

Lecture 12

Let – Implementation



T. METIN SEZGIN

Specifying the behavior



- Programs

```
(value-of-program exp)  
= (value-of exp [i=[1],v=[5],x=[10]])
```

- Expressions

- Constructors

```
const-exp  : Int → Exp  
zero?-exp  : Exp → Exp  
if-exp     : Exp × Exp × Exp → Exp  
diff-exp   : Exp × Exp → Exp  
var-exp    : Var → Exp  
let-exp    : Var × Exp × Exp → Exp
```

```
(value-of (const-exp n) ρ) = (num-val n)  
(value-of (var-exp var) ρ) = (apply-env ρ var)
```

```
(value-of (diff-exp exp1 exp2) ρ)  
= (num-val  
  (-  
    (expval->num (value-of exp1 ρ))  
    (expval->num (value-of exp2 ρ))))
```

- Observer

```
value-of : Exp × Env → ExpVal
```

Specifying the behavior



- Programs

$(\text{value-of-program } \text{exp})$
 $= (\text{value-of } \text{exp} \text{ } [i=[1], v=[5], x=[10]])$

- Expressions

 - Constructors

$\text{const-exp} : \text{Int} \rightarrow \text{Exp}$
 $\text{zero?-exp} : \text{Exp} \rightarrow \text{Exp}$
 $\text{if-exp} : \text{Exp} \times \text{Exp} \times \text{Exp} \rightarrow \text{Exp}$
 $\text{diff-exp} : \text{Exp} \times \text{Exp} \rightarrow \text{Exp}$
 $\text{var-exp} : \text{Var} \rightarrow \text{Exp}$
 $\text{let-exp} : \text{Var} \times \text{Exp} \times \text{Exp} \rightarrow \text{Exp}$

$$\frac{(\text{value-of } \text{exp}_1 \ \rho) = \text{val}_1}{(\text{value-of } (\text{zero?-exp } \text{exp}_1) \ \rho)}$$
$$= \begin{cases} (\text{bool-val } \#t) & \text{if } (\text{expval} \rightarrow \text{num } \text{val}_1) = 0 \\ (\text{bool-val } \#f) & \text{if } (\text{expval} \rightarrow \text{num } \text{val}_1) \neq 0 \end{cases}$$
$$\frac{(\text{value-of } \text{exp}_1 \ \rho) = \text{val}_1}{(\text{value-of } (\text{if-exp } \text{exp}_1 \ \text{exp}_2 \ \text{exp}_3) \ \rho)}$$
$$= \begin{cases} (\text{value-of } \text{exp}_2 \ \rho) & \text{if } (\text{expval} \rightarrow \text{bool } \text{val}_1) = \#t \\ (\text{value-of } \text{exp}_3 \ \rho) & \text{if } (\text{expval} \rightarrow \text{bool } \text{val}_1) = \#f \end{cases}$$

 - Observer

$\text{value-of} : \text{Exp} \times \text{Env} \rightarrow \text{ExpVal}$

Specifying the behavior



- Programs

$(\text{value-of-program } \text{exp})$
 $= (\text{value-of } \text{exp} \text{ } [i=[1], v=[5], x=[10]])$

- Expressions

- Constructors

$\text{const-exp} : \text{Int} \rightarrow \text{Exp}$
 $\text{zero?-exp} : \text{Exp} \rightarrow \text{Exp}$
 $\text{if-exp} : \text{Exp} \times \text{Exp} \times \text{Exp} \rightarrow \text{Exp}$
 $\text{diff-exp} : \text{Exp} \times \text{Exp} \rightarrow \text{Exp}$
 $\text{var-exp} : \text{Var} \rightarrow \text{Exp}$
 $\text{let-exp} : \text{Var} \times \text{Exp} \times \text{Exp} \rightarrow \text{Exp}$

$$\frac{(\text{value-of } \text{exp}_1 \text{ } \rho) = \text{val}_1}{(\text{value-of } (\text{let-exp } \text{var } \text{exp}_1 \text{ body}) \text{ } \rho) = (\text{value-of } \text{body} \text{ } [\text{var} = \text{val}_1] \rho)}$$
$$(\text{value-of } (\text{let-exp } \text{var } \text{exp}_1 \text{ body}) \text{ } \rho) = (\text{value-of } \text{body} \text{ } [\text{var} = (\text{value-of } \text{exp}_1 \text{ } \rho)] \rho)$$

- Observer

$\text{value-of} : \text{Exp} \times \text{Env} \rightarrow \text{ExpVal}$

Behavior implementation



what we envision

Let $\rho = [i=1, v=5, x=10]$.

```
(value-of
  <<- (- (x, 3), - (v, i)) >>
  ρ)
```

```
= [(-
  [(value-of <<- (x, 3)>> ρ)]
  [(value-of <<- (v, i)>> ρ)])]
```

```
= [(-
  (-
    [(value-of <<x>> ρ)]
    [(value-of <<3>> ρ)])
    [(value-of <<- (v, i)>> ρ)])]
```

```
= [(-
  (-
    10
    [(value-of <<3>> ρ)])
    (value-of <<- (v, i)>> ρ))]
```

```
= [(-
  (-
    10
    3)
    [(value-of <<- (v, i)>> ρ)])]
```

```
= [(-
  7
  [(value-of <<- (v, i)>> ρ)])]
```

```
= [(-
  7
  (-
    [(value-of <<v>> ρ)]
    [(value-of <<i>> ρ)])])]
```

```
= [(-
  7
  (-
    5
    [(value-of <<i>> ρ)])])]
```

```
= [(-
  7
  (-
    5
    1)])]
```

```
= [(-
  7
  4)]
```

```
= [3]
```

Behavior implementation



what we envision

Let $\rho = [x=[33], y=[22]]$.

```
(value-of
  <<if zero?(- (x,11)) then - (y,2) else - (y,4)>>
   $\rho$ )

= (if (expval->bool (value-of <<zero?(- (x,11))>>  $\rho$ ))
      (value-of <<- (y,2)>>  $\rho$ )
      (value-of <<- (y,4)>>  $\rho$ ))

= (if (expval->bool (bool-val #f))
      (value-of <<- (y,2)>>  $\rho$ )
      (value-of <<- (y,4)>>  $\rho$ ))

= (if #f
      (value-of <<- (y,2)>>  $\rho$ )
      (value-of <<- (y,4)>>  $\rho$ ))

= (value-of <<- (y,4)>>  $\rho$ )

= [18]
```

Nugget



Intro to implementation

It all revolves around **value-of**

The Interpreter



```
run : String → ExpVal
(define run
  (lambda (string)
    (value-of-program (scan&parse string))))

value-of-program : Program → ExpVal
(define value-of-program
  (lambda (pgm)
    (cases program pgm
      (a-program (exp1)
        (value-of exp1 (init-env)))))))
```


The Interpreter



value-of : $Exp \times Env \rightarrow ExpVal$

(define value-of

(lambda (exp env)

(cases expression exp

(value-of (const-exp n) ρ) = n

(const-exp (num) (num-val num))

(value-of (var-exp var) ρ) = (apply-env ρ var)

(var-exp (var) (apply-env env var))

(value-of (diff-exp exp_1 exp_2) ρ) =
 $\left[(- \left[(value-of \exp_1 \rho) \right] \left[(value-of \exp_2 \rho) \right]) \right]$

(diff-exp (exp1 exp2)

(let ((val1 (value-of exp1 env))

(val2 (value-of exp2 env)))

(let ((num1 (expval->num val1))

(num2 (expval->num val2)))

(num-val

(- num1 num2))))))

(value-of exp_1 ρ) = val_1

(value-of (zero?-exp exp_1) ρ)

= $\begin{cases} (\text{bool-val } \#t) & \text{if } (\text{expval} \rightarrow \text{num } val_1) = 0 \\ (\text{bool-val } \#f) & \text{if } (\text{expval} \rightarrow \text{num } val_1) \neq 0 \end{cases}$

(zero?-exp (exp1)

(let ((val1 (value-of exp1 env)))

(let ((num1 (expval->num val1)))

(if (zero? num1)

(bool-val #t)

(bool-val #f))))))

(value-of exp_1 ρ) = val_1

(value-of (if-exp exp_1 exp_2 exp_3) ρ)

= $\begin{cases} (value-of \exp_2 \rho) & \text{if } (\text{expval} \rightarrow \text{bool } val_1) = \#t \\ (value-of \exp_3 \rho) & \text{if } (\text{expval} \rightarrow \text{bool } val_1) = \#f \end{cases}$

(if-exp (exp1 exp2 exp3)

(let ((val1 (value-of exp1 env)))

(if (expval->bool val1)

(value-of exp2 env)

(value-of exp3 env))))

(value-of exp_1 ρ) = val_1

(value-of (let-exp var exp_1 $body$) ρ)

= (value-of $body$ [$var = val_1$] ρ)

(let-exp (var exp1 body)

(let ((val1 (value-of exp1 env)))

(value-of body

(extend-env var val1 env))))))