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)

```
(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

Scheme Applicative Order Example

```
(define add (lambda (x) (+ x 20)))
(define min (lambda (x y) (if (< x y) x y)))
(trace add)
(min (add 5) (add 20))
[Entering #[compound-procedure 4 add] Args: 20]
[40
        <== #[compound-procedure 4 add]  Args: 20]  ; <==
        means exiting this fct
[Entering #[compound-procedure 4 add] Args: 5]
 [25
         <== #[compound-procedure 4 add] Args: 5]
;Value: 25
```

Strict versus Non-strict Languages

- A *strict* language requires all arguments to be well-defined, so applicative order can be used
- A n*on-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
```

Forcing Normal Order in Scheme

```
(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:

a10 = b10 + c10

b10 = 3 * b9

c10 = 8 * c9

b9 = 5

c9 = 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))
```