

Maude 2.0 Quick Reference

⌘ = only available in full Maude, fully parenthesized
¶ = important point to remember

Include comments:

```
--- Comment to end of line  
*** Comment to end of line
```

=====

Define modules:

```
fmod is ... endfm --- Functional module  
mod is endm --- System module  
⌘ omod is ... endom --- Object-oriented module
```

Example structure:

```
fmod is  
  <Include modules>  
  <Declare sorts>  
  <Declare operations>  
  <Define variables>  
  <Define operations>  
endfm
```

=====

Include modules:

```
protecting <module> --- No junk or confusion  
extending <module> --- Allows junk but no confusion  
including <module> --- Allows junk and confusion  
"junk" - Adding new irreducible terms to sorts in imported module  
"confusion" - Redefining existing terms in imported module
```

=====

Declare sorts:

```
sort <sort> --- Declare a single sort  
sorts <sort>+ --- Declare multiple sorts  
subsort <sort> < <sort> --- Define a relationship between two sorts  
subsorts {<sort> <}+ <sort> --- Define relationships between multiple sorts
```

=====

Declare variables:

```
var <var> : <sort> --- Declare a single variable  
vars <var>+ : <sort> --- Declare multiple variables  
var <var> : [<sort>] --- Declare a kind
```

=====

Declare operations:

```
op <op>+ : <sort>* -> <sort> {[<attrs>]}?  
  --- Declare prefix operation's signature  
op _<op>_ : <sort> <sort> <sort> -> <sort> {[<attrs>]}?  
  --- Declare infix operation's signature  
ops <op>+ : <sort>* -> <sort> {[<attrs>]}?
```

```

--- Declare multiple prefix operations' signatures
ops _<op1>__ ___<op2> __<op3>_ : <sort> <sort> <sort> -> <sort> {[<attrs>]}?
--- Declare multiple mixfix operations' signatures

```

Optional attributes:

```

ctor --- Constructor
assoc --- Associative operator
comm --- Commutative operator
{left|right}? id: <id const> --- Define left, right, or two-sided identity
constant
idem --- Idempotent operator (duplicates discarded), can't use with assoc
iter --- Allow shortening of repeated operator applications:
  s s s s s 0 -> s_^5(0)
memo --- Memoize operator – cache result and fetch later for identical calls
prec <nznum> --- Override default operator precedence (41) – lower number =
higher precedence
gather ({e|E|&}{numargs}) --- If symbol in arg's position is e, only accept
operator of lower precedence than this operator for that arg. If E, accept
operators of less than or equal precedence. If &, accept operators of any
precedence.
owise --- Use the affected equation as the default if all other possible
matches fail
format ({{d|+|-|s|t|i|n...}+ }+) --- Specify how the operator should be
printed (after 'set print format on' has been activated)
  d - Default spacing
  + - Increment global indent counter
  - - Decrement global indent counter
  s - Single space
  t - Single tab
  i - Spaces determined by global indent counter
  n - Newline
  r - Red
  g - Green
  y - Yellow
  b - Blue
  m - Magenta
  c - Cyan
  u - Underline
  ! - Bold
  o - Revert to original color and style
ditto --- Copy attributes from similar attribute into this one
strat (i...j 0) --- Specify the order of argument evaluation
frozen {{(i..j)}?} --- Freeze all or some operator arguments by preventing
reduction of those arguments
label <label> --- Label a statement
metadata "str" --- Explain a statement
nonexec --- Instruct Maude to ignore statement

```

=====

Define operations:

```

eq <expr> = <result> --- Simple equation
eq <op>({<var>:<sort> }+) = <result> --- On-the-fly variable declaration
ceq <expr1> = {true/false/<expr4>}
      if {{<expr2> /= <expr3>} |
      {not}? {<expr2> == <expr3>}}
      {/\|or|and} {<cexpr2>}
eq <expr1> = if <cexpr> then <expr2> else <expr3> fi

```

```
ceq <op>( (S:String) ! ) = T if T := "Goodbye"
```

¶ Maude's goal is to reduce an expression until it contains only constructors.

```
=====
```

Define membership axioms:

```
cmb <expr1> : <sort> if <expr2> --- <expr1> is a <sort> if <expr2> holds
mb <expr1> : <sort> --- <expr1> is a <sort>
```

```
=====
```

Defining transitions:

```
rl [<transition>] : <expr1> => <expr2> --- Unconditional rewrite law
crl [<transition>] : <expr1> => <expr2> if <expr3> --- Conditional rewrite law
```

```
=====
```

Object-oriented features:

```
class <clsid> | {<var> : <sort> ,}* var : <sort> --- Declare a class with
attributes
<objname> : <clsid> | {<val> : <var> ,}* <val> : <var> > --- Initialize an
instance of a class
subclass <clsid1> < <clsid2> --- Declare <clsid1> to be a subclass of <clsid2>
msg <op> : Oid <args> -> Msg --- Declare a "member operation"
```

Miscellaneous:

```
(<const>).<sort> --- Disambiguate an overloaded constant
```

```
=====
```

Built-in modules:

```
NAT --- Natural numbers
FLOAT --- Floating-point numbers
RAT --- Rational numbers
QID --- Quoted identifier ('Computer)
STRING --- Quoted strings ("hello")
"hello" + "world" = "helloworld"
substr("hello", 2, 3) = "llo"
find("hello jello", 2) = 2
find("hello jello", 3) = 8
rfind("hello jello", 2) = notFound
rfind("hello jello", 3) = 2
```

```
=====
```

Reduce expression:

```
red{uce}? <expr> --- Reduce an expression
red in <module> : <expr> --- Reduce an expression, using a particular module's
definition
set trace on --- Enable execution tracing
```

Rewrite expression:

```
rew [<maxtrans>] <expr> --- Rewrite an expression using up to <maxtrans>
transitions and the default algorithm
frew [<maxtrans>] <expr> --- Rewrite an expression using the fair-rewrite
algorithm that gives equal priority to all rewrite laws
```

search <expr> =>+ <result> --- Search for a sequence of at least one rewriting rule to transform <expr> into <result>
search <expr> =>! <result> --- Search for a sequence of rewriting rules to transform <expr> into the terminal (non-rewritable) <result>
show path <num> --- Trace the rewriting states that occurred to reach state <num>