Lecture 12
Let

T. METIN SEZGIN

1

LET: our pet language

Program ::= *Expression*
`a-program (exp1)`

Expression ::= *Number*
`const-exp (num)`

Expression ::= `-(`*Expression* `,` *Expression*`)`
`diff-exp (exp1 exp2)`

Expression ::= `zero? (`*Expression*`)`
`zero?-exp (exp1)`

Expression ::= `if` *Expression* `then` *Expression* `else` *Expression*
`if-exp (exp1 exp2 exp3)`

Expression ::= *Identifier*
`var-exp (var)`

Expression ::= `let` *Identifier* `=` *Expression* `in` *Expression*
`let-exp (var exp1 body)`

2

An example program

- Input

```
"-(55, -(x,11))"
```

- Scanning & parsing

```
(scan&parse "-(55, -(x,11))")
```

- The AST

```
#(struct:a-program
   #(struct:diff-exp
      #(struct:const-exp 55)
      #(struct:diff-exp
         #(struct:var-exp x)
         #(struct:const-exp 11))))
```

Program ::= *Expression*
`a-program (exp1)`

Expression ::= *Number*
`const-exp (num)`

Expression ::= `-(` *Expression* `,` *Expression* `)`
`diff-exp (exp1 exp2)`

Expression ::= `zero? (` *Expression* `)`
`zero?-exp (exp1)`

Expression ::= `if` *Expression* `then` *Expression* `else` *Expression*
`if-exp (exp1 exp2 exp3)`

Expression ::= *Identifier*
`var-exp (var)`

Expression ::= `let` *Identifier* `=` *Expression* `in` *Expression*
`let-exp (var exp1 body)`

3

Nugget

Steps of inventing a language

4

Components of the language

- Syntax and datatypes
- Values
- Environment
- Behavior specification
- Behavior implementation
  - Scanning
  - Parsing
  - Evaluation

5

Syntax data types

Program ::= *Expression*
`a-program (exp1)`

Expression ::= *Number*
`const-exp (num)`

Expression ::= `-(`*Expression* `,` *Expression*`)`
`diff-exp (exp1 exp2)`

Expression ::= `zero? (`*Expression*`)`
`zero?-exp (exp1)`

Expression ::= `if` *Expression* `then` *Expression* `else` *Expression*
`if-exp (exp1 exp2 exp3)`

Expression ::= *Identifier*
`var-exp (var)`

Expression ::= `let` *Identifier* `=` *Expression* `in` *Expression*
`let-exp (var exp1 body)`

```
(define-datatype program program?
  (a-program
    (exp1 expression?)))

(define-datatype expression expression?
  (const-exp
    (num number?))
  (diff-exp
    (exp1 expression?)
    (exp2 expression?))
  (zero?-exp
    (exp1 expression?))
  (if-exp
    (exp1 expression?)
    (exp2 expression?)
    (exp3 expression?))
  (var-exp
    (var identifier?))
  (let-exp
    (var identifier?)
    (exp1 expression?)
    (body expression?)))
```

6

Nugget

Values

7

Values

- Set of values manipulated by the program
  - Expressed values
    - Possible values of expressions
  - Denoted values
    - Possible values of variables

$$ExpVal = Int + Bool$$
$$DenVal = Int + Bool$$

- Interface for values
  - Constructors
  - Observers

num-val	: $Int \to ExpVal$
bool-val	: $Bool \to ExpVal$
expval->num	: $ExpVal \to Int$
expval->bool	: $ExpVal \to Bool$

8

Environments

- Same model of environment from before

- ρ ranges over environments.
- [] denotes the empty environment.
- $[var = val]\rho$ denotes `(extend-env var val ρ)`.
- $[var_1 = val_1, var_2 = val_2]\rho$ abbreviates $[var_1 = val_1]([var_2 = val_2]\rho)$, etc.
- $[var_1 = val_1, var_2 = val_2, \ldots]$ denotes the environment in which the value of var_1 is val_1, etc.

- Use `[x=3] [y=7] [u=5]ρ` to abbreviate

```
(extend-env 'x 3
  (extend-env 'y 7
    (extend-env 'u 5 ρ)))
```

9

Nugget

We specify the meaning of expressions first

10

Specifying the behavior

- Programs

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

- Expressions
  - Constructors

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

```
(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	$: Exp \times Env \rightarrow ExpVal$

11

Specifying the behavior

- Programs

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

- Expressions
  - Constructors

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

$$\frac{\texttt{(value-of } exp_1\ \rho) = val_1}{\texttt{(value-of (zero?-exp } exp_1)\ \rho) = \begin{cases}\texttt{(bool-val \#t)} & \text{if (expval->num } val_1) = 0\\ \texttt{(bool-val \#f)} & \text{if (expval->num } val_1) \neq 0\end{cases}}$$

$$\frac{\texttt{(value-of } exp_1\ \rho) = val_1}{\texttt{(value-of (if-exp } exp_1\ exp_2\ exp_3)\ \rho) = \begin{cases}\texttt{(value-of } exp_2\ \rho) & \text{if (expval->bool } val_1) = \texttt{\#t}\\ \texttt{(value-of } exp_3\ \rho) & \text{if (expval->bool } val_1) = \texttt{\#f}\end{cases}}$$

  - Observer

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

12

Specifying the behavior

- Programs

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

- Expressions
  - Constructors

const-exp	$: Int \to Exp$
zero?-exp	$: Exp \to Exp$
if-exp	$: Exp \times Exp \times Exp \to Exp$
diff-exp	$: Exp \times Exp \to Exp$
var-exp	$: Var \to Exp$
let-exp	$: Var \times Exp \times Exp \to Exp$

$$\frac{\texttt{(value-of } exp_1 \ \rho) = val_1}{\texttt{(value-of (let-exp } var \ exp_1 \ body) \ \rho) = \texttt{(value-of } body \ [var = val_1]\rho)}$$

```
(value-of (let-exp var exp1 body) ρ)
= (value-of body [var=(value-of exp1 ρ)]ρ)
```

  - Observer

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

13

Behavior implementation

what we envision

```
Let ρ = [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⌉
```

14

Lecture 13
Let – Implementation

T. METIN SEZGIN

15

Behavior implementation

what we envision

Let ρ = [x=⌈33⌉,y=⌈22⌉].

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

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

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

= (if #f
    (value-of <<-(y,2)>> ρ)
    (value-of <<-(y,4)>> ρ))

= (value-of <<-(y,4)>> ρ)

= ⌈18⌉
```

16

Nugget

Intro to implementation
It all revolves around **`value-of`**

17

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))))))
```

18

The Interpreter

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

```
(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) = $\lceil$(- $\lfloor$(value-of exp_1 ρ)$\rfloor$ $\lfloor$(value-of exp_2 ρ)$\rfloor$)$\rceil$

```
      (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)))))
```

$$\frac{\text{(value-of } exp_1 \ \rho) = val_1}{\text{(value-of (zero?-exp } exp_1) \ \rho) = \begin{cases} \text{(bool-val \#t)} & \text{if (expval->num } val_1) = 0 \\ \text{(bool-val \#f)} & \text{if (expval->num } val_1) \neq 0 \end{cases}}$$

```
      (zero?-exp (exp1)
        (let ((val1 (value-of exp1 env)))
          (let ((num1 (expval->num val1)))
            (if (zero? num1)
              (bool-val #t)
              (bool-val #f)))))
```

$$\frac{\text{(value-of } exp_1 \ \rho) = val_1}{\text{(value-of (if-exp } exp_1 \ exp_2 \ exp_3) \ \rho) = \begin{cases} \text{(value-of } exp_2 \ \rho) & \text{if (expval->bool } val_1) = \text{\#t} \\ \text{(value-of } exp_3 \ \rho) & \text{if (expval->bool } val_1) = \text{\#f} \end{cases}}$$

```
      (if-exp (exp1 exp2 exp3)
        (let ((val1 (value-of exp1 env)))
          (if (expval->bool val1)
            (value-of exp2 env)
            (value-of exp3 env))))
```

$$\frac{\text{(value-of } exp_1 \ \rho) = val_1}{\text{(value-of (let-exp } var \ exp_1 \ body) \ \rho) = \text{(value-of } body \ [var = val_1]\rho)}$$

```
      (let-exp (var exp1 body)
        (let ((val1 (value-of exp1 env)))
          (value-of body
            (extend-env var val1 env))))))))
```