Comp 317: Semantics of Programming Languages
Problem Sheet 4: Solutions

Solutions

1. (a) This module denotes the initial algebra, which in this case is the term algebra.

   Notes. The keyword `fmod` indicates a 'functional module', whose meaning is the initial algebra. In this case, because the module has no equations, the initial algebra is the term algebra.

   (b) $S = \{\texttt{Digit},\ \texttt{NumeralExp}\}$

$$\begin{array}{rcl} \Sigma_{[\,],\ \texttt{Digit}} & = & \{\ 0, 1\ \} \\ \Sigma_{\texttt{NumeralExp Digit},\ \texttt{NumeralExp}} & = & \{\ \_\,\_\ \} \\ \Sigma_{\texttt{NumeralExp NumeralExp},\ \texttt{NumeralExp}} & = & \{+\} \\ \Sigma_{w,s} & = & \emptyset \quad \text{for all other } w, s \end{array}$$

   (c) Carrier sets are terms

$$T_{\Sigma,\texttt{Digit}} = \{\texttt{0}, \texttt{1}\}$$

$$T_{\Sigma,\texttt{Numeral}} = \{\texttt{0}, \texttt{1}, \texttt{00}, \texttt{01}, \texttt{10}, \ldots, \texttt{0} + \texttt{101}, \ldots\}\ .$$

   and, for example, $T_{\Sigma,+}(t_1, t_2) = t_1{+}t_2$ (as a string).

   (d) $h_{\texttt{Digit}}$ is the familiar semantics of Digits, and $h_{\texttt{NumeralExp}}$ that of numerals (with addition thrown in)!

   (e) i. $h_{\texttt{N}}(\texttt{001}) = 2(h_{\texttt{N}}(\texttt{00})) + h_{\texttt{D}}(1) = 2(2(h_{\texttt{N}}(\texttt{0}) + h_{\texttt{D}}(\texttt{0}))) + 1 = 2(0 + 0) + 1 = 1$.

   ii. 25

   iii. 6

   iv. 10

   v. $h_{\texttt{N}}((\texttt{10} + \texttt{1011})\texttt{1}) = 2(h_{\texttt{N}}(\texttt{10} + \texttt{1011})) + h_{\texttt{D}}(\texttt{1})$
   $= 2(h_{\texttt{N}}(\texttt{10}) + h_{\texttt{N}}(\texttt{1011})) + 1 = \cdots = 2(4 + 11) + 1 = 31$

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

(a bit wierd).

i. 1
ii. 3
iii. 1
iv. 2
v. 2

(g)
```
fmod NUMERAL-EXPRESSION-SEMANTICS is

    protecting NUMERAL-EXPRESSION .
    protecting INT .

    op  [[ _ ]]  : Digit -> Int .
    op  [[ _ ]]  : NumeralExp -> Int .

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

    eq  [[ 0 ]]  =  0 .
    eq  [[ 1 ]]  =  1 .
    eq  [[ N D ]]  =  2 * [[ N ]]  + [[ D ]] .
    eq  [[ N + N' ]]  =  [[ N ]] + [[ N' ]] .

endfm
```