & y R z -> x$6 R z
4:  A x : A y : A z : x R y & y R z -> x R z
5:  x_1 R y_3
6:  A x : E y : x R y
7:  y_3 R x_1
8:  y_3 R x_1 -> x_1 R y_3
9:  A y : x_1 R y == y R x_1
10:  A x : A y : x R y == y R x

------------------

1:  x_1 R x_1
```

Instantiating three times....

```
Enter quit to break out of interface:   SU x$6 := x_1
x$6:= x_1
Line 15
use the universal hypothesis
A z : x$6 R y$7 & y$7 R z -> x$6 R z
by creating an instance to be further specified and
      used:

1:  x_1 R y$7 & y$7 R z$8 -> x_1 R z$8
2:  A z : x_1 R y$7 & y$7 R z -> x_1 R z
3:  A y : A z : x_1 R y & y R z -> x_1 R z
4:  A x : A y : A z : x R y & y R z -> x R z
5:  x_1 R y_3
6:  A x : E y : x R y
7:  y_3 R x_1
8:  y_3 R x_1 -> x_1 R y_3
9:  A y : x_1 R y == y R x_1
10:  A x : A y : x R y == y R x

------------------

1:  x_1 R x_1
```

Specifying once...

```
Enter quit to break out of interface:   SU y$7 := y_3
y$7:= y_3
Line 15
use the universal hypothesis
A z : x$6 R y$7 & y$7 R z -> x$6 R z
by creating an instance to be further specified and
      used:

1:  x_1 R y_3 & y_3 R z$8 -> x_1 R z$8
2:  A z : x_1 R y_3 & y_3 R z -> x_1 R z
3:  A y : A z : x_1 R y & y R z -> x_1 R z
4:  A x : A y : A z : x R y & y R z -> x R z
5:  x_1 R y_3
6:  A x : E y : x R y
7:  y_3 R x_1
8:  y_3 R x_1 -> x_1 R y_3
9:  A y : x_1 R y == y R x_1
10:  A x : A y : x R y == y R x

------------------

1:  x_1 R x_1
```

Specifying twice...

```
Enter quit to break out of interface:   SU z$8 := x_1
z$8:= x_1
Line 15
use the universal hypothesis
A z : x$6 R y$7 & y$7 R z -> x$6 R z
by creating an instance to be further specified and
      used:

1:  x_1 R y_3 & y_3 R x_1 -> x_1 R x_1
2:  A z : x_1 R y_3 & y_3 R z -> x_1 R z
3:  A y : A z : x_1 R y & y R z -> x_1 R z
4:  A x : A y : A z : x R y & y R z -> x R z
5:  x_1 R y_3
6:  A x : E y : x R y
7:  y_3 R x_1
8:  y_3 R x_1 -> x_1 R y_3
9:  A y : x_1 R y == y R x_1
10:  A x : A y : x R y == y R x

------------------

1:  x_1 R x_1
```

Specifying three times...

```
Enter quit to break out of interface:   l()
Enter quit to break out of interface:   l
Line 16
use
x_1 R y_3 & y_3 R x_1 -> x_1 R x_1
, first part, showing that
x_1 R y_3 & y_3 R x_1
or the desired conclusion holds:

1:  A z : x_1 R y_3 & y_3 R z -> x_1 R z
2:  A y : A z : x_1 R y & y R z -> x_1 R z
3:  A x : A y : A z : x R y & y R z -> x R z
4:  x_1 R y_3
5:  A x : E y : x R y
6:  y_3 R x_1
7:  y_3 R x_1 -> x_1 R y_3
8:  A y : x_1 R y == y R x_1
9:  A x : A y : x R y == y R x
10:  ~ x_1 R x_1

------------------

1:  x_1 R y_3 & y_3 R x_1
```

Doing propositional logic...

```
Enter quit to break out of interface:   r
Line 18
prove
x_1 R y_3 & y_3 R x_1
first part:  prove
x_1 R y_3:

1:  A z : x_1 R y_3 & y_3 R z -> x_1 R z
2:  A y : A z : x_1 R y & y R z -> x_1 R z
3:  A x : A y : A z : x R y & y R z -> x R z
4:  x_1 R y_3
5:  A x : E y : x R y
6:  y_3 R x_1
7:  y_3 R x_1 -> x_1 R y_3
8:  A y : x_1 R y == y R x_1
9:  A x : A y : x R y == y R x
10:  ~ x_1 R x_1

------------------

1:  x_1 R y_3
```

```
Enter quit to break out of interface:   gl 4; d
Line 18
prove
x_1 R y_3 & y_3 R x_1
first part:  prove
x_1 R y_3:

1:  x_1 R y_3
2:  A x : E y : x R y
3:  y_3 R x_1
4:  y_3 R x_1 -> x_1 R y_3
5:  A y : x_1 R y == y R x_1
6:  A x : A y : x R y == y R x
7:  ~ x_1 R x_1
8:  A z : x_1 R y_3 & y_3 R z -> x_1 R z
9:  A y : A z : x_1 R y & y R z -> x_1 R z
10:  A x : A y : A z : x R y & y R z -> x R z

------------------

1:  x_1 R y_3
```

```
Line 19
prove
x_1 R y_3 & y_3 R x_1
second part:  prove
y_3 R x_1:

1:  A z : x_1 R y_3 & y_3 R z -> x_1 R z
2:  A y : A z : x_1 R y & y R z -> x_1 R z
3:  A x : A y : A z : x R y & y R z -> x R z
4:  x_1 R y_3
5:  A x : E y : x R y
6:  y_3 R x_1
7:  y_3 R x_1 -> x_1 R y_3
8:  A y : x_1 R y == y R x_1
9:  A x : A y : x R y == y R x
10:  ~ x_1 R x_1

------------------

1:  y_3 R x_1
```

```
Enter quit to break out of interface:   gl 6; d
Line 19
prove
x_1 R y_3 & y_3 R x_1
second part:  prove
y_3 R x_1:

1:  y_3 R x_1
2:  y_3 R x_1 -> x_1 R y_3
3:  A y : x_1 R y == y R x_1
4:  A x : A y : x R y == y R x
5:  ~ x_1 R x_1
6:  A z : x_1 R y_3 & y_3 R z -> x_1 R z
7:  A y : A z : x_1 R y & y R z -> x_1 R z
8:  A x : A y : A z : x R y & y R z -> x R z
9:  x_1 R y_3
10:  A x : E y : x R y

------------------

1:  y_3 R x_1
```

```
Line 17
use
x_1 R y_3 & y_3 R x_1 -> x_1 R x_1
second part, show that the desired conclusion follows
      from
x_1 R x_1:

1:  x_1 R x_1
2:  A z : x_1 R y_3 & y_3 R z -> x_1 R z
3:  A y : A z : x_1 R y & y R z -> x_1 R z
4:  A x : A y : A z : x R y & y R z -> x R z
5:  x_1 R y_3
6:  A x : E y : x R y
7:  y_3 R x_1
8:  y_3 R x_1 -> x_1 R y_3
9:  A y : x_1 R y == y R x_1
10:  A x : A y : x R y == y R x

------------------

1:  x_1 R x_1
```

and we are there.

```
Enter quit to break out of interface:   d
Q. E. D.

Line 1:


------------------

1:  (A x : E y : x R y) & (A x : A y : x R y == y R x)
     & (A x : A y : A z : x R y & y R z -> x R z) -> (A x
      : x R x)




Enter quit to break out of interface:
```

It is important to notice that the length of this proof is not correctly measured by the number of pages it takes to present here. The length is more accurately measured by the amount of text the user had to enter, which is much, much shorter.

4.5 Reprise of fourth example

We reprise the fourth example in a much more sophisticated way. The `d` command has more powers than we have revealed: if the first premise and the conclusion can be made to match by assigning values to instantiables, Marcel will make those assignments. Be careful that they are the ones you want, though!

```
Enter quit to break out of interface:   s (Ax:Ey:x R y) & (Ax:Ay:xRy==yRx) & (A

R:  types-output type then input type(s):  ['prop', 0, 0]
precedence (even groups right, odd left):          12
```

```
Line 1:


------------------

1:  (A x : E y : x R y) & (A x : A y : x R y == y R x)
     & (A x : A y : A z : x R y & y R z -> x R z) -> (A x
      : x R x)




Enter quit to break out of interface:   r
Line 2
prove
(A x : E y : x R y) & (A x : A y : x R y == y R x)
     & (A x : A y : A z : x R y & y R z -> x R z) -> (A x
      : x R x)
by assuming
(A x : E y : x R y) & (A x : A y : x R y == y R x)
     & (A x : A y : A z : x R y & y R z -> x R z)
and deducing
A x : x R x:

1:  (A x : E y : x R y) & (A x : A y : x R y == y R x)
     & (A x : A y : A z : x R y & y R z -> x R z)


------------------

1:  A x : x R x
```

The entry line runs off the page as before.

```
Enter quit to break out of interface:   r
Line 3
prove the universal
A x : x R x
by proving an arbitrary instance
:

1:  (A x : E y : x R y) & (A x : A y : x R y == y R x)
     & (A x : A y : A z : x R y & y R z -> x R z)


------------------

1:  x_1 R x_1



Enter quit to break out of interface:   l
Line 4
use
(A x : E y : x R y) & (A x : A y : x R y == y R x)
     & (A x : A y : A z : x R y & y R z -> x R z)
by breaking it into its parts
(A x : E y : x R y) & (A x : A y : x R y == y R x)
and
A x : A y : A z : x R y & y R z -> x R z:

1:  (A x : E y : x R y) & (A x : A y : x R y == y R x)

2:  A x : A y : A z : x R y & y R z -> x R z

------------------

1:  x_1 R x_1
```

```
Enter quit to break out of interface:   l
Line 5
use
(A x : E y : x R y) & (A x : A y : x R y == y R x)
by breaking it into its parts
A x : E y : x R y
and
A x : A y : x R y == y R x:

1:  A x : E y : x R y
2:  A x : A y : x R y == y R x
3:  A x : A y : A z : x R y & y R z -> x R z

------------------

1:  x_1 R x_1



Enter quit to break out of interface:   l
Line 6
use the universal hypothesis
A x : E y : x R y
by creating an instance to be further specified and
      used:

1:  E y : x$2 R y
2:  A x : E y : x R y
3:  A x : A y : x R y == y R x
4:  A x : A y : A z : x R y & y R z -> x R z

------------------

1:  x_1 R x_1
```

```
Enter quit to break out of interface:   l
Line 7
use the existential hypothesis
E y : x$2 R y
by introducing a new witness:

1:  x$2 R y_3
2:  A x : E y : x R y
3:  A x : A y : x R y == y R x
4:  A x : A y : A z : x R y & y R z -> x R z

------------------

1:  x_1 R x_1



Enter quit to break out of interface:   SU x$2 := x_1
x$2:= x_1
Line 7
use the existential hypothesis
E y : x$2 R y
by introducing a new witness:

1:  x_1 R y_3
2:  A x : E y : x R y
3:  A x : A y : x R y == y R x
4:  A x : A y : A z : x R y & y R z -> x R z

------------------

1:  x_1 R x_1
```

This is the only "set unknown" command in this version!

```
Enter quit to break out of interface:   gl 3
Line 7
use the existential hypothesis
E y : x$2 R y
by introducing a new witness:

1:  A x : A y : x R y == y R x
2:  A x : A y : A z : x R y & y R z -> x R z
3:  x_1 R y_3
4:  A x : E y : x R y

------------------

1:  x_1 R x_1



Enter quit to break out of interface:   l
Line 8
use the universal hypothesis
A x : A y : x R y == y R x
by creating an instance to be further specified and
      used:

1:  A y : x$4 R y == y R x$4
2:  A x : A y : x R y == y R x
3:  A x : A y : A z : x R y & y R z -> x R z
4:  x_1 R y_3
5:  A x : E y : x R y

------------------

1:  x_1 R x_1
```

```
Enter quit to break out of interface:   l
Line 9
use the universal hypothesis
A y : x$4 R y == y R x$4
by creating an instance to be further specified and
      used:

1:  x$4 R y$5 == y$5 R x$4
2:  A y : x$4 R y == y R x$4
3:  A x : A y : x R y == y R x
4:  A x : A y : A z : x R y & y R z -> x R z
5:  x_1 R y_3
6:  A x : E y : x R y

------------------

1:  x_1 R x_1
```

```
Enter quit to break out of interface:   l
Line 10
break biconditional assumption
x$4 R y$5 == y$5 R x$4
into its constituent implications:

1:  x$4 R y$5 -> y$5 R x$4
2:  y$5 R x$4 -> x$4 R y$5
3:  A y : x$4 R y == y R x$4
4:  A x : A y : x R y == y R x
5:  A x : A y : A z : x R y & y R z -> x R z
6:  x_1 R y_3
7:  A x : E y : x R y

------------------

1:  x_1 R x_1
```

```
Enter quit to break out of interface:   l
Line 11
use
x$4 R y$5 -> y$5 R x$4
, first part, showing that
x$4 R y$5
or the desired conclusion holds:

1:  y$5 R x$4 -> x$4 R y$5
2:  A y : x$4 R y == y R x$4
3:  A x : A y : x R y == y R x
4:  A x : A y : A z : x R y & y R z -> x R z
5:  x_1 R y_3
6:  A x : E y : x R y
7:  ~ x_1 R x_1

------------------

1:  x$4 R y$5
```

```
Enter quit to break out of interface:   gl 5; d
Line 11
use
x$4 R y$5 -> y$5 R x$4
, first part, showing that
x$4 R y$5
or the desired conclusion holds:

1:  x_1 R y_3
2:  A x : E y : x R y
3:  ~ x_1 R x_1
4:  y$5 R x$4 -> x$4 R y$5
5:  A y : x$4 R y == y R x$4
6:  A x : A y : x R y == y R x
7:  A x : A y : A z : x R y & y R z -> x R z

------------------

1:  x$4 R y$5
```

You can see on the next page that the "done" command here determines what values to assign to `x$4` and `y$5`.

```
Line 12
use
x$4 R y$5 -> y$5 R x$4
second part, show that the desired conclusion follows
      from
y$5 R x$4:

1:  y_3 R x_1
2:  y_3 R x_1 -> x_1 R y_3
3:  A y : x_1 R y == y R x_1
4:  A x : A y : x R y == y R x
5:  A x : A y : A z : x R y & y R z -> x R z
6:  x_1 R y_3
7:  A x : E y : x R y

------------------

1:  x_1 R x_1
```

Something which I would like to figure out a way to fix is that instantiables in the text that Marcel generates as hints as to how the proof is carried out in human terms are not updated, since they are generated at the time of original rule invocation.

```
Enter quit to break out of interface:   gl 5; l; l; l
Line 12
use
x$4 R y$5 -> y$5 R x$4
second part, show that the desired conclusion follows
      from
y$5 R x$4:

1:  A x : A y : A z : x R y & y R z -> x R z
2:  x_1 R y_3
3:  A x : E y : x R y
4:  y_3 R x_1
5:  y_3 R x_1 -> x_1 R y_3
6:  A y : x_1 R y == y R x_1
7:  A x : A y : x R y == y R x

------------------

1:  x_1 R x_1
```

```
Line 13
use the universal hypothesis
A x : A y : A z : x R y & y R z -> x R z
by creating an instance to be further specified and
      used:

1:  A y : A z : x$6 R y & y R z -> x$6 R z
2:  A x : A y : A z : x R y & y R z -> x R z
3:  x_1 R y_3
4:  A x : E y : x R y
5:  y_3 R x_1
6:  y_3 R x_1 -> x_1 R y_3
7:  A y : x_1 R y == y R x_1
8:  A x : A y : x R y == y R x

------------------

1:  x_1 R x_1
```

```
Line 14
use the universal hypothesis
A y : A z : x$6 R y & y R z -> x$6 R z
by creating an instance to be further specified and
      used:

1:  A z : x$6 R y$7 & y$7 R z -> x$6 R z
2:  A y : A z : x$6 R y & y R z -> x$6 R z
3:  A x : A y : A z : x R y & y R z -> x R z
4:  x_1 R y_3
5:  A x : E y : x R y
6:  y_3 R x_1
7:  y_3 R x_1 -> x_1 R y_3
8:  A y : x_1 R y == y R x_1
9:  A x : A y : x R y == y R x

------------------

1:  x_1 R x_1
```

```
Line 15
use the universal hypothesis
A z : x$6 R y$7 & y$7 R z -> x$6 R z
by creating an instance to be further specified and
      used:

1:  x$6 R y$7 & y$7 R z$8 -> x$6 R z$8
2:  A z : x$6 R y$7 & y$7 R z -> x$6 R z
3:  A y : A z : x$6 R y & y R z -> x$6 R z
4:  A x : A y : A z : x R y & y R z -> x R z
5:  x_1 R y_3
6:  A x : E y : x R y
7:  y_3 R x_1
8:  y_3 R x_1 -> x_1 R y_3
9:  A y : x_1 R y == y R x_1
10:  A x : A y : x R y == y R x

------------------

1:  x_1 R x_1
```

```
Enter quit to break out of interface:   l
Line 16
use
x$6 R y$7 & y$7 R z$8 -> x$6 R z$8
, first part, showing that
x$6 R y$7 & y$7 R z$8
or the desired conclusion holds:

1:  A z : x$6 R y$7 & y$7 R z -> x$6 R z
2:  A y : A z : x$6 R y & y R z -> x$6 R z
3:  A x : A y : A z : x R y & y R z -> x R z
4:  x_1 R y_3
5:  A x : E y : x R y
6:  y_3 R x_1
7:  y_3 R x_1 -> x_1 R y_3
8:  A y : x_1 R y == y R x_1
9:  A x : A y : x R y == y R x
10:  ~ x_1 R x_1

------------------

1:  x$6 R y$7 & y$7 R z$8
```

```
Enter quit to break out of interface:   r
Line 18
prove
x$6 R y$7 & y$7 R z$8
first part:  prove
x$6 R y$7:

1:  A z : x$6 R y$7 & y$7 R z -> x$6 R z
2:  A y : A z : x$6 R y & y R z -> x$6 R z
3:  A x : A y : A z : x R y & y R z -> x R z
4:  x_1 R y_3
5:  A x : E y : x R y
6:  y_3 R x_1
7:  y_3 R x_1 -> x_1 R y_3
8:  A y : x_1 R y == y R x_1
9:  A x : A y : x R y == y R x
10:  ~ x_1 R x_1

------------------

1:  x$6 R y$7
```

```
Enter quit to break out of interface:   gl 4; d
Line 18
prove
x$6 R y$7 & y$7 R z$8
first part:  prove
x$6 R y$7:

1:  x_1 R y_3
2:  A x : E y : x R y
3:  y_3 R x_1
4:  y_3 R x_1 -> x_1 R y_3
5:  A y : x_1 R y == y R x_1
6:  A x : A y : x R y == y R x
7:  ~ x_1 R x_1
8:  A z : x$6 R y$7 & y$7 R z -> x$6 R z
9:  A y : A z : x$6 R y & y R z -> x$6 R z
10:  A x : A y : A z : x R y & y R z -> x R z

------------------

1:  x$6 R y$7
```

This "done" command assigns values to `x$6` and `y$7`, as can be seen on the next page.

```
Line 19
prove
x$6 R y$7 & y$7 R z$8
second part:  prove
y$7 R z$8:

1:  A z : x_1 R y_3 & y_3 R z -> x_1 R z
2:  A y : A z : x_1 R y & y R z -> x_1 R z
3:  A x : A y : A z : x R y & y R z -> x R z
4:  x_1 R y_3
5:  A x : E y : x R y
6:  y_3 R x_1
7:  y_3 R x_1 -> x_1 R y_3
8:  A y : x_1 R y == y R x_1
9:  A x : A y : x R y == y R x
10:  ~ x_1 R x_1

------------------

1:  y_3 R z$8
```

```
Enter quit to break out of interface:   gl 6; d
Line 19
prove
x$6 R y$7 & y$7 R z$8
second part:  prove
y$7 R z$8:

1:  y_3 R x_1
2:  y_3 R x_1 -> x_1 R y_3
3:  A y : x_1 R y == y R x_1
4:  A x : A y : x R y == y R x
5:  ~ x_1 R x_1
6:  A z : x_1 R y_3 & y_3 R z -> x_1 R z
7:  A y : A z : x_1 R y & y R z -> x_1 R z
8:  A x : A y : A z : x R y & y R z -> x R z
9:  x_1 R y_3
10:  A x : E y : x R y

------------------

1:  y_3 R z$8
```

. . . and this "done" command handles the last instantiable.

```
Line 17
use
x$6 R y$7 & y$7 R z$8 -> x$6 R z$8
second part, show that the desired conclusion follows
      from
x$6 R z$8:

1:  x_1 R x_1
2:  A z : x_1 R y_3 & y_3 R z -> x_1 R z
3:  A y : A z : x_1 R y & y R z -> x_1 R z
4:  A x : A y : A z : x R y & y R z -> x R z
5:  x_1 R y_3
6:  A x : E y : x R y
7:  y_3 R x_1
8:  y_3 R x_1 -> x_1 R y_3
9:  A y : x_1 R y == y R x_1
10:  A x : A y : x R y == y R x

------------------

1:  x_1 R x_1




Enter quit to break out of interface:   d
Q. E. D.

Line 1:


------------------

1:  (A x : E y : x R y) & (A x : A y : x R y == y R x)
     & (A x : A y : A z : x R y & y R z -> x R z) -> (A x
      : x R x)
```

```
Enter quit to break out of interface:
```

5 Marcel's internal language in full

In this section I will give a full account of Marcel's internal language and a discussion of its higher-order features (how it implements sets and functions).

6 Command reference for the native interface

In this section I will give the complete command reference for the native interface.