Scheming Recursively

Lecturer: Professor Albert R. Meyer

September 9, 1997

Scheme programming focuses on *Values*:

1. Numbers,
   written as 0, 1, -2.7, 987e11, ...

2. Truth Values (Booleans),
   written as `#t`, `#f`.

... that's all for today.

The value of the *Expression*

```
(+ (* 2 3 4) (- 9 (+ 7 5)) 7.1)
```

is the number 28.1.

$$\underbrace{\texttt{(+ (* 2 3 4) (- 9 (+ 7 5)) 7.1)}}_{\text{Expression}} \Longrightarrow \underbrace{28.1}_{\text{Value}}$$

The value of the expression `(< 2 3.7 5)` is true.

$$\texttt{(< 2 3.7 5)} \Longrightarrow \texttt{\#t}$$

Expression

A *combination* looks like:

$$(\langle\text{operator}\rangle\ \ \langle\text{argument}_1\rangle\ \ \langle\text{argument}_2\rangle \ldots)$$

where $\langle$operator$\rangle$, $\langle$argument$_1\rangle$, . . . are themselves expressions.

Value Mantra for Combinations:

"The value of a combination is the result of APPLYing the *value* of the $\langle$operator$\rangle$ to the *values* of the $\langle$argument$\rangle$'s."

Value Mantra for Primitive Applications

“1. The value of operator + applied to number values is the sum of the numbers.

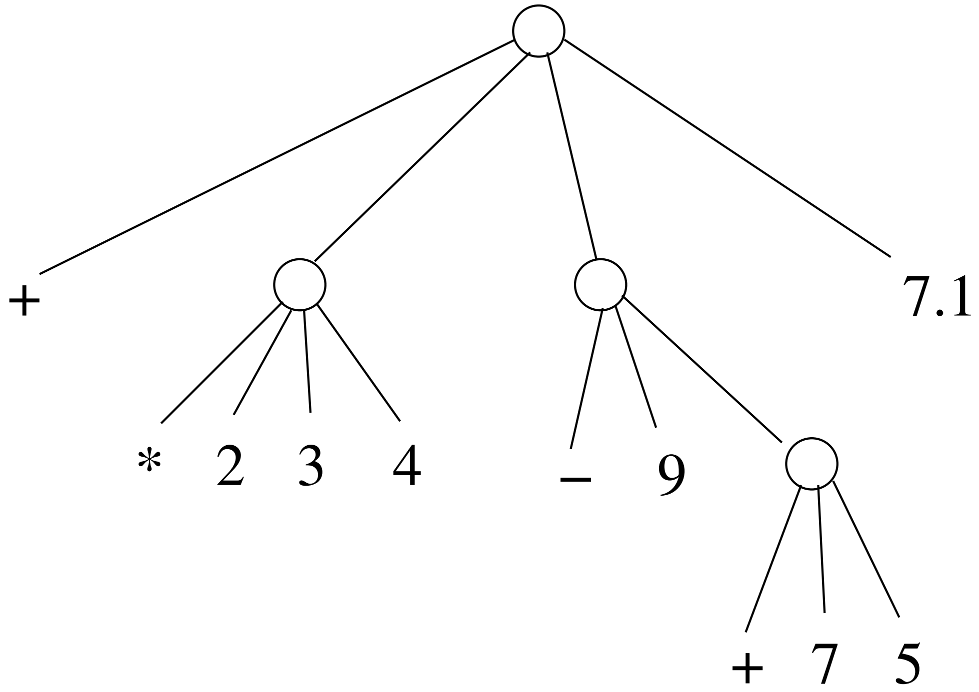
2. The value of op `*` applied to numbers is the product of the numbers.

3. The value of op < applied to numbers is `#t` if the numbers are in strictly increasing order; it is `#f` otherwise.

4. The value of op − applied to two numbers is their difference. Op − applied to other than two numbers is somewhat arbitrary (look it up if you're worried about it).

and so on . . . ”

Expression as a tree




```
(+ (* 2 3 4) (- 9 (+ 7 5)) 7.1)
```

IF expressions

$$\texttt{(if \#t 1 2)} \Longrightarrow 1$$

$$\texttt{(if \#f 1 2)} \Longrightarrow 2$$

$$\texttt{(if (< 3 2) 0 (> 9 8 7))} \Longrightarrow \texttt{\#t}$$

(if ⟨predicate⟩ ⟨consequent⟩ ⟨alternative⟩)

Value Mantra for IF

"When the value of ⟨predicate⟩ is `#f`, the value of the if-expression is the value of the ⟨alternative⟩; otherwise, it is the value of the ⟨consequent⟩."

What is the value of

`(+ x 1)`?

What is the value of `x`?

— Depends how `x` is defined:

```
(define x 3)

(+ x 1) ⟹ 4


(define x -1)

(+ x 1) ⟹ 0
```

A VARIABLE is an expression:

`x    xy    xplusy    x.1.y`

A set of definitions of the values of variables is called an *environment*.

Value Mantra for Variables

"The value of a variable in an environment is whatever the environment defines it to be."

"At 'top-level' of REPL:

$$\texttt{(define } \langle\text{variable}\rangle\ \langle\text{expression}\rangle\texttt{)}$$

causes the value of ⟨variable⟩ to be the value of ⟨expression⟩ in the top-level, 'global' environment."

This is the **Global Define Mantra**.

Technicality:
We don't ask or care about the value of `(define x 3)`.

Compound procedures

```
(define (add1 x)
    (+ x 1))
```

adds a compound procedure named `add1` to the global environment.

Now

`(add1 2)` $\Longrightarrow 3$

`(add1 7)` $\Longrightarrow 8$

Value Mantra for Compound Procedure Application

"The value of a compound procedure named `proc` applied to a value, n, in an environment where `proc` is defined by

$$\texttt{(define (proc x)}\ \langle\text{expression}\rangle\texttt{)}$$

is the value of ⟨expression⟩ in the environment with `x` given the value n."

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

Value of `(square 3)` is the value of

```
(* x x)
```

in an environment with `x` given the value 3.
(Compound Procedure Apply Mantra)

So Value of `x` is 3. (Variable mantra)

So Value of `(* x x)` is result of

Applying the value of `*` to 3 and 3 (Combination Mantra),

which is the product of 3 and 3, namely 9
(Primitive Application Mantra).

Sum of Squares

Let $s(n) = 0^2 + 1^2 + 2^2 + \cdots + n^2$

So $\quad s(0) = 0, \quad s(1) = 1, \quad s(2) = 5,$

$\quad s(3) = 14, \quad s(4) = 30, \quad \ldots$

Useful fact:

$$s(n) = \underbrace{0^2 + 1^2 + 2^2 + \cdots + (n-1)^2}_{s(n-1)} + n^2$$

1. We know $s(0) = 0$.

2. Useful fact tells us how to find the value of $s(n)$ *assuming* we know $s(n-1)$. *Wishful thinking* is ok here!

In Scheme

```
(define (sum-sq n)
  (if (zero? n)
      0
      (+ (sum-sq (dec n))
         (square n))))
```

Example:

$$\texttt{(sum-sq 3)} \Longrightarrow s(3),\ \text{namely } 14.$$

Using the Mantras Again

Val of `(sum-sq 3)` is

Val of

```
(if (zero? n)
    0
    (+ (sum-sq (dec n))
       (square n)))
```

when Val of *n* is 3.

Val of `(zero? n)` when `n` is 3, is `#f`.

So Val of `(if ... )` above is Val of

```
(+ (sum-sq (dec n))
   (square n))
```

Val of `(square n)` when `n` is 3 is 9.

Val of `(dec n)` is 2.

So Val of `(sum-sq 3)` is the sum of 9 and the Val of `(sum-sq 2)`.

$$\vdots$$

$$9 + 4 + 1 + 0 = 14$$

How to tell if an integer n is odd?

Wishful thinking, assume we could tell if $n - 1$ is odd.
Then n is odd iff $n - 1$ is not odd:

```
(define (odd? n)
  (if (zero? n)
      #f
      (not (odd? (dec n)))))
```

Val of `(odd? 3)` is

[because `(zero? 3)` $\Longrightarrow$ `#f`]

Val of `(not (odd? 2))`, which is

Val of `(not (not (odd? 1)))`, which is

Val of `(not (not (not (odd? 0))))`, which is

Val of `(not (not (not #f)))`, which is

`#t`

Making More than One Wish

```
(define (odd? n)
  (if (zero? n)
      #f
      (even? (dec n))))

(define (even? n)
  (if (zero? n)
      #t
      (odd? (dec n))))
```

Val of `(odd? 3)` is

Val of `(even? 2)`, which is

Val of `(odd? 1)`, which is

Val of `(even? 0)`, which is

`#t`

Imperative Style

$$\sum_{i=0}^{n} i^2 = 0^2 + 1^2 + \cdots + n^2$$

"Starting with $i = 0$, repeatedly add i^2 to the running sum and increment i until i exceeds n."

Corresponds to simple “register machine” or “assembly language” program:

```
start:      i := 0;
            s := 0;
loop:       if  i = n + 1  goto end;
            temp := square i;
            i := i + 1;
            s := s + temp;
            goto loop;
end:
```

at the end the sum of squares is in `s`.

Recursive Style is Just as Powerful

```
(define (sum-sq-imp n)
  (loop 0 0 n))

(define (loop i s n)
  (if (= i (inc n))
      (end s)
      (next i s n (square i))))

(define (next i s n temp)
  (loop (inc i)
        (+ s temp)
        n))
```

Another example: `odd?/even?`

```
start:      if (n = 0) goto end-false;
            n := n - 1;
            if (n = 0) goto end-true;
            n := n -1;
            goto start;
```

Register machines correspond to a
special case
of recursive procedures called
TAIL-RECURSIVE
procecures