Lecture 19
Translation

T. METIN SEZGIN

1

Big idea

```
let x = 200
in let f = proc (z) -(z,x)
   in let x = 100
      in let g = proc (z) -(z,x)
         in -((f 1), (g 1))
```

2

Nuggets

- Use lexical addresses instead of variable names
- Implement a new model of environment
  - Use addresses
  - Much like a memory model

Implementing lexical addressing

The Idea: rewrite **`value-of`** (i.o.w. write a translator)

```
let x = 37
in proc (y)
     let z = -(y,x)
     in -(x,y)
```

```
#(struct:a-program
   #(struct:nameless-let-exp
      #(struct:const-exp 37)
      #(struct:nameless-proc-exp
         #(struct:nameless-let-exp
            #(struct:diff-exp
               #(struct:nameless-var-exp 0)
               #(struct:nameless-var-exp 1))
            #(struct:diff-exp
               #(struct:nameless-var-exp 2)
               #(struct:nameless-var-exp 1))))))
```

The translator: the target language

Expression ::= `%lexref` *number*
$\boxed{\texttt{nameless-var-exp (num)}}$

Expression ::= `%let` *Expression* `in` *Expression*
$\boxed{\texttt{nameless-let-exp (exp1 body)}}$

Expression ::= `%lexproc` *Expression*
$\boxed{\texttt{nameless-proc-exp (body)}}$

The translator: Exp x Senv → NamelessExp

Static Environment

Senv = *Listof*(*Sym*)
Lexaddr = *N*

empty-senv : () → *Senv*
```
(define empty-senv
  (lambda ()
    '()))
```

extend-senv : *Var* × *Senv* → *Senv*
```
(define extend-senv
  (lambda (var senv)
    (cons var senv)))
```

apply-senv : *Senv* × *Var* → *Lexaddr*
```
(define apply-senv
  (lambda (senv var)
    (cond
      ((null? senv)
       (report-unbound-var var))
      ((eqv? var (car senv))
       0)
      (else
        (+ 1 (apply-senv (cdr senv) var))))))
```

Translator 1

translation-of-program : *Program* → *Nameless-program*

```
(define translation-of-program
  (lambda (pgm)
    (cases program pgm
      (a-program (exp1)
        (a-program
          (translation-of exp1 (init-senv)))))))
```

init-senv : () → *Senv*

```
(define init-senv
  (lambda ()
    (extend-senv 'i
      (extend-senv 'v
        (extend-senv 'x
          (empty-senv))))))
```

7

Translator 2

translation-of : *Exp* × *Senv* → *Nameless-exp*

```
(define translation-of
  (lambda (exp senv)
    (cases expression exp
      (const-exp (num) (const-exp num))
      (diff-exp (exp1 exp2)
        (diff-exp
          (translation-of exp1 senv)
          (translation-of exp2 senv)))
      (zero?-exp (exp1)
        (zero?-exp
          (translation-of exp1 senv)))
      (if-exp (exp1 exp2 exp3)
        (if-exp
          (translation-of exp1 senv)
          (translation-of exp2 senv)
          (translation-of exp3 senv)))
```

```
(var-exp (var)
  (nameless-var-exp
    (apply-senv senv var)))
(let-exp (var exp1 body)
  (nameless-let-exp
    (translation-of exp1 senv)
    (translation-of body
      (extend-senv var senv))))
(proc-exp (var body)
  (nameless-proc-exp
    (translation-of body
      (extend-senv var senv))))
(call-exp (rator rand)
  (call-exp
    (translation-of rator senv)
    (translation-of rand senv)))
(else
  (report-invalid-source-expression exp)))))
```

8

Interpretation

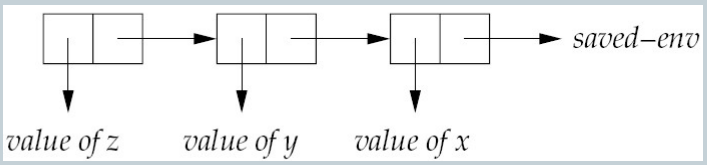
Nameless interpreter

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

New environment interface

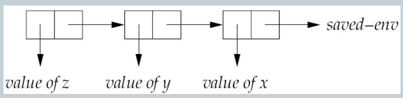
nameless-environment

nameless-environment?	: *SchemeVal* → *Bool*
empty-nameless-env	: () → *Nameless-env*
extend-nameless-env	: *Expval* × *Nameless-env* → *Nameless-env*
apply-nameless-env	: *Nameless-env* × *Lexaddr* → *DenVal*



New environment interface

nameless-environment?	: *SchemeVal* → *Bool*
empty-nameless-env	: () → *Nameless-env*
extend-nameless-env	: *Expval* × *Nameless-env* → *Nameless-env*
apply-nameless-env	: *Nameless-env* × *Lexaddr* → *DenVal*

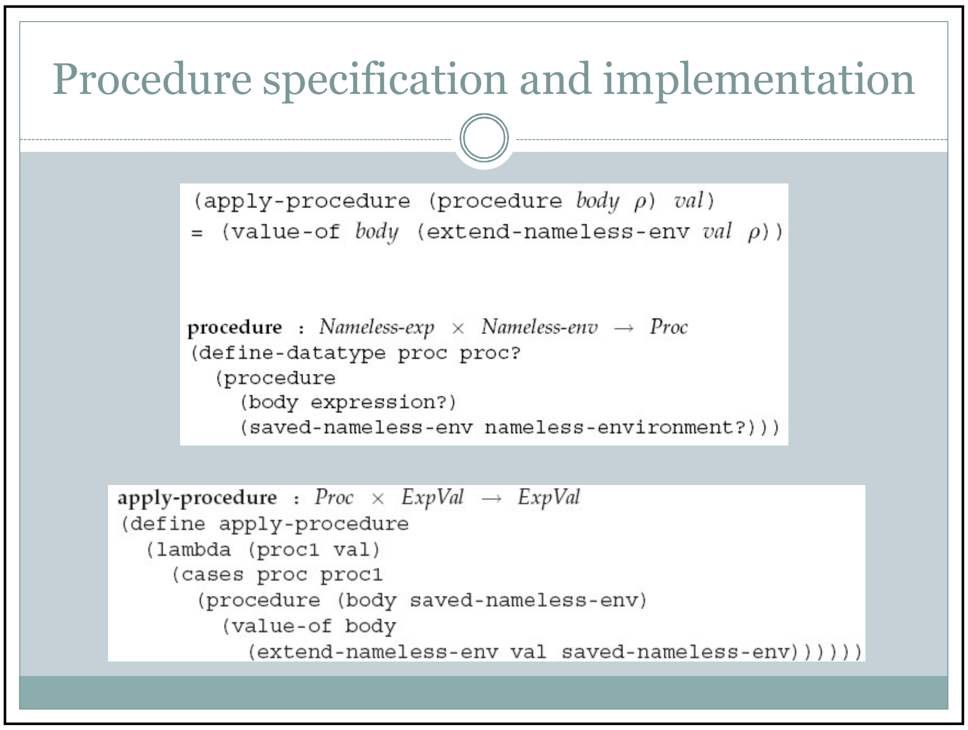


nameless-environment? : *SchemeVal* → *Bool*
```
(define nameless-environment?
  (lambda (x)
    ((list-of expval?) x)))
```

empty-nameless-env : () → *Nameless-env*
```
(define empty-nameless-env
  (lambda ()
    '()))
```

extend-nameless-env : *ExpVal* × *Nameless-env* → *Nameless-env*
```
(define extend-nameless-env
  (lambda (val nameless-env)
    (cons val nameless-env)))
```

apply-nameless-env : *Nameless-env* × *Lexaddr* → *ExpVal*
```
(define apply-nameless-env
  (lambda (nameless-env n)
    (list-ref nameless-env n)))
```



Procedure specification and implementation

```
(apply-procedure (procedure body ρ) val)
= (value-of body (extend-nameless-env val ρ))
```

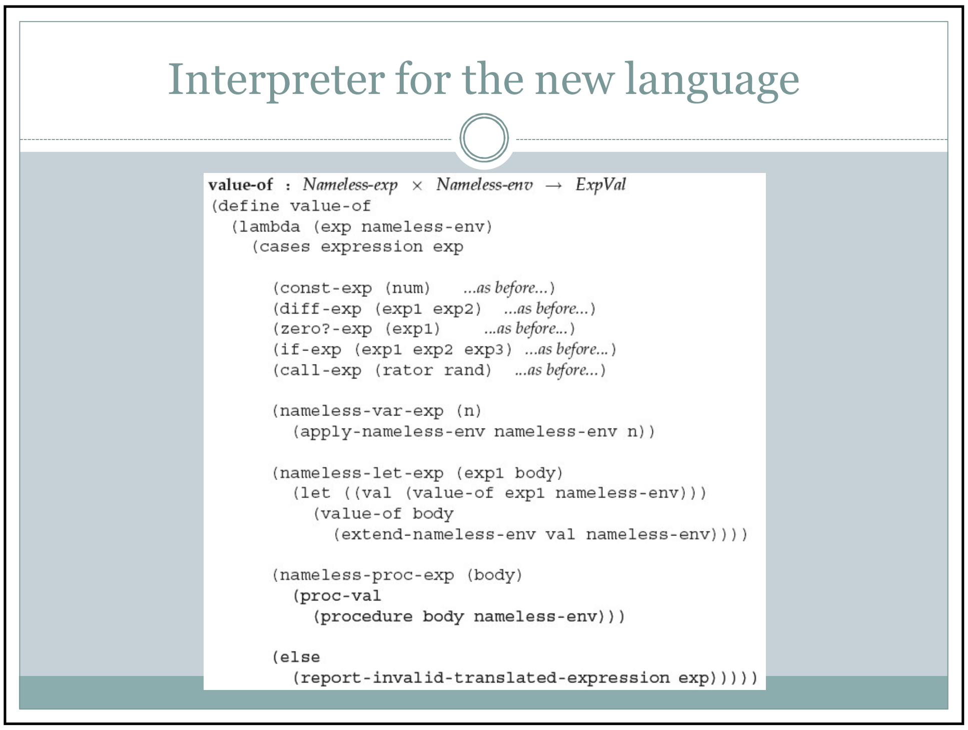
procedure : *Nameless-exp* × *Nameless-env* → *Proc*

```
(define-datatype proc proc?
  (procedure
    (body expression?)
    (saved-nameless-env nameless-environment?)))
```

apply-procedure : *Proc* × *ExpVal* → *ExpVal*

```
(define apply-procedure
  (lambda (proc1 val)
    (cases proc proc1
      (procedure (body saved-nameless-env)
        (value-of body
          (extend-nameless-env val saved-nameless-env))))))
```

13



Interpreter for the new language

value-of : *Nameless-exp* × *Nameless-env* → *ExpVal*

```
(define value-of
  (lambda (exp nameless-env)
    (cases expression exp

      (const-exp (num)    ...as before...)
      (diff-exp (exp1 exp2)    ...as before...)
      (zero?-exp (exp1)    ...as before...)
      (if-exp (exp1 exp2 exp3)  ...as before...)
      (call-exp (rator rand)   ...as before...)

      (nameless-var-exp (n)
        (apply-nameless-env nameless-env n))

      (nameless-let-exp (exp1 body)
        (let ((val (value-of exp1 nameless-env)))
          (value-of body
            (extend-nameless-env val nameless-env))))

      (nameless-proc-exp (body)
        (proc-val
          (procedure body nameless-env)))

      (else
        (report-invalid-translated-expression exp)))))
```

14