Comp 317: Semantics of Programming Languages

Fifth Tutorial Sheet: Fun with Maude!

These questions ask you to try your hand at Maude programming. Maude is an interpreted language. You can start the Maude interpreter by typing `maude` at the command-line (under Windows on the CSC network, look under 'Java Apps'). You should then get the Maude prompt: `Maude>`, at which you can type in modules or commands. One useful command is `q` (for 'quit'); another is `in myFile`, which will load a file called myFile.maude from the current working directory (cwd). Other useful commands are: `pwd`, to find out what the cwd is, and `cd Dir`, which will change the cwd to a directory ('folder' in winspeak) called Dir (in the cwd). The command you can have the most fun with is `red` or `reduce`, which should be followed by a term (then a space and a dot!): this will use equations from the most-recently read module to reduce a term.

1. Save the following specification (from the third tutorial) in a file called numerals.maude (or whatever you want):

   ```
   fmod NUMERAL-EXPRESSION is

     sorts Digit NumeralExp .
     subsort Digit < NumeralExp .

     ops 0 1 : -> Digit .

     op _ _ : NumeralExp Digit -> NumeralExp .

     op _+_ : NumeralExp NumeralExp -> NumeralExp .

   endfm
   ```

   Start Maude, load your file, then type

   ```
   red 1 1 0 1 .
   ```

   at the prompt.

   Now type

   ```
   red 1 1 + 1 0 .
   ```

at the prompt — and make sense of the results!

Add precedences to the binary operators: '`[prec` n`]`' before the dot at the end of the declaration. Give a lower number to `_ _` to make it more tightly-binding. Now try the last reduction again.

2. You can have more fun if you actually have some equations, so create a file 'arithmetic.maude' with:

```
fmod  ARITHMETIC  is

  sort  Nat .

    *** numerals in unary notation
  op  0 : -> Nat .
  op  succ : Nat -> Nat .

    *** addition
  op  _+_ : Nat Nat -> Nat [prec 44] .

    *** multiplication
  op  _*_ : Nat Nat -> Nat [prec 35] .

    *** exponentiation
  op  _**_ : Nat Nat -> Nat [prec 30] .

  vars  M N : Nat .

    *** an algorithm for addition on unary numerals:
  eq  M + 0  =  M .
  eq  M + succ(N)  =  succ(M + N) .

endfm
```

And try reducing a few terms.

Give equations to define multiplication for unary numerals, and test them with a few reductions. Do the same for exponentiation.

(Just for fun: addition is repeatedly adding one; multiplication is repeatedly adding; and exponentiation is repeatedly multiplying — what's the next operation, and define it in Maude.)

3. Remember the example questions for the class test that asked you to give models for the `NUMERAL-EXPRESSION` signature, then evaluate terms in those models (i.e., calculating $h(t)$ for various terms t)? The example model given was

   - $A_{\texttt{Digit}} = \{0, 1\}$
   - $A_{\texttt{NumeralExp}} = \{0, 1, 2, \ldots\}$
   - $A_{\texttt{0}} = 0$
   - $A_{\texttt{1}} = 1$
   - $A_{\texttt{\_\_}}(x, y) = 2x + y$
   - $A_{+}(x, y) = x + y$

   It was slightly tricky but mainly tedious to evaluate terms, so let's get Maude to do the donkey work. First, let's simplify a bit by taking the carrier sets to be integers, so we can use Maude's built-in integers (in the module `INT`), and let's write `[[_]]` instead of h. We can 'code up' this model in the following Maude spec.

```
fmod  EXAMPLE-A  is

  protecting  NUMERAL-EXPRESSION .
  protecting  INT .

    *** fancy notation for the homomorphism h : T_Sigma -> A
    ***
  op  [[ _ ]] : NumeralExp -> Int .

  var  D : Digit .
  vars N N' : NumeralExp .

    *** A's implementation of the constants
    ***
  eq  [[ 0 ]]   =  0 .
  eq  [[ 1 ]]   =  1 .

    *** A's implementation of _ _
    ***
  eq  [[ N D ]]   =  2 * [[ N ]] + [[ D ]] .

    *** A's implementation of +
    ***
  eq  [[ N + N' ]]   =  [[ N ]] + [[ N' ]] .

endfm
```

   Now get Maude to check your answers to last week's questions by typing

```
red [[ (1 0 + 1 0 1 1) 1 ]] .
```

etc., at the prompt.

4. Now use Maude to code up the following model:

   - $B_{\texttt{Digit}} = \{true, false\}$
   - $B_{\texttt{NumeralExp}} = \{true, false\}$
   - $B_{0} = false$
   - $B_{1} = true$
   - $B_{\_\_}(x, y) = x \ and \ y$
   - $B_{+}(x, y) = x \ or \ y$

   and evaluate several terms in that model. (Use Maude's built-in module BOOL for truth values, and type

   ```
   show module BOOL .
   ```

   at the prompt to see what the syntax of the logical operations are.)

5. Do the same with ARITHMETIC: code up a few whacky models and reduce terms. If we've covered this in the lecture before the tutorial, play Spot the Model with the equations in ARITHMETIC.