Recitation Notes, April 22

Execution Objects and the Analyze Evaluator

In the analyze evaluator (A-EVAL), the analyze procedure creates an execution object, which is a procedure that can be used to complete an evaluation by applying it to an environment. For example, the analyze-self-evaluating procedure is

\[
\text{(define (analyze-self-evaluating exp)}
\text{(lambda (env) exp))}
\]

1. Draw a picture representing the execution object used in ((analyze '1) the-global-environment).

The Analyze Evaluator

\[
\text{(define (eval exp env)}
\text{((analyze exp) env))}
\]

\[
\text{(define (analyze exp)}
\text{(cond ((self-evaluating? exp) (analyze-self-evaluating exp))}
\text{((quoted? exp) (analyze-quoted exp))}
\text{((variable? exp) (analyze-variable exp))}
\text{((assignment? exp) (analyze-assignment exp))}
\text{((definition? exp) (analyze-definition exp))}
\text{((if? exp) (analyze-if exp))}
\text{((lambda? exp) (analyze-lambda exp))}
\text{((begin? exp) (analyze-sequence (begin-actions exp)))}
\text{((cond? exp) (analyze (cond->if exp))}
\text{((application? exp) (analyze-application exp))}
\text{(else}
\text{\text{(error "Unknown expression type -- ANALYZE" exp))))}
\]
(define (analyze-self-evaluating exp)
  (lambda (env) exp))

2. Complete the following:

(define (analyze-quoted exp)
  (let ((qval (text-of-quotration exp)))
    (lambda (env) qval)))

(define (analyze-variable exp)
  (lambda (env) (lookup-variable-value exp env)))

(define (analyze-assignment exp)
  (let ((var (assignment-variable exp))
        (val-exe (analyze (assignment-value exp))))
    (lambda (env)
      (set-variable-value! var env 'ok))))

(define (analyze-definition exp)
  (let ((var (definition-variable exp))
        (val-exe (analyze (definition-value exp))))
    (lambda (env)
      (define-variable! var env 'ok))))

(define (analyze-if exp)
  (let ((pred-exe (analyze (if-predicate exp)))
        (con-exe (analyze (if-consequent exp)))
        (alt-exe (analyze (if-alternative exp))))
    (lambda (env)
      (if (true? (pred-exe env))
          (con-exe env)
          (alt-exe env)))))

3. Draw the exe object for (analyze '(if #t 1 2) t-g-e):
(define (analyze-lambda exp)
  (let ((vars (lambda-parameters exp))
         (body-exe (analyze-sequence (lambda-body exp)))
         (lambda (env) (make-procedure vars body-exe env)))
    (lambda-env make-procedure vars body-exe env))

4. Complete the following:

(defun (analyze-application exp)
  (let ((op-exe (analyze (operator exe)))
         (arg-exes )
         (lambda (env)
           (execute-application op-exe env)
           (map (lambda (arg-exe) (arg-exe env))
                arg-exes))))

(defun (execute-application proc args)
  (cond ((primitive-procedure? proc)
         (apply-primitive-procedures proc args))
        ((compound-procedure? proc)
         (procedure-body proc)
         (extend-environment (procedure-parameters proc)
                           args
                           (procedure-environment proc)))
       (else (error "Unknown procedure type" proc)))

(defun (analyze-sequence exps)
  (define (sequentially exe1 exe2)
    (lambda (env) (exe1 env) (exe2 env)))
  (define (loop first-exe rest-exes)
    (if (null? rest-exes)
        first-exe
        (loop (sequentially first-exe (car rest-exes))
               (cdr rest-exes))))
  (let ((exes (map analyze exps)))
    (if (null? exes)
        (error "Empty sequence -- ANALYZE")
        (loop (car exes) (cdr exes))))

5. Draw the exe object for (analyze '(lambda (y) (* y y))):