COMP0020 Functional Programming

Lecture 12

Recursive Algebraic Types

(RATs!)

Contents

- Motivation : why do we need recursive algebraic types ?
  - Type domains revisited : the list
- Example RATs and Polymorphic RATs :
  - Lists
  - Trees
- Functions that operate on sorted trees

Motivation

- Using algebraic types we can enumerate the legal values of a new type
- But what if we want more values than we can enumerate ?
- How could we ever define our own types that were as powerful as num or [char]
- Easy ! — use recursion

Type Domains Revisited

- The type [char] has many legal values : the empty list [ ], the list with one char ['A'], and so on.
- However, the definition of a list is both simple and finite :
  - A list of char may be either the empty list
  - Or a char together with a list of char
- This is the basis of the definition of algebraic types that potentially contain infinitely many values

Example RATs/PRATs

- A simple recursive algebraic type :
  mylistchar ::= MyNil | MyCons char mylistchar
- A **polymorphic** recursive algebraic type :
  mylist * ::= Empty | Cons * (mylist *)
- A function that operates on a polymorphic recursive algebraic type :

$$
\begin{aligned}
&myhd :: (mylist \ *) -> \ * \\
&myhd\ Empty \qquad\qquad = error\ \text{"}head\ of\ empty\ mylist\text{"} \\
&myhd\ (Cons\ x\ xs) \quad = x
\end{aligned}
$$

$$main \ = \ myhd\ (Cons\ 'A'\ (Cons\ 'B'\ (Cons\ 'C'\ Empty)))$$

Example RATs/PRATs (2)

- Example tree type :

```
bintree * ::= Emptytree | Node * (bintree *) (bintree *)

rightmost :: bintree * -> *
rightmost Emptytree = error "rightmost of empty tree"
rightmost (Node x lt Emptytree) = x
rightmost (Node x lt rt) = rightmost rt

x = (Node 3 (Node 4 Emptytree Emptytree) Emptytree)
main = rightmost x
```

Functions on sorted trees

- Adding an element to a **sorted** tree of numbers :

$$
\begin{array}{lllll}
inserttree & :: & bintree\ num \rightarrow num \rightarrow bintree\ num & & \\
inserttree & & Emptytree\ x & = & Node\ x\ Emptytree\ Emptytree \\
inserttree & & (Node\ v\ lt\ rt)\ x & = & Node\ v\ (inserttree\ lt\ x)\ rt,\ if\ (x <= v) \\
 & & & = & Node\ v\ lt\ (inserttree\ rt\ x),\ otherwise
\end{array}
$$

Functions on sorted trees (2)

- Membership of a **sorted** tree of numbers :

$$
\begin{aligned}
&membertree :: bintree\ num -> num -> bool\\
&membertree\ Emptytree\ x &&= False\\
&membertree\ (Node\ v\ lt\ rt)\ x &&= True,\ if\ (v = x)\\
& &&= (membertree\ lt\ x),\ if\ (x <= v)\\
& &&= (membertree\ rt\ x),\ otherwise
\end{aligned}
$$

Function on an unsorted tree

- Compare with membership of an **unsorted** tree of numbers :

$$
\begin{array}{lcl}
\mathit{memberutree} :: \mathit{bintree}\ \mathit{num} \rightarrow \mathit{num} \rightarrow \mathit{bool} & & \\
\mathit{memberutree}\ \mathit{Emptytree}\ x & = & \mathit{False} \\
\mathit{memberutree}\ (\mathit{Node}\ v\ lt\ rt)\ x & = & \mathit{True},\ \mathit{if}\ (v = x) \\
 & = & (\mathit{memberutree}\ lt\ x) \vee (\mathit{memberutree}\ rt\ x),\ \mathit{otherwise}
\end{array}
$$

Functions on sorted trees (3)

- Removing an element from a sorted tree of numbers :


```
subtree :: bintree num -> num -> bintree num
subtree Emptytree x      = Emptytree
subtree (Node v lt rt) x = rt, if (v = x) & (lt = Emptytree)
                         = Node v (subtree lt x) rt, if (x < v)
                         = Node v lt (subtree rt x), if (x > v)
                         = Node new (subtree lt new) rt, otherwise
                           where
                           new = rightmost lt
```

Summary

- Motivation : why do we need recursive algebraic types ?
  - Type domains revisited : the list
- Example RATs and Polymorphic RATs :
  - Lists
  - Trees
- Functions that operate on sorted trees

END OF LECTURE