CMSC 330: Organization of Programming Languages

OCaml Data Types

OCaml Data

- So far, we've seen the following kinds of data
  - Basic types (int, float, char, string)
  - Lists
    - One kind of data structure
    - A list is either [ ] or h::t, deconstructed with pattern matching
  - Tuples and Records
    - Let you collect data together in fixed-size pieces
  - Functions

- How can we build other data structures?
  - Building everything from lists and tuples is awkward

User Defined Types

- type can be used to create new names for types
  - Useful for combinations of lists and tuples

- Examples
  - `type my_type = int * (int list)`
  - `let (x:my_type) = (3, [1; 2])`

  - `type my_type2 = int*char*(int*float)`
  - `let (y:my_type2) = (3, 'a', (5, 3.0))`

(User-Defined) Variants

```
type coin = Heads | Tails

let flip x =
  match x with
    Heads -> Tails
  | Tails -> Heads

let rec count_heads x =
  match x with
    [] -> 0
  | (Heads::x') -> 1 + count_heads x'
  | (_::x') -> count_heads x'
```

In simplest form: Like a C **enum**

Basic pattern matching resembles C **switch**

Combined list and variant patterns possible

Constructing and Destructing Variants

- Syntax
  - **`type t = C1 | … | Cn`**
  - the ***`Ci`*** are called constructors
    - Must begin with a capital letter
- Evaluation
  - A constructor ***`Ci`*** is already a value
  - Destructing a value ***`v`*** of type ***`t`*** is done by pattern matching on ***`v`*** ; the patterns are the constructors ***`Ci`***
- Type Checking
  - ***`Ci`*** : ***`t`*** (for each ***`Ci`*** in ***`t`***'s definition)

Data Types: Variants with Data

- We can define variants that "carry data" too
  - Not just a constructor, but a constructor *plus values*

```
type shape =
   Rect of float * float (* width*length *)
 | Circle of float       (* radius *)
```

- **`Rect`** and **`Circle`** are constructors
  - where a **`shape`** is either a **`Rect(`*`w`*`,`*`l`*`)`**
    - for any floats ***`w`*** and ***`l`***
  - or a **`Circle`** ***`r`***
    - for any float ***`r`***

Data Types (cont.)

```
let area s =
  match s with
      Rect (w, l) -> w *. l
    | Circle r -> r  *. r *. 3.14
;;
area (Rect (3.0, 4.0));; (* 12.0  *)
area (Circle 3.0);;      (* 28.26 *)
```

- Use pattern matching to deconstruct values
  - Can bind pattern values to data parts
- Data types are *aka* algebraic data types and tagged unions

Data Types (cont.)

```
type shape =
    Rect of float * float (* width*length *)
  | Circle of float       (* radius *)

let lst = [Rect (3.0, 4.0) ; Circle 3.0]
```

- What's the type of `lst`?
  - **`shape list`**
- What's the type of `lst`'s first element?
  - **`shape`**