Code Printing for EOPL

Dieter Castel

January 23, 2015

3 Expressions

```
#lang eopl
(require "syntax.rkt")
(provide (all-defined-out))

;; Semantics
;;; an expressed value is either a number, or a boolean.

(define-datatype expval expval?
  (num-val
   (value number?))
  (bool-val
   (boolean boolean?)))

(define expval->string
  (lambda (v)
    (cases expval v
      (num-val (num) (string-append "Number: "
   (number->string num)))
      (bool-val (bool) (string-append "Boolean: " (if
   bool "#t" "#f"))))))

;; expval->num : ExpVal -> Int
;; Page: 70
(define expval->num
  (lambda (v)
    (cases expval v
      (num-val (num) num)
      (else (expval-extractor-error 'num v)))))

;; expval->bool : ExpVal -> Bool
;; Page: 70
(define expval->bool
  (lambda (v)
```

```scheme
    (cases expval v
      (bool-val (bool) bool)
      (else (expval-extractor-error 'bool v)))))

(define expval-extractor-error
  (lambda (variant value)
    (eopl:error 'expval-extractors "Looking for a ~s, found ~s"
                variant value)))

;; Environments

(define-datatype environment environment?
  (empty-env)
  (extend-env
   (bvar symbol?)
   (bval expval?)
   (saved-env environment?)))

(define apply-env
  (lambda (env search-sym)
    (cases environment env
      (empty-env ()
                 (eopl:error 'apply-env "No binding for ~s" search-sym))
      (extend-env (bvar bval saved-env)
                  (if (eqv? search-sym bvar)
                      bval
                      (apply-env saved-env search-sym)))))))

;; init-env : () -> Env
;; usage: (init-env) = [i=1, v=5, x=10]
;; (init-env) builds an environment in which i is bound to the
;; expressed value 1, v is bound to the expressed value 5, and x is
;; bound to the expressed value 10.
;; Page: 69

(define init-env
  (lambda ()
    (extend-env
     'i (num-val 1)
     (extend-env
      'v (num-val 5)
      (extend-env
       'x (num-val 10)
       (empty-env))))))
```

```
;; Interpreter for LET

;; value-of-program : Program -> ExpVal
;; Page: 71
(define value-of-program
  (lambda (pgm)
    (cases program pgm
      (a-program (exp1)
                 (value-of exp1 (init-env))))))

;; value-of : Exp * Env -> ExpVal
;; Page: 71
(define value-of
  (lambda (exp env)
    (cases expression exp

      (const-exp (num) (num-val num))

      (var-exp (var) (apply-env env var))

      (diff-exp (exp1 exp2)
                (let ((val1 (value-of exp1 env))
                      (val2 (value-of exp2 env)))
                  (let ((num1 (expval->num val1))
                        (num2 (expval->num val2)))
                    (num-val
                     (- num1 num2)))))

      (minus-exp (exp1) ;3.6 addition
                 (let ((val1 (value-of exp1 env)))
                   (let ((num1 (expval->num val1)))
                     (num-val (- 0 num1)))))

      (zero?-exp (exp1)
                 (let ((val1 (value-of exp1 env)))
                   (let ((num1 (expval->num val1)))
                     (if (zero? num1)
                         (bool-val #t)
                         (bool-val #f)))))

      (equal?-exp (exp1 exp2)
                  (let ((val1 (value-of exp1 env))
                        (val2 (value-of exp2 env)))
                    (if (equal? val1 val2)
                        (bool-val #t)
                        (bool-val #f))))

      (if-exp (exp1 exp2 exp3)
              (let ((val1 (value-of exp1 env)))
                (if (expval->bool val1)
                    (value-of exp2 env)
                    (value-of exp3 env))))

      (let-exp (var exp1 body)
                (let ((val1 (value-of exp1 env)))
                 (value-of body
                           (extend-env var val1 env))))

      )))
```

Listing 1: LET interpreter

```
#lang eopl
(provide (all-defined-out))

;;; Syntax
(define-datatype program program?
  (a-program (exp1 expression?)))

(define-datatype expression expression?
  (const-exp (num number?))
  (diff-exp (exp1 expression?) (exp2 expression?))
  (minus-exp (exp1 expression?)) ; 3.6 addition
  (zero?-exp (exp1 expression?))
  (equal?-exp (exp1 expression?) (exp2 expression?)) ;
   3.8 addition
  (if-exp
   (exp1 expression?)
   (exp2 expression?)
   (exp3 expression?))
  (var-exp (var symbol?))
  (let-exp
   (var symbol?)
   (exp1 expression?)
   (body expression?)))

(define (program->string pgm)
     (cases program pgm
      (a-program (exp1)
                 (exp->string exp1 ))))
```

Listing 2: Syntax file for LET