

Procedural Representation

Geoffrey Matthews

March 17, 2023

lookup

```
1 ;; lookup : symbol env! RCFAE-Value
2 (define (lookup name env)
3   (type-case Env env
4     [mtSub () (error 'lookup "no binding for identifier")]
5     [aSub (bound-name bound-value rest-env)
6      (if (symbol=? bound-name name)
7          bound-value
8          (lookup name rest-env))]]
9     [aRecSub (bound-name boxed-bound-value rest-env)
10      (if (symbol=? bound-name name)
11          (unbox boxed-bound-value)
12          (lookup name rest-env))]]))
13 (lookup name rest-env))]]))
14
15
16 (define (Env? x)
17   (procedure? x))
18 ;; mtSub : ()!Env
19 (define (mtSub)
20   (lambda (name)
21     (error 'lookup "no binding for identifier")))
22 ;; aSub: symbol FAE-Value Env!Env
23 (define (aSub bound-name bound-value env)
24   (lambda (want-name)
25     (cond
26       [(symbol=? want-name bound-name) bound-value]
27       [else (lookup want-name env)])))
28 ;; lookup : symbol Env! FAE-Value
29 (define (lookup name env)
30   (env name))
```

Figure 10.2: Recursion: Interpreter

```

1 ;; cyclically-bind-and-interp : symbol RCFAE env! env
2 (define (cyclically-bind-and-interp bound-id named-expr env)
3   (local ([define value-holder (box (numV 1729))])
4     [define new-env (aRecSub bound-id value-holder env)]
5     [define named-expr-val (interp named-expr new-env)])
6   (begin
7     (set-box! value-holder named-expr-val)
8     new-env)))
9
10
11 ;; interp : RCFAE env! RCFAE-Value
12 (define (interp expr env)
13   (type-case RCFAE expr
14     [num (n) (numV n)]
15     [add (l r) (num+ (interp l env) (interp r env))]
16     [mult (l r) (num* (interp l env) (interp r env))]
17     [if0 (test truth falsity)
18      (if (num-zero? (interp test env))
19          (interp truth env)
20          (interp falsity env))]
21     [id (v) (lookup v env)]
22     [fun (bound-id bound-body)
23      (closureV bound-id bound-body env)]
24     [app (fun-expr arg-expr)
25      (local ([define fun-val (interp fun-expr env)])
26        (interp (closureV-body fun-val)
27                 (aSub (closureV-param fun-val)
28                       (interp arg-expr env)
29                          (closureV-env fun-val)))))]
30     [rec (bound-id named-expr bound-body)
31      (interp bound-body
32               (cyclically-bind-and-interp bound-id
33                                             named-expr
34                                             env)))]))

```

Procedural Representation of Procedures

```
1
2 (define-type FAE-Value
3   [numV (n number?)]
4   [closureV (p procedure?)])
5 ;; interp : FAE Env->FAE-Value
6 (define (interp expr env)
7   (type-case FAE expr
8     [num (n) (numV n)]
9     [add (l r) (num+ (interp l env) (interp r env))]
10    [id (v) (lookup v env)]
11    [fun (bound-id bound-body)
12     (closureV (lambda (arg-val)
13                 (interp bound-body
14                           (aSub bound-id arg-val env))))])
15    [app (fun-expr arg-expr)
16     (local ([define fun-val (interp fun-expr env)]
17             [define arg-val (interp arg-expr env)])
18       ((closureV-p fun-val)
19        arg-val)))]))
```

Meta-circular interpreter

```
1
2 (define (number-or-procedure? v)
3   (or (number? v)
4       (procedure? v)))
5 (define-type Env
6   [mtSub]
7   [aSub (name symbol?) (value number-or-procedure?) (env Env?)])
8 ;; lookup : symbol Env !number-or-procedure
9 (define (lookup name env)
10  (type-case Env env
11    [mtSub () (error 'lookup "no binding for identifier")]
12    [aSub (bound-name bound-value rest-env)
13      (if (symbol=? bound-name name)
14          bound-value
15          (lookup name rest-env))]))
16 ;; interp : FAE Env !number-or-procedure
17 (define (interp expr env)
18  (type-case FAE expr
19    [num (n) n]
20    [add (l r) (+ (interp l env) (interp r env))]
21    [id (v) (lookup v env)]
22    [fun (bound-id bound-body)
23      (lambda (arg-val)
24        (interp bound-body
25          (aSub bound-id arg-val env)))]
26    [app (fun-expr arg-expr)
27      (local ([define fun-val (interp fun-expr env)]
28              [define arg-val (interp arg-expr env)])
29        (fun-val arg-val)))]))
```