Chapter 10 :: Functional Languages

Evaluation Order

Michael L. Scott
Evaluation Order Revisited

• Applicative order: evaluates all arguments before invoking function
  – what you're used to in imperative languages
  – usually faster

• Normal order: doesn’t evaluate arg until you need it
  – sometimes faster
  – terminates if anything will (Church-Rosser theorem)
Evaluation Order (Example)

(\(\text{and (not (= y 0)) (/ x y))}\))
Why normal order may be slow

(define double (lambda (x) (+ x x )))
(double (* 3 4))

Applicative Order
(double (* 3 4))
⇒ (double 12)
⇒ (+ 12 12)
⇒ 24

Normal Order
(double (* 3 4))
⇒ (+ (* 3 4) (* 3 4))
⇒ (+ 12 (* 3 4))
⇒ (+ 12 12)
⇒ 24
Scheme Evaluation Order

• In Scheme
  – functions use applicative order defined with lambda
    • arguments are evaluated right to left
  – special forms (aka macros) use normal order defined with syntax-rules
(define add (lambda (x) (+ x 20)))
(define min (lambda (x y) (if (< x y) x y)))
(trace add)
(min (add 5) (add 20))

[40
  <=- #[compound-procedure 4 add] Args: 20] ; <=- means exiting this fct
[25
;Value: 25
Strict versus Non-strict Languages

• A *strict* language requires all arguments to be well-defined, so applicative order can be used.

• A *non-strict* language does not require all arguments to be well-defined; it requires normal-order evaluation.

• Scheme is strict for functions, but non-strict for special forms.

• C is strict, except for boolean expressions.
Forcing Normal Order in Scheme

• Use _delay_ and _force_ constructs
  – _delay_: creates an expression but does not evaluate it
  – _force_: forces the evaluation of a delayed expression

• Example
  (define expr (delay (+ a 10)))
  (define a 15)
  (force expr) ➞ 25
(define naturals
  (letrec ((next (lambda (n)
      (cons n (delay (next (+ n 1)))))))
    (next 1)))

(define head car)
(define tail (lambda (stream) (force (cdr stream))))

(head naturals) ➞ 1
(head (tail naturals) ➞ 2
(head (tail (tail naturals))) ➞ 3
Memoization

• Memoization: Technique saves an expression’s result in some type of fast lookup structure
  – Thereafter references to the expression use this computed value
  – Brings performance of normal order evaluation within a constant factor of applicative order evaluation

• Spreadsheets use memoization
  Example:
  \[ a_{10} = b_{10} + c_{10} \]
  \[ b_{10} = 3 \times b_{9} \]
  \[ c_{10} = 8 \times c_{9} \]
  \[ b_{9} = 5 \]
  \[ c_{9} = 10 \]
Memoization (Potential Problem)

- May not work properly in the presence of side-effects
- Example:
  
  (define x 5)
  (define y 10)
  (define (z (* x y))
  (set! x 2)
  (define (a (* x y))