Recap

- Variables are names for values
  - Environment: dictionary/phonebook
  - Most recent binding used
  - Entries never changed, new entries added

- Environment frozen at fun definition
  - Re-binding variables cannot change a function
  - Same I/O behavior at every call

Recap: Environments

“Phone book”
- Variables = “names”
- Values = “phone number”

1. Evaluate:
   Find and use most recent value of variable

2. Extend: let x = e ;;
   Add new binding at end of “phone book”

Function bindings

Functions are values, can bind using val

let fname = fun x -> e ;;

Problem: Can’t define recursive functions!
- fname is bound after computing rhs value
- no (or “old”) binding for occurrences of fname inside e

let rec fname x = e ;;

Occurences of fname inside e bound to “this” definition

let rec fac x = if x <= 1 then 1 else x*fac (x-1)
Local bindings

So far: bindings that remain until a re-binding (“global”)

Local, “temporary” variables are useful inside functions
• Avoid repeating computations
• Make functions more readable

Let-in is an expression!

Evaluating let-in in env E:
1. Evaluate expr e1 in env E to get value \( v : t \)
2. Use extended \( E \) [\( x := v : t \)] (only) to evaluate e2

Nested bindings

Evaluating let-in in env E:
1. Evaluate expr e1 in env E to get value \( v : t \)
2. Use extended \( E \) [\( x := v : t \)] to evaluate e2
Nested bindings

```ocaml
let
  x = 10
in
let let
  x = 10
  in
  let
    y = 20
    in
    x * y
;;
```

Correct Formatting

### 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

### 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

\[ t = C_1 \text{ of } t_1 \mid C_2 \text{ of } t_2 \mid \ldots \mid C_n \text{ of } t_n \]

\( t \) is a new datatype.

A value of type \( t \) is either:
- a value of type \( t_1 \) placed in a box labeled \( C_1 \)
- a value of type \( t_2 \) placed in a box labeled \( C_2 \)
- …
- a value of type \( t_n \) placed in a box labeled \( C_n \)

Creating Values

How to create values of type \( attrib \)?

```ocaml
let a1 = Name "Ranjit";;
val a1 : 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)
```

```ocaml
let a_l = [a1;a2;a3];;
val a_l : attrib list = ...
```
One-of types

- We’ve defined a “one-of” type named `attrib`
  
  ```
  datatype 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 ;
  ```

- Can create uniform `attrib` lists

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

How to tell what’s in the box?

```
match e with
  Names -> 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
```

Type rules?

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

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

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

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?

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]

Lists aren’t built-in!

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 “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)

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

let rec append (l1, l2) =

Base pattern
Ind pattern

• 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

Q: What's 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) * (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: