Lecture 12: Functional Languages

CSC 131
Fall, 2006

Lazy vs. Eager Evaluation

- Eager: Evaluate operand, substitute operand value in for formal parameter, and evaluate.
- Lazy: Substitute operand in for formal parameter and evaluate body, evaluating operand only when needed.
  - Each actual parameter evaluated either not at all or only once!
  - Like left-most outermost, but more efficient

Lazy vs. Eager

```
- fun test (x:{a:int,b:unit}) =
           if (#a{a=2,b=print("A\n")} = 2)
                        then (#a x)
                        else (#a x);

- test {a=7, b=print("B")}
```

What will be printed?

Call-by-need

- Efficient implementation of call-by-name (Algol 60)
- If purely functional language then only evaluate expression at most once, because can never change.

Programming w/Lazy Lists

```
fun from n = n :: from (n+1)
val nats = from 1;
fun nth 1 (fst::rest) = fst
      | nth n (fst::rest) = nth (n-1) rest
```

Approximating square roots

```
fun approxsqrts x = let
   fun from approx = approx ::
            from (0.5*(approx + x/approx))
in
   from 1.0
end;

fun within eps (approx1 :: approx2 :: rest) =
        if abs(approx1 - approx2) < eps
                then approx1
                else within eps (approx2::rest);

fun sqrtapprox x eps = within eps (approxsqrts x)
```

Why not be lazy?

- Eager language easier to implement w/ conventional techniques
- If side-effects then when will they occur?
- If computing in parallel, want to start as soon as feasible.
  - Even if result may be wasted!
- Can simulate in eager languages by adding extra dummy parameter to delay evaluation.

Program Correctness

- Referential transparency makes verification easier -- replace equals by equals.
- let val I = E in E' end is equivalent to [E/I]E'
- Not true if imperative features.
- Only true if lazy evaluation.
- Let E be a functional expression (no side effects). If E converges to a value v with eager evaluation then it converges to the same value with lazy eval.

Imperative Features

- Ref is a type constructor building references.
- val p = ref 47; !p is value referred to by p
- p := !p + 1;
- e1; e2; ...; en evaluated for side effects and return value of en.
- while e1 do e2;

Implementation Issues

- Slower than imperative
  - Lists instead of arrays
  - Passing around functions is expensive
  - Recursion can use more space
  - Lack of destructive updating
  - Listful style
  - Lazy has its own extra overhead

Summary of Functional Langs

- Use requires alternative approaches
- Referential transparency supports reasoning
- Higher-order functions powerful
- Some loss of efficiency balanced by programmer efficiency
- Implicit Polymorphism