CSE 130 : Winter 2012
Programming Languages

Lecture 3: 
Datatypes

Ranjit Jhala
UC San Diego

Wednesday, January 18, 2012
What about more complex data?

Many kinds of expressions:

1. Simple
2. Variables
3. Functions
What about more complex data?

- We’ve seen some **base** types and values:
  - Integers, Floats, Bool, String etc.

- Some ways to **build** up types:
  - Products (tuples), records, “lists”
  - Functions

- Design Principle: **Orthogonality**
  - Don’t clutter **core language** with stuff
  - Few, powerful orthogonal building techniques
  - Put “**derived**” types, values, functions in **libraries**
What about more complex data?

- We’ve seen some **base** types and values:
  - Integers, Floats, Bool, String etc.

- Some ways to **build** up types:
  - Products (tuples), records, “lists”
  - Functions

- Time for an IN CLASS EXERCISE!
Next: Building datatypes

Three key ways to build complex types/values

1. “Each-of” types
Value of T contains value of T1 and a value of T2

2. “One-of” types
Value of T contains value of T1 or a value of T2

3. “Recursive”
Value of T contains (sub)-value of same type T
Suppose I wanted ...

... a program that processed lists of attributes

- Name (string)
- Age (integer)
- DOB (int-int-int)
- Address (string)
- Height (float)
- Alive (boolean)
- Phone (int-int)
- email (string)

Many kinds of attributes (too many to put in a record)
- can have multiple names, addresses, phones, emails etc.

Want to store them in a list. Can I?
Constructing Datatypes

\[ \text{type } \mathfrak{t} = \mathsf{C}_1 \text{ of } t_1 \mid \mathsf{C}_2 \text{ of } t_2 \mid \ldots \mid \mathsf{C}_n \text{ of } t_n \]

\(\mathfrak{t}\) is a new datatype.

A value of type \(\mathfrak{t}\) is either:

- a value of type \(t_1\) placed in a box labeled \(\mathsf{C}_1\)
- a value of type \(t_2\) placed in a box labeled \(\mathsf{C}_2\)
- \(\ldots\)
- a value of type \(t_n\) placed in a box labeled \(\mathsf{C}_n\)
Suppose I wanted ...

Attributes:

- Name (string)
- Age (integer)
- DOB (int-int-int)
- Address (string)
- Height (real)
- Alive (boolean)
- Phone (int-int)
- email (string)

```plaintext
type attrib =
    Name of string
| Age of int
| DOB of int*int*int
| Address of string
| Height of float
| Alive of bool
| Phone of int*int
| Email of string;
```

Wednesday, January 18, 2012
Creating Values

How to create values of type `attrib`?

```ocaml
# let a1 = Name "Ranjit";;
val x : attrib = Name "Ranjit"

# let a2 = Height 5.83;;
val a2 : attrib = Height 5.83

# let year = 1977 ;;
val year : int = 1977

# let a3 = DOB (9,8,year) ;;
val a3 : attrib = DOB (9,8,1977)

# let a_l = [a1;a2;a3];;
val a3 : attrib list = ...
```

```
type attrib =
  Name of string
| Age of int
| DOB of int*int*int
| Address of string
| Height of float
| Alive of bool
| Phone of int*int
| Email of string;;
```
One-of types

- We’ve defined a “one-of” type named `attrib`.
- Elements are one of:
  - string,
  - int,
  - int*int*int,
  - float,
  - bool ...

- Can create uniform `attrib` lists

- Suppose I want a function to print `attribs`...

```
datatype attrib =
  Name of string
| Age of int
| DOB of int*int*int
| Address of string
| Height of real
| Alive of bool
| Phone of int*int
| Email of string;
```
How to tell what's in the box?

```
type attrib =
   Name of string
| Age of int
| DOB of int*int*int
| Address of string
| Height of float
| Alive of bool
| Phone of int*int
| Email of string;;
```

match e with
   Name s -> e1
| Age i -> e2
| DOB (m,d,y) -> e3
| Address addr -> e4
| Height h -> e5
| Alive b -> e6
| Phone (a,n) -> e7
| Email e -> e8

Pattern-match expression: check if e is of the form ...  

- On match:  
  - value in box bound to pattern variable  
  - matching result expression is evaluated

- Simultaneously test and extract contents of box
match-with is an Expression

```
match e with
    Name s -> e1
| Age i -> e2
| DOB (m,d,y) -> e3
| Address addr -> e4
| Height h -> e5
| Alive b -> e6
| Phone (a,n) -> e7
| Email e -> e8
```
**match-with** is an Expression

```
match e with
    C1 x1 -> e1
| C2 x2 -> e2
| ...
| Cn xn -> en
```

Type rules?

- \( e_1, e_2, \ldots, e_n \) must have same type
- Which is type of whole expression
Benefits of **match-with**

1. Simultaneous **test-extract-bind**

2. Compile-time checks for:
   - missed cases: ML warns if you miss a `t` value
   - redundant cases: ML warns if a case never matches
What about “Recursive” types?

```plaintext
type int_list =
    Nil
| Cons of int * int_list
```

Think about this! What are values of `int_list`?

Cons(1,Cons(2,Cons(3,Nil)))  Cons(2,Cons(3,Nil))  Cons(3,Nil)  Nil

```
```
Cons
1,
```
Cons
2,
```
Cons
3,
Nil
Lists aren’t built-in!

Lists are a derived type: built using elegant core!

1. Each-of
2. One-of
3. Recursive

:: is just a pretty way to say “Cons”
[] is just a pretty way to say “Nil”
Some functions on Lists: Length

let rec len l =
  match l with
  | Nil -> 0
  | Cons(_,t) -> 1 + (len t)

Base pattern

Inductive Expression

Ind pattern

Base Expression

Matches everything, no binding

Pattern-matching in order
- Must match with Nil
Some functions on Lists: Append

let rec append (l1,l2) =

• Find the right induction strategy
  - Base case: pattern + expression
  - Induction case: pattern + expression

Well designed datatype gives strategy
Some functions on Lists: \textbf{Max}

\texttt{let rec max xs =}

- **Base pattern**
- **Ind pattern**

\begin{itemize}
\item Find the right \textbf{induction} strategy
  \begin{itemize}
  \item Base case: pattern + expression
  \item Induction case: pattern + expression
  \end{itemize}
\end{itemize}

Well designed datatype gives strategy

Wednesday, January 18, 2012
null, hd, tl are all functions ...

**Bad ML style:** More than aesthetics!

Pattern-matching better than test-extract:

- **ML checks all cases covered**
- **ML checks no redundant cases**
- ...at *compile*-time:
  - fewer errors (crashes) during execution
  - get the bugs out ASAP!
Another Example: Calculator

We want an arithmetic calculator to evaluate expressions like:

- $4.0 + 2.9 = 6.9$
- $3.78 - 5.92 = -2.14$
- $(4.0 + 2.9) \times (3.78 - 5.92) = -14.766$

Q: What is a ML datatype for such expressions?
Another Example: Calculator

We want an arithmetic calculator to evaluate expressions like:

• \(4.0 + 2.9 = 6.9\)
• \(3.78 - 5.92 = -2.14\)
• \((4.0 + 2.9) \times (3.78 - 5.92) = -14.766\)

What's a ML function for evaluating such expressions?
Random Art from Expressions

PA #2

Build more funky expressions, evaluate them, to produce:

Wednesday, January 18, 2012