Computer Science 312

Defining New Data Types
What Is a Data Type?

• A set of values and operations:
  – Constructors
  – Type recognition predicate
  – Selector for alternative forms
  – Accessors
Example: \textit{bintree}

\[
\textit{bintree} \ ::= \ \textit{number} \mid (\textit{symbol} \ \textit{bintree} \ \textit{bintree})
\]

- Constructors (for different kinds of bintrees)
- Type recognition predicate
- Selector for alternative forms (leaf or interior node)
- Accessors (for getting leaf or interior components)
Use `define-datatype`  

\[\text{<bintree> ::= <number> | (<symbol> <bintree> <bintree>)}\]  

(define-datatype bintree bintree?
  (leaf-node
    (datum number?))
  (interior-node
    (key symbol?)
    (left bintree?)
    (right bintree?)))

`define-datatype` is a special form included in the `eopl` language of Dr Racket (`#lang eopl`)
What’s Going on Here?

• Define a data type `bintree` with a predicate `bintree`?

  `(define-datatype bintree bintree?)`

• A `bintree` is either
  – A leaf node consisting of a datum which is a number
    `(leaf-node
      (datum number?))`
  – An interior node consisting of a key that is a symbol, a left child that is a `bintree`, and a right child that is a `bintree`
    `(interior-node
      (key symbol?)
      (left bintree?)
      (right bintree?))`
What’s Going on Here?

• Creates a data type with the following interface:

  (define-datatype bintree bintree?)

• A one-argument procedure for constructing a leaf node, with a test to make sure that the argument is a number

  (leaf-node
   (datum number?))

• A three-argument procedure for building an interior node, with a test for the first arg being a symbol and the other two args being a bintree

  (interior-node
   (key symbol?)
   (left bintree?)
   (right bintree?))
General Format

(define-datatype type-name predicate-name
   {(variant-name {(field-name predicate)})}*{)+)

(define-datatype bintree bintree?
   (leaf-node
      (datum number?))
   (interior-node
      (key symbol?)
      (left bintree?)
      (right bintree?)))

<bintree> ::= <number> | (<symbol> <bintree> <bintree>)
Creating bintrees

> (define tree-a (interior-node 'a (leaf-node 1) (leaf-node 2)))

> (define tree-b (interior-node 'b (leaf-node 3) (leaf-node 4)))

> (define tree-c (interior-node 'c tree-a tree-b))

Supports type checking:

(leaf-node 'foo)
Selecting a Variant With cases

(define leaf-sum
  (lambda (tree)
    (cases bintree tree
      (leaf-node (datum) datum)
      (interior-node (key left right)
        (+ (leaf-sum left) (leaf-sum right)))))))
General Format of **cases**

(cases type-name expression
   {(variant-name (field-name)* consequent)}*)

(cases bintree tree
   (leaf-node (datum) datum)
   (interior-node (key left right)
      (+ (leaf-sum left) (leaf-sum right)))))