Computer Programs Et09-12

dr. H. Corporaal
dr.ir. A.J.C. van Gemund
Computer Systems Group

`http://cs.et.tudelft.nl/~heco`
`email: heco@cs.et.tudelft.nl`

Session 7: The Environment Evaluation Model

Slide 1.7. 1

Overview

- Environment model of evaluation
  - global environment
  - local environment
  - frame
  - binding
  - shadowing
- Procedure object (closure)
- Evaluation rules

Slide 1.7. 2

Substitution model revisited

Given: (define (fun x y)
(+ x y))

Eval: (fun (+ 1 2) (+ 3 4))
↓ substitute x and y in body
(+ (+ 1 2) (+ 3 4))
↓ eval subexpressions
(+ 3 7)
↓ apply + using 3 and 7
10

Slide 1.7. 3

Exit substitution model

```
Given:  ( define   ( fun!  x  y )
             ( sequence
                 ( set!   y  2 )
                 ( +  x  y )))

Eval:   ( fun!   ( +  1  2 ) ( +  3  4 ))
                     ↓ substitute x and y in body
             ( sequence
                 ( set!   ( +  3  4 )  2 )    ← huh !
                 ( +   ( +  1  2 ) ( +  3  4 )))
                     ↓
                     ?
```

Slide 1.7. 4

Exit substitution model

Note, in the (set! y 2) we should not substitute y; it's a special form. Still the answer is not correct !

```
Eval:  ( fun! ( +  1  2 ) ( +  3  4 ))
                  ↓    substitute x and y in body
       ( sequence
           ( set!   y  2 )          ← take care !
           ( +   ( +  1  2 ) ( +  3  4 )))
                  ↓
                  10 (instead of 5)
```

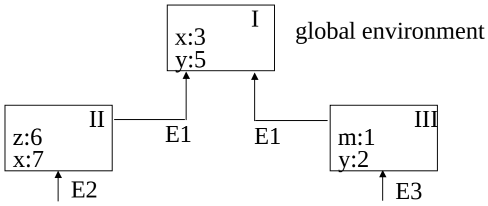
Slide 1.7. 5

What's needed?

- To implement assignment, a model is needed which understands changing values.
- set! and define should be covered
- Multiple instances of <lambda> should be covered (objects and state)

Slide 1.7. 6

Environment, frames and bindings



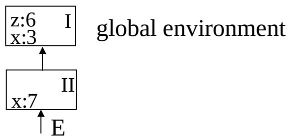
E1
— → pointers to environments
E3

I
II → frames
III

x:3
— → bindings
y:2

Slide 1.7. 7

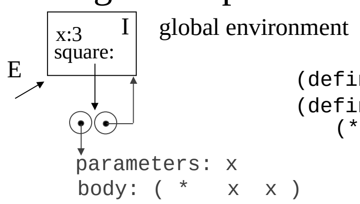
Variables and their values



- The value of a variable with respect to an environment E is the value given by the binding of the variable in the first frame in the environment that contains a binding for that variable
  - the value of <x> in <E> is 7 (bound in frame <II>)
  - the binding <x:7> in frame <II> shadows the binding <x:3> in frame <I>
- The bindings in frames are the places we need to store new values when we use assignments

Slide 1.7. 8

Creating a new procedure




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

- A procedure is created when evaluating <lambda>
- A <lambda> expression is always evaluated in an environment <E>
- A procedure object consists of two parts, the lambda construct (parameters and body) and a pointer to the environment <E>

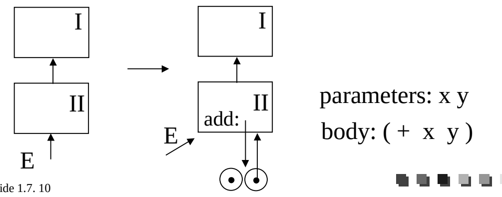
Slide 1.7. 9

Creating a new procedure, 2

E: (define (add x y)
(+ x y))

E: (define add (lambda (x y) (+ x y)))

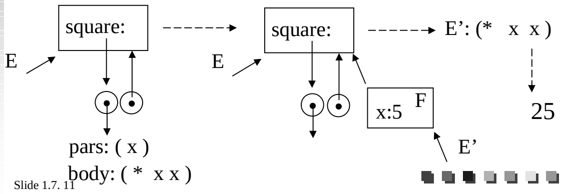
variable name — parameters — body



Slide 1.7. 10

Evaluating a procedure

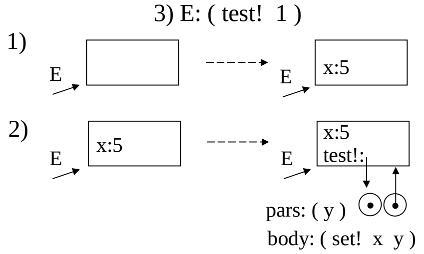
- To evaluate a procedure:
  - create a new frame <F> attached to procedure's env.
  - bind actual arguments in <F>
  - create new environment E' pointing to <F>
  - evaluate the body of the procedure in environment E'
- Example: (square 5)



Slide 1.7. 11

set! again

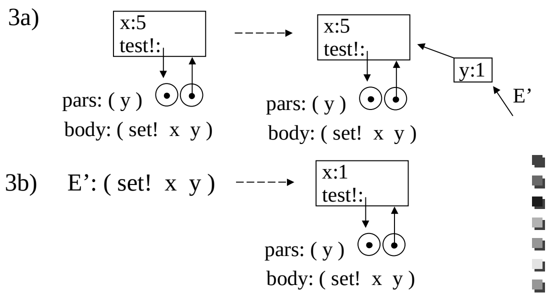
- Example: 1) E: (define x 5)
  2) E: (define (test! y)
  (set! x y))
  3) E: (test! 1)



Slide 1.7. 12

set! again, 2

3) E: (test! 1)



Objects and local state, 1

- Example:

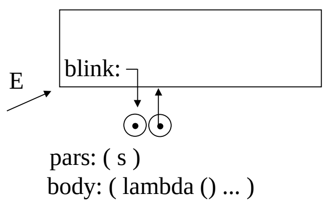
```
( define   ( blink  s )
     ( lambda  ( )
           ( sequence
                ( if  ( =  s 0 ) ( set!   s  1 ) ( set!   s  0 ))
                s )))

( define    B1    ( blink  0 ))
( define    B2    ( blink  1 ))

( B1 )
( B2 )
( B1 )
```

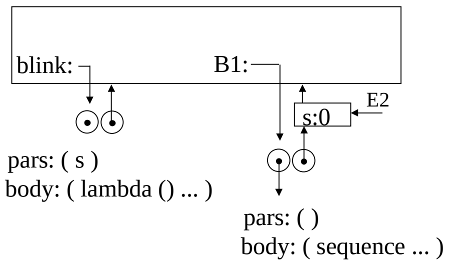

Objects and local state, 2

```
E:  ( define  ( blink   s )
         ( lambda   ( )
              ( sequence
                   ( if  ( =  s  0 ) ( set!  s 1 ) ( set!  s 0 ))
                   s )))
```



Objects and local state, 3

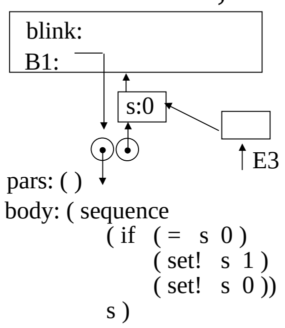
E: (define B1 (blink 0))



Slide 1.7. 16

Objects and local state, 4

E: (B1)



- Evaluate in <E3> : (sequence (if .....) s)

Slide 1.7. 17

Summary

- Environments are needed to model assignment and objects with state
- In order to understand and apply the environment model keep the following in mind:
  - How to create a procedure object (when evaluating a lambda expression)
  - How to apply a procedure object: creation of a frame attached to the procedure environment
  - Handling internal defines
  - Handling set!

Slide 1.7. 18

Examples

- Factorial (exercise 3.9)
  - recursive
  - iterative
- Internal defines
- Objects and let-expressions
- Object generator

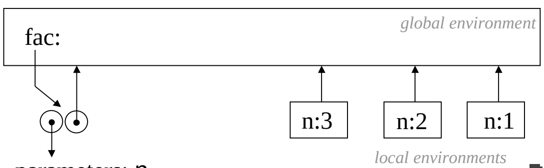
Recursive factorial

```
(define (fac n)
  (if (= n 1)
    1
    (* n (fac (- n 1))) ))

(fac 3)  -->  6
```


Recursive factorial

Evaluate: (fac 3)



parameters: n
body:
```
(if (= n 1)
      1
      (* n (fac (- n 1))))
```

Iterative factorial

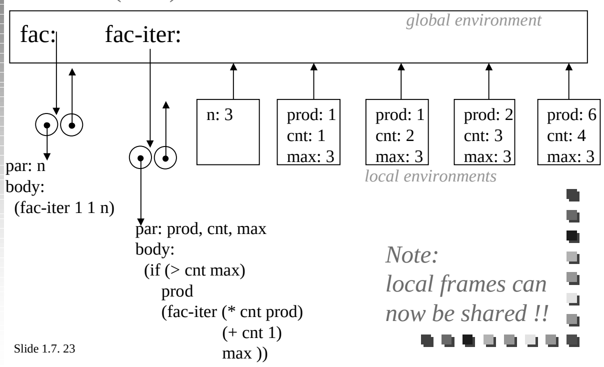
```
(define (fac n)
  (fac-iter 1 1 n))

(define (fac-iter prod cnt max)
  (if (> cnt max)
      prod
      (fac-iter (* cnt prod)
                (+ cnt 1)
                max) ))

(fac 3) -->  6
```


Iterative factorial

Evaluate: (fac 3)



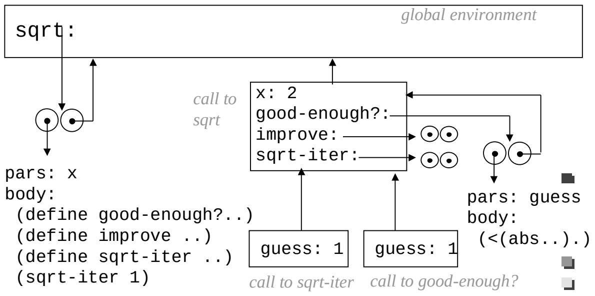
Note:
local frames can
now be shared !!

Internal definitions

```
(define (sqrt x)
  (define (good-enough? guess)
    (< (abs (- (square guess) x)) 0.001) )
  (define (improve guess)
    (average guess (/ x guess)) )
  (define (sqrt-iter guess)
    (if (good-enough? guess)
        guess
        (sqrt-iter (improve guess)) ))
  (sqrt-iter 1) )
```

Internal definitions

Evaluate: (sqrt 2)



Internal definitions

- Internal defined names are bound in the current (i.e. active) frame
- Names of local procedures do not interfere with global ones
- Local procedures can access arguments of the enclosing procedure (so called free variables)
  - e.g. good-enough? can access x

Objects and let-expressions

```
(define object
  (let ((counter 0))
    (lambda (x)
      (set! counter (+ x counter))
      counter )))

(object 1) → 1
(object 3) → 4
(object 2) → 6
```

Objects and let-expressions

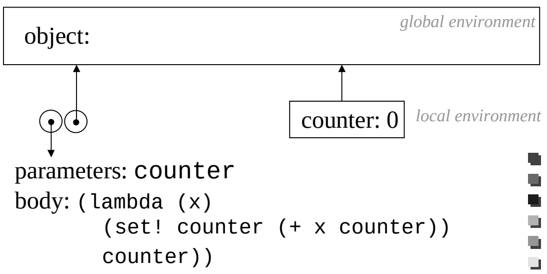
- After let-expansion:

```
(define object
  ((lambda (counter)
     (lambda (x)
       (set! counter (+ x counter))
       counter ))
    0))
```

Slide 1.7. 28

Objects and let-expressions

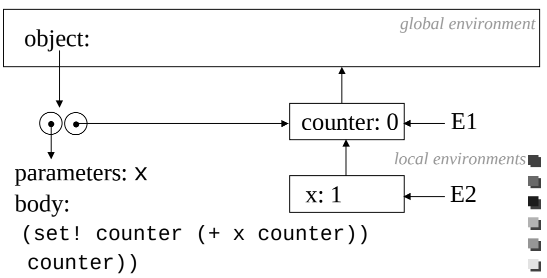
Step 1: evaluating the let-expression



Slide 1.7. 29

Objects and let-expressions

Step 2: evaluating (object 1)



Slide 1.7. 30

Object creator

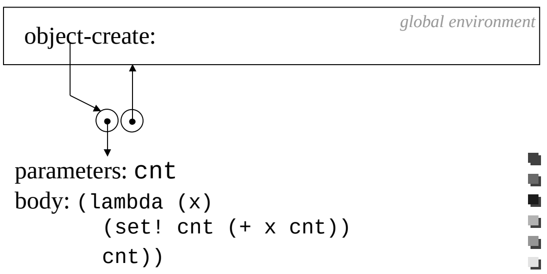
```
(define (obj-create cnt)
  (lambda (x)
    (set! cnt (+ x cnt))
    cnt)))

(define x-obj (obj-create 0))
(define y-obj (obj-create 0))

(x-obj 1) → 1
(x-obj 4) → 5
(y-obj 3) → 3
(x-obj 1) → 6
```

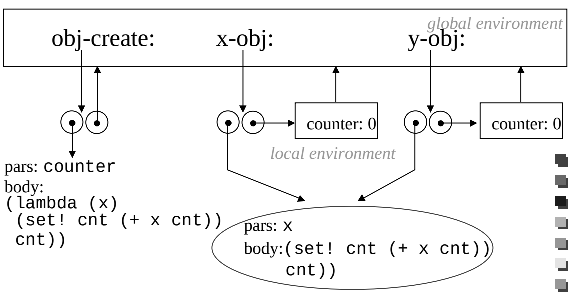

Object creator

Step 1: define ‘obj-create’



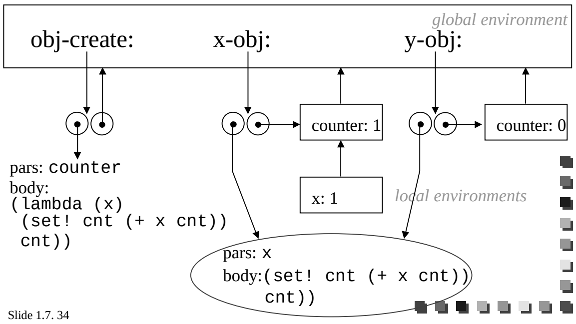
Object creator

Step 2: define ‘x-obj’ and ‘y-obj’



Object creator

Step 3: evaluate (x-obj 1)



Slide 1.7. 34