CSE 341:
Programming Languages

Spring 2005

Lecture 11 — “Objects” in ML; Mutual Recursion

Key idioms with closures

- Create similar functions
- Pass functions with private data to iterators (map, *fold*, ...)
- Combine functions

- Provide an ADT
- As a *callback* without the "wrong side" specifying the environment.
- Partially apply functions ("currying")

Provide an ADT

A record of functions is much like an object.

Free variables are prviate variables.

Our “set” example is fancy stuff, but you should be able to understand it.

```
datatype set = S of {add:int -> set, member:int -> bool}
val empty_set = fn : unit -> set
```

Callbacks

A common idiom: Library takes a function to apply later, when an *event* occurs. Examples:

- When a key is pressed, a mouse moved, etc.
- When a packet arrives from the network

The function may be a filter ("I want the packet") or return a result ("draw a line"), etc.

Library may accept multiple callbacks. Different callbacks may need different private state with different types.

Fortunately, the type of a function does not depend on the type of free variables.

Note: This is why Java added anonymous inner classes (for "event listeners").

Mutual Recursion

We haven't yet seen how multiple functions can recursively call each other? (Why can't we do this with what we have?)

ML uses the keyword `and` to provide different *scope* rules. Example:

```
fun even i = if i=0 then true  else odd (i-1)
and odd  i = if i=0 then false else even (i-1)
```

Roughly extends the binding form for functions from `fun f1 x1 = e1` to `fun f1 x1 = e1 and f2 x2 = e2 and ...  and fn xn = en`.

Actually, you can have `val` bindings too, but bindings being defined are in scope only inside function bodies. (Why?)

Syntax gotcha: Easy to forget that you write `and fi xi = ei`, not `and fun fi xi = ei`.

Mutual Recursion Idioms

1. Encode a state machine (see `product_sign` example)
   - Sometimes easier to understand than explicit state values.
2. Process mutually recursive types, example:

```
datatype webtext = Empty
                 | Link of webpage * string * webtext
                 | Word of string * webtext
and webpage = Found of string * webtext
            | Unfound of string
```

   A function "crawl for word" is inherently mutually recursive. (You could make a datatype for "webtext or webpage", but that's ugly.)

   Problem: the Web has *cycles*, which (sigh) is a common need for mutation in ML.

   Unproblem: When crawling, we don't want cycles (use Unfound if we have seen the page before).