Functional Programming

Functional Programming Style
- Write many small functions (2-liners)
- Each loop corresponds to 1 function
- No assignment, only function calls
- Write base cases of recursion
  - case for empty list, maybe for atoms
  - maybe case for singleton list
- Write recursive cases
  - case(s) for nonempty list

Example: Nesting of Parentheses
- Base case empty list: 1
- Base case atoms: 0
- Recursive case: max (1+car, cdr)
- Finished function

```scheme
(define (nest x)
  (cond ((null? x) 1)
       ((not (pair? x)) 0)
       (else (max (+ 1 (nest (car x)))
                  (nest (cdr x))))))
```
Example: Integer Equation

- Given: lengths l1, l2, l3, len
- Can len be constructed from pieces of lengths l1, l2, and l3?

```scheme
(define (test len l1 l2 l3)
  (if (<= len 0) (= len 0)
      (or (test (- len l1) l1 l2 l3)
          (test (- len l2) l1 l2 l3)
          (test (- len l3) l1 l2 l3)))
)
```

ML vs. Scheme

- **Scheme**
  - primitive syntax
  - dynamically typed
  - Lists as built-in data type
- **ML**
  - fancy syntax
  - statically typed, type inference
  - recursive data types

History of ML

- Developed at Edinburgh (early '80s) as Meta-Language for a program verification system
- Now a general purpose language
- Development of ML 2000
- CAML (INRIA), Moby (U Chicago), F#
Features of ML

- Strong, static typing
- Type inference
- Recursive data types
- Parametric polymorphism
- Pattern matching
- Exception handling

Syntax Comparison

- Scheme
  
  (define (fac n)
   (if (= n 0) 1
     (* n (fac (- n 1)))))

- ML
  
  fun fac (n) =
    if n = 0 then 1
    else n * fac (n - 1)

Typing

- Scheme
  - types are checked at run time (e.g., fac could be called with a list as argument)

- ML
  - types are checked by compiler (fac must be called with integer as arg.)
    - compiler infers types
    - no run time type errors (core dumps)
Interactive System
- User enters one definition at a time
- Input expressions define variable \textit{it}
- Feels like interpreter
- ML compiles each definition

Lists
- Empty list
  \texttt{nil}
- Cons
  ::
- List syntax
  1 :: 2 :: 3 :: \texttt{nil}
  \{1, 2, 3\}
- Lists are homogenous

Recursive Data Types
- Enumeration types
  \texttt{datatype Color = red | blue | green}
- Integer trees
  \texttt{datatype Tree = Leaf of int | Node of Tree * Tree}
Pattern Matching

- fun foo red = 0
  | foo blue = 1
  | foo green = 2;
val foo = fn : Color -> int

- fun max (i, j: int) = 
  if i > j then i else j;
val max = fn : int * int -> int

- fun height (Leaf _) = 0
  | height (Node (l, r)) = 
    1 + max (height l, height r);
val height = fn : Tree -> int

Parametric Polymorphism

- fun id x = x;
val id = fn : 'a -> 'a

- datatype 'a Tree = Leaf of 'a
  | Node of 'a Tree * 'a Tree;
- fun height (Leaf _) = 0
  | height (Node (l, r)) = 
    1 + max (height l, height r);
val height = fn : 'a Tree -> int

More Examples

- fun length nil = 0
  | length (_::t) = 1 + length t;
val length = fn : 'a list -> int

- length [1, 2, 3];
val it = 3 : int

- height (Node (Leaf 1,
  Node (Leaf 2, Leaf 3)));
val it = 2 : int

- id 42;
val it = 42 : int

- id [1, 2, 3];
val it = [1,2,3] : int list
Tuples and Unit

- `(1, 2);`
  val it = (1,2) : int * int
- `fun add (x : int, y) = x + y;
  val add = fn : int * int -> int`
- `();
  val it = () : unit

- **Tuples have at least two elements**
- **Extra parentheses don’t count**
- **All functions have exactly one argument!**

Currying

- `fun add x y = x + y : int;
  val add = fn : int -> int -> int`
- `val add = fn x => fn y => x + y : int;
  val add = fn : int -> int -> int`
- `val add1 = add 1;
  val add1 = fn : int -> int`
- `val x = add1 10;
  val x = 11 : int`

Summary

- **ML is compiled**
- **Fancy type system with type inference**
- **No OO (makes type inference undecidable)**
- **Quite efficient**
  - average probably about half the speed of C
  - CAML can be 10 times faster than C
- **Has been used for systems programming**
- **Some use in industry, especially in Europe**
- **Good for parallel programming**
Applications of Functional Languages

- LISP
  - Artificial Intelligence, Emacs, MACSYMA
- ML
  - Several theorem provers, networking code [http://www.cs.cmu.edu/~fox/foxnet.html]
- Erlang
  - Ericson phone switches [http://www.erlang.org]
- Sisal
  - number crunching language from LLNL