CS61A
The Structure and Interpretation Of Computer Programs

Lecture 1.1.2:

Functional Programming

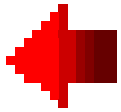
2005-06-21

Kurt Meinz

www-inst.eecs.berkeley.edu/~cs61a/

Lecture outline

- **Functions**
- **Recursion**



Functions

- **Formally:**

  For all functions f and for all x, y in the domain of f:

  f(x), f(y) exist and

  x=y ➔ f(x)=f(y)

- **In English:**
  - **Every expression has a value**
  - **Same arguments ➔ Same value**
    - **Functions are timeless!**

Functions

- **Mathematical Terms (loosely):**
  - **Domain: The set of all possible inputs (arguments) for f**
  - **Range: The set of all possible outputs (return values) for f**

Functions

- **Procedural Terms:**
  - o "Formal Parameter"
  - o "Formal Definition"
  - o "Actual Argument Expression"
  - o "Actual Argument Value“

```
(define (square x) (* x x))
(square (+ 2 3))
```

Functions

- **Procedural Terms:**
  - o "Formal Parameter” ➔ “x”
  - o "Formal Definition” ➔ “(* x x)”
  - o "Actual Argument Expression” ➔ “(+ 2 3)”
  - o "Actual Argument Value” ➔ “5”

```
(define (square x) (* x x))
(square (+ 2 3))
```

Functions: Rules of evaluation

Applicative Order: ("Inside-out")

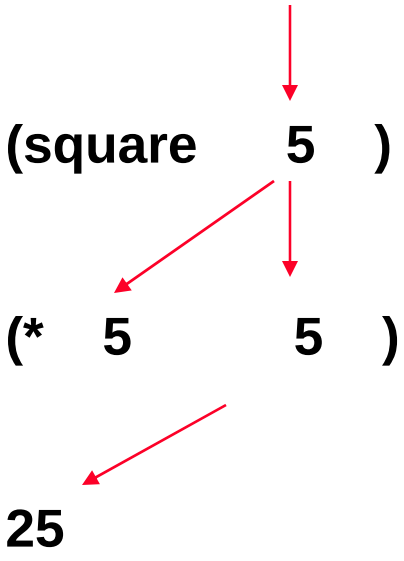
1. Evaluate AA Expression(s)
2. Substitute AA Value for Formal Parameters ("Binding")
3. Evaluate function definition w/ Bindings
4. Return value to caller

Functions: Rules of evaluation

Applicative Order: ("Inside-out")

(define (square x) (* x x))

(square (+ 2 3))



1. Evaluate AAE for “square”:
   - (+ 2 3) ➔ 5
2. Bind x=5,
3. Eval square body
   - (* x x) ➔ 25
4. Return 25

Functions: Rules of evaluation

Normal Order: (“Lazy")

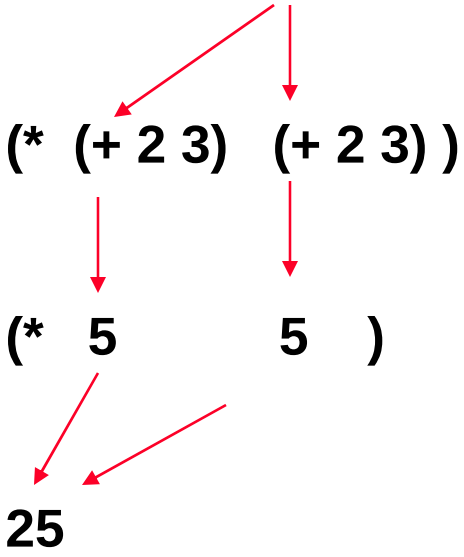
1. Bind Formal Parameter to AA Expression(s)
2. Evaluate function definition w/ Bindings
    1. Evaluate AAEs as necessary
3. Return value to caller

Functions: Rules of evaluation

Normal Order: ("Lazy")

(define (square x) (* x x))

(square (+ 2 3))



1. Bind x=(+ 2 3)
2. Evaluate body of square with binding
3. "Forced" to evaluate AAEs to get a value to print.

Functions: Rules of evaluation

Notes:

o Scheme uses applicative order.

- We'll build a normal order interpreter later.

o Purely functional language ➔ no difference between applicative and normal evaluation.

- Called "referential transparency" (i.e. can replace an AAE with an AAV and vice-versa)

Functions: Rules of evaluation

Notes: Non-functional Languages

(define (zero x) (- x x))

(random 100)

Applicative:
(zero (random 100))
(zero 17)
(- 17 17)

Normal:
(zero (random 100))
(- (random 100) (random 100))
(- 17 32)

Functions: Rules of evaluation

Notes: Sometimes normal can be faster

How??

Functions: Rules of evaluation

Notes: Sometimes normal can be faster

```
(define (first-one x y)
        x)
(first-one 10 (factorial 99999999999999))
```

Why? Because the factorial is never needed, so it is never used! (“Lazy”)

Lecture outline

- **Functions**
- **Recursion**