User-defined Datatypes

Haskell allows the definition of new datatypes:

\[
data \text{Datatype} \ a_1 \ldots a_n = \text{constr}_1 \mid \ldots \mid \text{constr}_m
\]

where
1. \text{Datatype} is the name of a new type constructor of arity \( n \geq 0 \),
2. \( a_1, \ldots, a_n \) are distinct type variables representing the arguments of \text{DatatypeName} and
3. \( \text{constr}_1, \ldots, \text{constr}_m \) (\( m \geq 1 \)) describe the way in which elements of the new datatype are constructed.

User-defined Datatypes...

Each \text{constr} can take one of two forms:

1. \( \text{Name type}_1 \ldots \text{type}_r \) where \text{Name} is a previously unused constructor function name (i.e. an identifier beginning with a capital letter). This declaration introduces \text{Name} as a new constructor function of type:

\[
type_1 \rightarrow \ldots \rightarrow type_r \rightarrow \text{Datatype} \ a_1 \ldots a_n
\]

2. \( \text{type}_1 \oplus \text{type}_2 \) where \( \oplus \) is a previously unused constructor function operator (i.e. an operator symbol beginning with a colon). This declaration introduces \( \oplus \) as a new constructor function of type:

\[
type_1 \rightarrow type_2 \rightarrow \text{Datatype} \ a_1 \ldots a_n
\]

User-defined Datatypes...

The following definition introduces a new type \text{Day} with elements \text{Sun}, \text{Mon}, \text{Tue},...:

\[
data \text{Day} = \text{Sun}\mid \text{Mon}\mid \text{Tue}\mid \text{Wed}\mid \text{Thu}\mid \text{Fri}\mid \text{Sat}
\]

Simple functions manipulating elements of type \text{Day} can be defined using pattern matching:

\[
\begin{align*}
\text{what\_shall\_I\_do} \ \text{Sun} & = \ "relax" \\
\text{what\_shall\_I\_do} \ \text{Sat} & = \ "go \ shopping" \\
\text{what\_shall\_I\_do} \ _ & = \ "go \ to \ work"
\end{align*}
\]
Another example uses a pair of constructors to provide a representation for temperatures which may be given using either of the centigrade or fahrenheit scales:

```haskell
data Temp = Centigrade Float | Fahrenheit Float
```

```haskell
freezing :: Temp -> Bool
freezing (Centigrade temp) = temp <= 0.0
freezing (Fahrenheit temp) = temp <= 32.0
```

Datatype definitions may also be recursive. The following example defines a type representing binary trees with values of a particular type at their leaves:

```haskell
data Tree a = Lf a | Tree a :+: Tree a
```

For example,

```haskell
(Lf 12 :+: (Lf 23 :+: Lf 13)) :+: Lf 10
```

has type `Tree Int` and represents the binary tree:

```
  10
   / \
  13  23
   /
  12
```

Calculate the list of elements at the leaves of a tree traversing the branches of the tree from left to right.

```haskell
leaves :: Tree a -> [a]
leaves (Lf l) = [l]
leaves (l :+: r) = leaves l ++ leaves r
```

Using the binary tree above as an example:

```haskell
[12, 23, 13, 10]
(24 reductions, 73 cells)
```

These slides were derived directly from the Gofer manual.

Functional programming environment, Version 2.20
copyright Mark P. Jones 1991.

A copy of the Gofer manual can be found in

```
/home/cs520/2003/gofer/docs/goferdoc.ps
```