Procedural Representation

Geoffrey Matthews

March 17, 2023

lookup

```
;; lookup : symbol env! RCFAE-Value
(define (lookup name env)
  (type-case Env env
        [mtSub () (error ‘lookup "no binding for identifier")]
        [aSub (bound-name bound-value rest-env)
        (if (symbol=? bound-name name)
            bound-value
          (lookup name rest-env))]
        [aRecSub (bound-name boxed-bound-value rest-env)
           (if (symbol=? bound-name name)
         (unbox boxed-bound-value)
       (lookup name rest-env))]))
       (lookup name rest-env))]))


(define (Env? x)
  (procedure? x))
;; mtSub : ()!Env
(define (mtSub)
  (lambda (name)
    (error 'lookup "no binding for identifier")))
;; aSub: symbol FAE-Value Env!Env
(define (aSub bound-name bound-value env)
  (lambda (want-name)
    (cond
     [(symbol=? want-name bound-name) bound-value]
     [else (lookup want-name env)])))
;; lookup : symbol Env! FAE-Value
(define (lookup name env)
  (env name))
```

Figure 10.2: Recursion: Interpreter

```
;; cyclically-bind-and-interp : symbol RCFAE env! env
(define (cyclically-bind-and-interp bound-id named-expr env)
  (local ([define value-holder (box (numV 1729))]
    [define new-env (aRecSub bound-id value-holder env)]
    [define named-expr-val (interp named-expr new-env)])
   (begin
    (set-box! value-holder named-expr-val)
    new-env)))


;; interp : RCFAE env! RCFAE-Value
(define (interp expr env)
  (type-case RCFAE expr
       [num (n) (numV n)]
       [add (l r) (num+ (interp l env) (interp r env))]
       [mult (l r) (num* (interp l env) (interp r env))]
       [if0 (test truth falsity)
      (if (num-zero? (interp test env))
          (interp truth env)
        (interp falsity env))]
       [id (v) (lookup v env)]
       [fun (bound-id bound-body)
      (closureV bound-id bound-body env)]
       [app (fun-expr arg-expr)
      (local ([define fun-val (interp fun-expr env)])
       (interp (closureV-body fun-val)
         (aSub (closureV-param fun-val)
               (interp arg-expr env)
               (closureV-env fun-val))))]
       [rec (bound-id named-expr bound-body)
      (interp bound-body
        (cyclically-bind-and-interp bound-id
                  named-expr
                  env))]))
```

Procedural Representation of Procedures

```

(define-type FAE-Value
  [numV (n number?)]
  [closureV (p procedure?)])
;; interp : FAE Env->FAE-Value
(define (interp expr env)
  (type-case FAE expr
       [num (n) (numV n)]
       [add (l r) (num+ (interp l env) (interp r env))]
       [id (v) (lookup v env)]
       [fun (bound-id bound-body)
      (closureV (lambda (arg-val)
           (interp bound-body
             (aSub bound-id arg-val env))))]
       [app (fun-expr arg-expr)
      (local ([define fun-val (interp fun-expr env)]
        [define arg-val (interp arg-expr env)])
       ((closureV-p fun-val)
        arg-val))]))
```

Meta-circular interpreter

```

(define (number-or-procedure? v)
  (or (number? v)
      (procedure? v)))
(define-type Env
  [mtSub]
  [aSub (name symbol?) (value number-or-procedure?) (env Env?)])
;; lookup : symbol Env !number-or-procedure
(define (lookup name env)
  (type-case Env env
       [mtSub () (error 'lookup "no binding for identifier")]
       [aSub (bound-name bound-value rest-env)
       (if (symbol=? bound-name name)
           bound-value
         (lookup name rest-env))]))
;; interp : FAE Env!number-or-procedure
(define (interp expr env)
  (type-case FAE expr
       [num (n) n]
       [add (l r) (+ (interp l env) (interp r env))]
       [id (v) (lookup v env)]
       [fun (bound-id bound-body)
      (lambda (arg-val)
        (interp bound-body
          (aSub bound-id arg-val env)))]
       [app (fun-expr arg-expr)
      (local ([define fun-val (interp fun-expr env)]
        [define arg-val (interp arg-expr env)])
       (fun-val arg-val))]))
```