Today: 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

Suppose I wanted ...

... a program that processed lists of attributes

- Name (string)
- Age (integer)
- DOB (int-int-int)
- Address (string)
- Height (real)
- 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

datatype t = C1 of t1 | C2 of t2 | ... | Cn of tn

- t is a new datatype.
- A value of type t is either:
  - a value of type t1 placed in a box labeled C1
  - or a value of type t2 placed in a box labeled C2
  - or ...
  - or a value of type tn placed in a box labeled Cn

Suppose I wanted ...

Attributes:

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

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;

What about more complex data?

- We’ve seen some base types and values:
  - Integers, Reals, 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
Creating Values

How to create values of type `attrib`?

```plaintext
val x = Name "Ranjit";
val a1 = Height 5.83;
val a2 = Age 1977;
val a3 = DOB (9,8,1977);
val a = [a1,a2,a3];
```

One-of types

- We've defined a "one-of" type named `attrib`
- Elements are one of:
  - String,
  - int,
  - int*int,int,
  - real,
  - bool ...

- Can create uniform `attrib` lists
- Suppose I want a function to print `attrib`s...

How to tell what's in the box?

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

- On match:
  - value in box bound to pattern variable
  - corresponding result expression is evaluated
- Simultaneously test and extract contents of box

Case-of is an Expression

Type rules?

- `e1, e2, ..., en` must have same type
- Which is type of whole expression

Benefits of case-of

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?

```ml
datatype int_list =
  Nil of unit
| Cons of (int * int_list)
```

Think about this! What are values of int_list:
- `Cons(1, Cons(2, Nil))`
- `Nil`
- `Cons(2, Nil)`

Lists aren’t built-in!

```ml
datatype int_list =
  Nil
| Cons of int * int_list
```

Lists are a “derived” type, built using elegant core!
1. Each-of
2. One-of
3. Recursive
   - `[]` is just a pretty way to say “Nil”
   - `::` is just a pretty way to say “Cons”

Some functions on Lists: Length

```ml
fun len l =
  case l of
  Nil => 0
| Cons(_,t) => 1 + (len t)
```

- Matches everything, no binding
- Pattern-matching in order
  - Must match with `Nil`

Some functions on Lists: Append

```ml
fun append (l1, l2) =
  case l1 of
  Nil => l2
| Cons(_,t) => Cons(t, append(l1, l2))
```

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

- Well-designed datatype gives strategy

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) * (3.78 - 5.92) = -14.766`

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

We want an arithmetic calculator to evaluate expressions like:

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

What's a ML function for evaluating such expressions?

```ml
fun eval e =
  case e of
    Const r => r
    Var (e, e2) => (eval e) + (eval e2)
    Mul (e, e2) => (eval e) \times (eval e2)
```

Functioning recursively in ML

Example: Factorial

```ml
fun fac n =
  if n = 0
  then 0
  else n \times fac (n-1)
```

Example: Clone

```ml
fun clone (x, n) =
  if n = 0
  then []
  else x :: clone(x, n-1)
```

Example: Interval

```ml
fun interval (x, j) =
  if j > 3
  then []
  else x :: interval(+1, j)
```
**Example: List Maximum**

Find maximum element in +ve int list

```
fun max (x,y) = if x > y then x else y;
fun listMax l =
  let
    fun helper (max,l) =
      if (null l)
      then max
      else helper (max (?h::t),l)
  in
    helper (0, l)
  end;
```

**Example: List Maximum**

Find maximum element in +ve int list... in a more ML-ish way

```
fun max (x,y) = if x > y then x else y
fun listMax l =
  let
    fun helper ([],[]) = []
    | helper ([],h::t) = helper(max(h,t),t)
    | helper (h::t,[]) = h::t
  in
    helper (0, l)
  end;
```