6.001 SICP – Evaluation I Recitation
11/19/2004

• Eval review
• Evaluation examples
define
lambda
apply
• New language elements

Stages of an interpreter

Lexical analyzer
Parser
Evaluator
Environment
Printer

Arithmetic calculator

(define (tag-check e sym) (and (pair? e) (eq? (car e) sym)))
(define (sum? e) (tag-check e 'plus*))

(define (eval exp)
  (cond
   ((number? exp) exp)
   ((sum? exp)    (eval-sum exp))
   (else
    (error "unknown expression " exp))))

(define (eval-sum exp)
  (+ (eval (cadr exp)) (eval (caddr exp)))))

(eval '(plus* 24 (plus* 5 6)))

We are just walking through a tree ...

sum? checks the tag
The Evaluators

1. arithmetic calculator
2. add names: quote, define, environment is a table
3. add conditionals: eval-greater, eval-if, first apply
4. operators in environment: make-primitive, scheme-apply
5. environment as parameter to eval eval-special and lookup
6. compound procedures
   • lambda evaluates to double bubble (list of params, body, encl.env.)
   • apply on double bubble drops frame from encl.env., binds names to values, evals body in new env.
   • environment is now a list of tables

Eval 6 review - eval
(define (eval exp env)
  (cond ((number? exp) exp)
        ((symbol? exp) (lookup exp env))
        ((define? exp) (eval-define exp env))
        ((if? exp) (eval-if exp env))
        ((lambda? exp) (eval-lambda exp env))
        ((application? exp) (apply (eval (car exp) env) (map (lambda (e) (eval e env)) (cdr exp))))
        (else (error "unknown expression " exp)))))

(define (define? exp) (tag-check exp 'define*))
(define (if? exp) (tag-check exp 'if*))
(define (application? e) (pair? e))
(define (lambda? e) (tag-check e 'lambda*))

Eval 6 review - env
(define (extend-env-with-new-frame names values env)
  (let ((new-frame (make-table))
        (make-bindings! names values new-frame)
        (cons new-frame env)))
(define (make-bindings! names values table)
  (for-each (lambda (name value) (table-put! table name value)) names values))

; the initial global environment
(define GE (extend-env-with-new-frame (list 'plus* 'greater*)
                                (list (make-primitive +) (make-primitive >))
                                nil))

Eval 6 review - lookup / define
; lookup searches the list of frames for the first match
(define (lookup name env)
  (if (null? env) (error "unbound variable: " name)
   (let ((binding (table-get (car env) name)))
     (if (null? binding)
       (lookup name (cdr env))
       binding-value binding))))

; define changes the first frame in the environment
(define (eval-define exp env)
  (let ((name (cadr exp))
        (defined-to-be (caddr exp)))
    (table-put! (car env) name (eval defined-to-be env))
    'undefined))

Eval 6 review - if
(define (eval-if exp env)
  (let ((predicate (cadr exp))
        (consequent (caddr exp))
        (alternative (cadddr exp)))
    (cond ((eq? test #t) consequent)
          ((eq? test #f) alternative)
          (else (error "unilateral boolean test")
                predicate))))

Eval 6 review - lambda
(define (eval-lambda exp lambda)
  (let ((args (cadr lambda))
        (body (caddr lambda)))
    (make-compound args body env)))

; ADT that implements the "double bubble"
(define (compound? exp) (tag-check exp 'compound))
(define (map-compound parameters body env)
  (list compound-tag parameters body env))
(define (apply-compound exp (tag-check exp compound-tag))
  (define (parameters compound) (cadr compound))
  (define (body compound) (cadddr compound))
  (define (env compound) (cadddr compound))
Eval 6 review - apply

```
(define scheme-apply apply)
(define (apply operator operands)
  (cond ((primitive? operator)
          (scheme-apply (get-scheme-procedure operator) operands))
        ((compound? operator)
          (eval (body operator)
                (extend-env-with-new-frame
                 (parameters operator) operands
                 (env operator))))
        (else (error "operator not a procedure: 
                 operator")))
```

Eval examples

```
; assume (table-put! (car GE) \times* (make-primitive *))

(eval '(define* z* (plus* 1 3)) GE)
(eval '(define* mpy* (lambda* (x* y*) \times* x* y*)) GE)
(eval '(mpy* 3 z*) GE)
```

evaluation example

```
(eval '(define* z* (plus* 1 3)) GE)

(define* z* (plus* 1 3))    | GE
(plus* 1 3) | GE
```

evaluation example

```
(define (eval exp env)
  (cond ((number? exp) exp)
        ((symbol? exp) (lookup exp env))
        ((define? exp) (eval-define exp env))
        ((if? exp) (eval-if exp env))
        ((lambda? exp) (eval-lambda exp env))
        ((application? exp) (apply (eval (car exp) env)
                                   (map (lambda (e) (eval e env))
                                        (cdr exp))))
        (else (error "unknown expression " exp))))
```

evaluation example

```
(eval '(define* z* (plus* 1 3)) GE)

(define* z* (plus* 1 3))    | GE
(plus* 1 3) | GE
```

```
(define (eval-define exp env)
  (let ((name (cadr exp))
        (defined-to-be (caddr exp)))
    (table-put! (car env) name (eval defined-to-be env)
                \undefined))
```

```
(define (eval-lambda exp env)
  (let ((name (cadr exp))
        (defined-to-be (caddr exp)))
    (table-put! (car env) name (eval defined-to-be env)
                \undefined))
```

```
(define (eval-if exp env)
  (let ((test (cadr exp))
        (consequent (caddr exp)))
    (if (eval test env)
        (eval consequent env)
        (eval (caddr (rest (rest exp))) env)))
```

```
(define (eval-exp env)
  (cond ((number? exp) exp)
        ((symbol? exp) (lookup exp env))
        ((define? exp) (eval-define exp env))
        ((if? exp) (eval-if exp env))
        ((lambda? exp) (eval-lambda exp env))
        ((application? exp) (apply (eval (car exp) env)
                                   (map (lambda (e) (eval e env))
                                        (cdr exp))))
        (else (error "unknown expression " exp))))
```
evaluation example

(eval '(define* z* (plus* 1 3)) GE)

(define* z* (plus* 1 3)) | GE
(plus* 1 3) | GE
1 | GE
3 | GE

(define (eval exp env)
  (cond ((number? exp) exp)
        ((symbol? exp) (lookup exp env))
        ((define? exp) (eval-define exp env))
        ((if? exp) (eval-if exp env))
        ((lambda? exp) (eval-lambda exp env))
        ((application? exp) apply
          (eval (car exp) env)
          (map (lambda (e) (eval e env)) (cdr exp)))
        (else (error "unknown expression " exp)))))

evaluation example

(eval '(define* z* (plus* 1 3)) GE)

(define* z* (plus* 1 3)) | GE
(plus* 1 3) | GE
1 | GE ==> 1
3 | GE ==> 3

(define (eval exp env)
  (cond ((number? exp) exp)
        ((symbol? exp) (lookup exp env))
        ((define? exp) (eval-define exp env))
        ((if? exp) (eval-if exp env))
        ((lambda? exp) (eval-lambda exp env))
        ((application? exp) apply
          (eval (car exp) env)
          (map (lambda (e) (eval e env)) (cdr exp))
        ))
        (else (error "unknown expression " exp)))))

evaluation example

(eval '(define* z* (plus* 1 3)) GE)

(define* z* (plus* 1 3)) | GE
(plus* 1 3) | GE
1 | GE ==> 1
3 | GE ==> 3
plus* | GE
(define (eval exp env)
  (cond ((number? exp)       exp)
        ((symbol? exp)      (lookup exp env))
        ((define? exp)      (eval-define exp env))
        ((if? exp)          (eval-if exp env))
        ((lambda? exp)      (eval-lambda exp env))
        ((application? exp) (apply (eval (car exp) env)
                                  (map (lambda (e) (eval e env))
                                  (cdr exp))))
        (else (error "unknown expression " exp)))))

evaluation example

(eval '(define* z* (plus* 1 3)) GE)

(define* z* (plus* 1 3))    | GE
(plus* 1 3)                | GE
1      | GE ==> 1
3      | GE ==> 3
plus*  | GE ==> (primitive [prim +])

(define scheme-apply apply)

(define (apply operator operands)
  (cond ((primitive? operator)
             (scheme-apply (get-scheme-procedure operator) operands))
       ((compound? operator)
             (eval (body operator)
                   (extend-env-with-new-frame
                    (parameters operator) operands
                    (env operator))))
       (else (error "operator not a procedure: " operator)))))

evaluation example

(eval '(define* z* (plus* 1 3)) GE)

(define* z* (plus* 1 3))    | GE
(plus* 1 3)                | GE
1      | GE ==> 1
3      | GE ==> 3
plus*  | GE ==> (primitive [prim +])

(define scheme-apply apply)

(define (apply operator operands)
  (cond ((primitive? operator)
             (scheme-apply (get-scheme-procedure operator) operands))
       ((compound? operator)
             (eval (body operator)
                   (extend-env-with-new-frame
                    (parameters operator) operands
                    (env operator))))
       (else (error "operator not a procedure: " operator)))))

(define scheme-apply apply)

(define (apply operator operands)
  (cond ((primitive? operator)
             (scheme-apply (get-scheme-procedure operator) operands))
       ((compound? operator)
             (eval (body operator)
                   (extend-env-with-new-frame
                    (parameters operator) operands
                    (env operator))))
       (else (error "operator not a procedure: " operator)))))
### Evaluation Example

```scheme
(cons (eval '(define* mpy* (lambda* (x* y*) (times* x* y*))) GE) (define* mpy* (lambda* (x* y*) (times* x* y*))) GE)
```

### Evaluation Example

```scheme
(eval* '(define* mpy* (lambda* (x* y*) (times* x* y*))) GE)
```

### Evaluation Example

```scheme
(let (args (cadr exp))
  (body (caddr exp)))
```

### Evaluation Example

```scheme
(define compound-tag 'compound)
```

### Evaluation Example

```scheme
(define (eval exp env)
  (cond ((number? exp) exp)
        ((symbol? exp) (lookup exp env))
        ((define? exp) (eval-define exp env))
        ((if? exp) (eval-if exp env))
        ((lambda? exp) (eval-lambda exp env))
        ((application? exp) (apply (eval (car exp) env) (map (lambda (e) (eval e env)) (cdr exp))))
        (else (error "unknown expression" exp)))))
```
evaluation example

SICP 6.001

(eval '(define* mpy* (lambda* (x* y*) (times* x* y*))) GE)

(define mpy* (lambda* (x* y*) (times* x* y*))) | GE
(lambda* (x* y*) (times* x* y*)) | GE
(make-compound '(x* y*) '(times* x* y*) GE)

==> (list 'compound '(x* y*) '(times* x* y*) GE)

;modified environment!

SICP 6.001

evaluation example

SICP 6.001

;assume (table-put! (car GE) 'times* (make-primitive *))

(eval '(define z* (plus* 1 3)) GE)
(eval '(define mpy* (lambda* (x* y*) (times* x* y*))) GE)
(eval '(mpy* 3 z*) GE)

SICP 6.001

evaluation example

SICP 6.001

(eval '(mpy* 3 z*) GE)

SICP 6.001

;modified environment!

define (eval exp env)

|cond|number? exp|symbol? exp|define? exp|if? exp|lambda? exp|application? exp|
|exp|lookup exp env|eval-define exp env|eval-if exp env|eval-lambda exp env|apply (eval (car exp) env)
|else (error "unknown expression") exp)

SICP 6.001

evaluation example

SICP 6.001

(eval '(mpy* 3 z*) GE)

SICP 6.001

(mpy* 3 z*) | GE

SICP 6.001

;modified environment!
(define scheme-apply apply)

(define (apply operator operands)
  (cond ((primitive? operator)
     (scheme-apply (get-scheme-procedure operator) operands))
    ((compound? operator)
     (eval (body operator)
           (extend-env-with-new-frame
            (parameters operator)
            operands
            (env operator)))))
    (else (error "operator not a procedure: " operator))))

evaluation example
(eval '(mpy* 3 z*) GE)

(mpy* 3 z*) | GE
mpy* | GE ==> *(compound ...)
3 | GE ==> 3
z* | GE ==> 4

(apply (list 'compound '(x* y*) '(times* x* y*) GE)
  '(3 4))
(ttimes* x* y*) |
(extend-env-with-new-frame '(x* y*) '(3 4) GE)
  [ E1--> (x*:3, y*:4) --> GE ]

(ttimes* x* y*) | E1
[scheme-apply (print *) 3 4]
  ==> 12!

evaluation example
(eval '(define class (quote* is-over)))

(define class (quote* is-over)) | GE
(quote* is-over) | GE

unbound variable: quote*

evaluation example
(eval '(define class (quote* is-over)))

(define class (quote* is-over)) | GE
(quote* is-over) | GE

unbound variable: quote*

(define (eval exp env)
  (cond ((number? exp) exp)
    ((quote? exp) (eval-quote exp))
    ((symbol? exp) (lookup exp env))
    ((define? exp) (eval-define exp env))
    ((if? exp) (eval-if exp env))
    ((lambda? exp) (eval-lambda exp env))
    ((application? exp) (apply
     (eval (car exp) env)
     (map (lambda (e) (eval e env))
          (cdr exp)))
     (else (error "unknown expression " exp)))))
(define quote? (tag-check exp 'quote*))

(define (eval-quote exp) (cadr exp))

evaluation example
(eval '(define class (quote* is-over)))

(define class (quote* is-over)) | GE
(quote* is-over) | GE
is-over

(table-put! (car GE) 'class 'is-over)
Summary

• Eval / Apply cycle is core of eval.
• eval calls apply with operator and args
• apply calls eval with expression and env
• no pending operations on either call – if expression is iterative so is evaluation

Exercise

Extend the define* evaluator so define* also returns a value.

e.g. (define* a 16) => 16

Eval 6 review - lookup / define

; lookup searches the list of frames for the first match
(define (lookup name env)
  (if (null? env)
      (error "unbound variable: " name)
      (let ((binding (table-get (car env) name)))
        (if (null? binding)
            (lookup name (cdr env))
            (binding-value binding))))

; define changes the first frame in the environment
(define (eval-define exp env)
  (let ((name (cadr exp))
        (defined-to-be (caddr exp)))
    (table-put! (car env) name (eval defined-to-be env))
    'undefined))

Exercise

Extend our language with the and operator.

(and <exp1> <exp2> .. <expn>)

Exercise -- add AND

(define (tag-check e sym) (and (pair? e) (eq? (car e) sym)))
(define (and? e) (tag-check e 'and*))

(define (eval exp)
  (cond
   ((number? exp) exp)
   ((boolean? exp) exp)
   ((and? exp) (eval-and exp))
   (else
    (error "unknown expression " exp)))))

(define (eval-and exp)
  (define (try-next terms)
    (if (null? terms)
        #t
        (let ((first-term (eval (car terms))))
          (if first-term
              (try-next (cdr terms))
              '#f))))
  (try-next (cdr exp)))

Exercise - 4

Modify the interpreter so we can write infix assignments instead.

(define MyVar 10) ➔
(myVar := 10)
Eval 6 review - lookup / define

; lookup searches the list of frames for the first match
(define (lookup name env)
  (if (null? env)
      (error "unbound variable: " name)
      (let ((binding (table-get (car env) name)))
        (if (null? binding)
            (lookup name (cdr env))
            (binding-value binding))))

; define changes the first frame in the environment
(define (eval-define exp env)
  (let ((name  (cadr exp))
          (defined-to-be (caddr exp))
          (table-put! (car env) name (eval defined-to-be env))
          'undefined))

680 SKF 3