CSc 372

Comparative Programming Languages

28: Haskell — Data Types

Christian Collberg

collberg@cs.arizona.edu

Department of Computer Science

University of Arizona

Copyright © 2004 Christian Collberg
Haskell allows the definition of new datatypes:

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

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.
Each *constr* can take one of two forms:

1. *Name* \( type_1 \ldots type_r \) where *Name* is a previously unused constructor function name (i.e. an identifier beginning with a capital letter). This declaration introduces *Name* as a new constructor function of type:

\[
type_1 \rightarrow \ldots \rightarrow type_r \rightarrow Datatype \ a_1 \ldots a_n
\]

2. \( type_1 \oplus type_2 \) 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:

\[
type_1 \rightarrow type_2 \rightarrow Datatype \ a_1 \ldots a_n
\]
The following definition introduces a new type `Day` with elements `Sun`, `Mon`, `Tue`,...:

```haskell
data Day = Sun | Mon | Tue | Wed | Thu | Fri | Sat
```

Simple functions manipulating elements of type `Day` can be defined using pattern matching:

```haskell
what_shall_I_do Sun = "relax"
what_shall_I_do Sat = "go shopping"
what_shall_I_do _ = "go to work"
```
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

freezing :: Temp -> Bool
freezing (Centigrade temp) = temp <= 0.0
freezing (Fahrenheit temp) = temp <= 32.0
```
User-defined Datatypes...

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

```
data Tree a = Lf a | Tree a :^: Tree a
```

- For example,

```
(Lf 12 :^: (Lf 23 :^: Lf 13)) :^: Lf 10
```

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

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

User-defined Datatypes...

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

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

Using the binary tree above as an example:

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

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.