TEL AVIV UNIVERSITY
Department of Computer Science
0368.1105 – Extended Introduction to Computer Science
Fall Semester, 2000/2001

Handout for Lecture 3

Applicative order evaluation

- To evaluate

```
(<operator> <operand1> <operand2> ... <operandn>)
```

- Evaluate in any order
  <operator> to get the procedure,
  and <operand1> <operand2> ... <operandn> to get the arguments.
- If <operator> is primitive, do whatever it does.
- If <operator> is compound, evaluate the body with the formal parameters replaced by the arguments.

Normal order evaluation

- To evaluate

```
(<operator> <operand1> <operand2> ... <operandn>)
```

- Evaluate <operator> to get the procedure.
- If <operator> is primitive, do whatever it does.
- If <operator> is compound, evaluate the body with the formal parameters replaced by the <operands>.

```
Applicative order

((lambda (x) (+ x x)) (* 3 4))
((lambda (x) (+ x x)) 12)
(+ 12 12)
24
```

```
Normal order

((lambda (x) (+ x x)) (* 3 4))
(+ (* 3 4) (* 3 4))
(+ 12 12)
24
```

In some cases, applicative order evaluation and normal order evaluation may yield different results. For example,

```
((lambda (x y) (+ x 2)) 3 (/ 1 0))
```

Note: Scheme is an Applicative order language

Iteration vs. Recursion

```
(define (factorial n)
  (if (= n 1)
      1
      (* n (factorial (- n 1)))))

;; Recursive exponentiation

(define exp-1
    (lambda (a b)
        (if (= b 0)
            1
            (* a (exp-1 a (- b 1))))))

;; Iterative exponentiation

(define exp-2
    (lambda (a b)
        (define exp-iter
            (lambda (a b ans)
                (if (= b 0)
                    ans
                    (exp-iter a (- b 1) (* a ans)))))
        (exp-iter a b 1)))

(exp-1 3 4)
(* 3 (exp-1 3 3))
(* 3 (* 3 (exp-1 3 2)))
(* 3 (* 3 (* 3 (exp-1 3 1))))
(* 3 (* 3 (* 3 (* 3 (exp-1 3 0)))))
(* 3 (* 3 (* 3 (* 3 1))))
(* 3 (* 3 (* 3 3)))
(* 3 (* 3 9))
(* 3 27)
81

(exp-2 3 4)
(exp-iter 3 4 1)
(exp-iter 3 3 3)
(exp-iter 3 2 9)
(exp-iter 3 1 27)
(exp-iter 3 0 81)
81
```