CS 330 Lecture 14

- Outline
  - Call-by-value and Call-by-need (lazy evaluation)
  - Tail recursion
  - Efficiency
  - Infinite data structures

Two simple functions

- fun sqr(x) = x * x
- fun zero(x) = 0;
- When a function is called, the argument is substituted for the function's formal parameter in the body
- Evaluation rule: expressions have multiple function calls – when and how many times is the argument evaluated

Evaluation in ML: call-by-value

- Simple expressions: constants, variables, function calles and conditional expressions
- ML's evaluation rule:
  - To compute the value of f(E), first compute the value of expression E then substitute in f
- sqr(sqr(sqr(2))) -> sqr(sqr(2x2)) -> sqr(sqr(4)) -> sqr(4x4) -> sqr(16) -> 16 * 16 => 256
- what about zero(sqr(sqr(sqr(2)))) ?

Evaluation of recursive functions

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

```
fact(4) -> 4 * fact(4-1)
        -> 4 * fact(3)
        -> 4 * (3 * fact(3-1))
        -> 4 * (3 * (2 * fact(2-1)))
        -> 4 * (3 * (2 * (1 * fact(1-1))))
        -> 4 * (3 * (2 * (1 * fact(0)))))
        -> 4 * (3 * (2 * (1 * 1)))
        -> 4 * (3 * (2 * 1))
        -> 4 * (3 * 2)
        -> 4 * 6
        -> 24
```

This seems inefficient. The larger number the more numbers wait to be multiplied. Can we do better ?

Tail-recursive factorial

- Associative law 4 * (3 * (fact(2)) = 12 * fact(2)
- fun facti (n,p) = if n=0 then p else facti(n-1, n *p)
- tail recursive = the result of recursive call is returned immediately without modification

```
facti(4,1) -> facti(4-1, 4x1)
           -> facti(3, 4)
           -> facti(2, 12)
           -> facti(2-1, 2 *12)
           -> facti(1-1, 1 * 24)
           -> facti(0, 24)
           -> 24
```


Special role of conditional expressions

- Conditional expressions (if – then – else, andalso, orelse) are not functions
- fun cond(p,x,y) = if p then x else y;
- fun badf n = cond(n=0, 1, n*badf(n-1));
- Every call to badf runs forever why ?

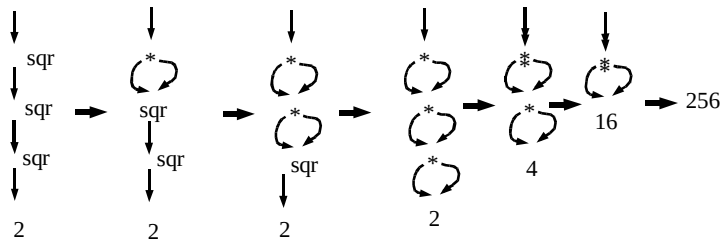
Call-by-name

- What if we give functions their arguments as expressions, not values ?
- Call-by-name rule: to compute the value of f(E), substitute E immediately into the body of f. Then compute the value of the resulting expression
- zero(sqr(sqr(sqr(2)))) is immediately 0
- However observe what happens in: sqr(sqr(sqr(2))) = sqr(sqr(2)) * sqr(sqr(2))

Call-by-need – Lazy evaluation

- Like call-by-name but ensure that each argument is evaluated at most once
- Pointer structure – Directed graph of functions and arguments
- Graph reduction

Graph reduction of sqr(sqr(sqr 2)))



Comparison of strict and lazy evaluation

- Call-by-need does the least possible evaluation but requires much book-keeeping
- Sometimes lazy evaluation saves a lot of spaces; sometimes it wastes space

Space leak with lazy evaluation for facti

```
facti(4,i) -> facti(4-1, 4 *1)
           -> facti(3-1, 3 * (4 * 1))
           -> facti(2-1, 2 * (3 * ( 4 * 1)))
           -> facti(1-1, 1 * (2 * ( 3 * ( 4 * 1))))
           -> 1 * (2 * (3 * (4 * 1)))
....
           -> 24
```


Haskell

- www.haskell.org
- Pure functional language with lazy evaluation
- Lazy lists = elements are not evaluated until their values are needed by the rest of the program
- All data structures in Haskell are Lazy
- As an exercise let's do infite lists in SML