CS276A Final S20

Name: Ethan Brinkman

Pledge: I pledge my honor as a person, that during this final, I have had no assistance from anyone, nor have I seen the work of anyone else.

Signed Ethan Brinkman

Check here \_\_\_ if you have deliberately not signed the pledge.

You must turn this in to me by 11am CDT Wednesday, May 20, 2020.

Please type your work into a Word document and attach the document to an email to allen@stolaf.edu

1. Extend the basic interpreter (as given on the 4th page of the Friday, April 17, activity sheet from class) to include the following features: minus, +, \*, /(interger divide), greater?, less?, equal?, car, cdr, cons, emptylist, null?, procedure definitions of several arguments, procedure calls of several arguments. All of these have been defined and explained in class and in the class activity sheets. You are to:

a. Send me a running interpreter with all of these features working.

b.You must send me a drracket file attached to an email to allen.

c. In addition, you must put a copy of that file in your repo.

d. You must evaluate the following BNF expressions in your interpreter and send me a copy of these evaluations attached to an email to allen.

**NOTE:**

To avoid filling this document with a large amount of code before #2, I will add the interpreter code at the end of the document.

7

x

if zero? (-(v,5)) then equal?(i,1) else x

let y = x in \*(x, y)

if null?(cons(5,emptylist)) then +(x, 2) else 2

if less?(x, v) then i else car(cons(emptylist, emptylist))

let x = 5 in let y = -(x,3) in ( proc(x,z) /(x,+(y,z)) 18 7)

(proc(x,y) (x y) proc(a) +(a, 7) 8)

(run "7")

;(num-val 7)

(run "if zero? (-(v,5)) then equal?(i,1) else x")

;(bool-val #t)

(run "x")

;(num-val 10)

(run "let y = x in \*(x, y)")

;(num-val 100)

(run "if null?(cons(5,emptylist)) then +(x, 2) else 2")

;(num-val 2)

(run "if less?(x, v) then i else car(cons(emptylist, emptylist))")

;(emptylist-val)

(run "let x = 5 in let y = -(x,3) in ( proc(x,z) /(x,+(y,z)) 18 7)")

;(num-val 2)

(run "(proc(x,y) (x y) proc(a) +(a, 7) 8)")

;(num-val 15)

2a. Provide the semantics of

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

The interpreter sees a var-exp, it looks up the var’s identifier in the environment to get a value

2b. Provide the semantics of

(let-exp (var exp1 body)

(let ((val1 (value-of exp1 env)))

(value-of body (extend-env var val1 env))))

The interpreter sees a let-exp, it creates a block with a local variable and associated value

3. Trace the evaluation of the following two BNF let language expressions:

a. if null?(cons(v,emptylist)) then 2 else +(x, 2)   
**(NOTE: My traces are MUCH better**-**formatted in the DrRacket file I sent you)**

(run "if null?(cons(v,emptylist)) then 2 else +(x, 2)")

(value-of-program (scan&parse "if null?(cons(v,emptylist)) then 2 else +(x, 2)"))

(value-of-program (a-program (if-exp (null?-exp (cons-exp (var-exp 'v) (emptylist-exp))) (const-exp 2) (add-exp (var-exp 'x) (const-exp 2)))))

(value-of (if-exp (null?-exp (cons-exp (var-exp 'v) (emptylist-exp))) (const-exp 2) (add-exp (var-exp 'x) (const-exp 2)))(init-env))

val1 -> (vo exp1 env) =

(vo (null?-exp (cons-exp (var-exp v) (emptylist-exp))) ie)

val11 -> (vo exp11 ie) =

(vo (cons-exp (var-exp v) (emptylist-exp)) ie)

val111 -> (vo exp1111 ie) =

(vo (var-exp v) ie) =

(apply-env ie v) =

(num-val 5)

val112 -> (vo exp1112 ie) =

(vo (emptylist-exp) ie) =

(emptylist-val)

(cons-val (num-val 5) (emptylist-val))

(if (null? (cons-val (num-val 5) (emptylist-val)))(bool-val #t)(bool-val #f) =

(bool-val #f)

(if (expval->bool val1) (vo (const-exp 2) ie) (vo (add-exp (var-exp x) (const-exp 2)ie)) =

(if (expval->bool (bool-val #f)) (vo (const-exp 2) ie) (vo (add-exp (var-exp x) (const-exp 2)ie)) =

(if #f (vo (const-exp 2) ie) (vo (add-exp (var-exp x) (const-exp 2)ie)) =

(vo (add-exp (var-exp x) (const-exp 2)ie))

val12 -> (vo (var-exp x) ie)

(apply-env ie x)

(num-val 10)

val13 -> (vo (const-exp 2) ie)

(num-val 2)

num12 -> (expval->num val12) =

(expval->num (num-val 10)) =

10

num13 -> (expval->num val13) =

(expval->num (num-val 2)) =

2

(num-val (+ num1 num2)) =

(num-val (+ 10 3)) =

(num-val 12)

b. let v = 9 in (proc(x,y) cons(x,y) car(cons(v,emptylist)) emptylist)

(run "let v = 9 in (proc(x,y) cons(x,y) car(cons(v,emptylist)) emptylist)")

(value-of-program (scan&parse "let v = 9 in (proc(x,y) cons(x,y) car(cons(v,emptylist)) emptylist)"))

(value-of-program (a-program (let-exp'v (const-exp 9)(call-exp (proc-exp '(x y) (cons-exp (var-exp 'x) (var-exp 'y)))(list (car-exp (cons-exp (var-exp 'v) (emptylist-exp))) (emptylist-exp))))))

(value-of (let-exp'v (const-exp 9)(call-exp (proc-exp '(x y) (cons-exp (var-exp 'x) (var-exp 'y)))(list (car-exp (cons-exp (var-exp 'v) (emptylist-exp))) (emptylist-exp)))) (init-env))

val1-> (vo exp1 ie) =

(vo (const-exp 9) ie) =

(num-val 9)

(vo body (extend-env var val1 ie)) =

(vo body (extend-env v 9 ie)) =

(vo (call-exp (proc-exp (x y) (cons-exp (var-exp x) (var-exp y)))(list (car-exp (cons-exp (var-exp v) (emptylist-exp))) (emptylist-exp))) (extend-env v 9 ie)) =

proc -> (expval->proc (value-of rator (extend-env var val1 ie))) =

(expval->proc (value-of (proc-exp (x y) (cons-exp (var-exp x) (var-exp y)) (extend-env v 9 ie))) =

(expval->proc (proc-val (procedure (x y) (cons-exp (var-exp x) (var-exp y)) (extend-env v 9 ie))))

(procedure (x y) (cons-exp (var-exp x) (var-exp y)) (extend-env var val1 ie))

args -> (map (lambda (rand) (value-of rand (extend-env v 9 ie))) rands)) =

(map (lambda (rand) (value-of rand (extend-env v 9 ie))) (car(cons(v,emptylist)) (emptylist))) =

((value-of (car(cons(v,emptylist)) (extend-env v 9 ie)) (value-of (emptylist) (extend-env v 9 ie))) =

((num-val 9)(emptylist-val))

(apply-procedure proc args) =

(apply-procedure (procedure (x y) (cons-exp (var-exp x) (var-exp y)) (extend-env var val1 ie)) ((num-val 9)(emptylist-val))) =

(value-of (cons-exp (var-exp x) (var-exp y)) (extend-env\* (x y) ((num-val 9) (emptylist-val)) (extend-env v 9 ie))) =

(value-of (cons-exp (var-exp x) (var-exp y)) (extend-env y (emptylist-val) (extend-env x (num-val 9) (extend-env v 9 ie)))) =

(cons-val (num-val 9) (emptylist-val))

**CODE FOR #1:**

(define-datatype expression expression?

(const-exp(num number?))

(var-exp(var symbol?))

(zero?-exp(exp1 expression?))

(if-exp

(exp1 expression?)

(exp2 expression?)

(exp3 expression?))

(let-exp

(var symbol?)

(exp1 expression?)

(body expression?))

(minus-exp

(exp1 expression?))

(diff-exp

(exp1 expression?)

(exp2 expression?))

(add-exp

(exp1 expression?)

(exp2 expression?))

(mul-exp

(exp1 expression?)

(exp2 expression?))

(div-exp

(exp1 expression?)

(exp2 expression?))

(equal?-exp

(exp1 expression?)

(exp2 expression?))

(greater?-exp

(exp1 expression?)

(exp2 expression?))

(less?-exp

(exp1 expression?)

(exp2 expression?))

(cons-exp

(exp1 expression?)

(exp2 expression?))

(car-exp

(exp1 expression?))

(cdr-exp

(exp1 expression?))

(null?-exp

(exp1 expression?))

(emptylist-exp)

(proc-exp

(vars (list-of symbol?))

(body expression?))

(call-exp

(rator expression?)

(rands (list-of expression?)))

)

(define value-of

(lambda (exp env)

(cases expression exp

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

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

(zero?-exp (exp1)

(let ((val1 (value-of exp1 env)))

(let ((num1 (expval->num val1)))

(if (zero? num1)

(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))))

(minus-exp (exp1)

(let ((val1 (value-of exp1 env)))

(let ((num1 (expval->num val1)))

(num-val (- num1)))))

(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)))))

(add-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)))))

(mul-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)))))

(div-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 (quotient num1 num2)))))

(equal?-exp (exp1 exp2)

(let ((val1 (value-of exp1 env))

(val2 (value-of exp2 env)))

(let ((num1 (expval->num val1))

(num2 (expval->num val2)))

(bool-val (= num1 num2)))))

(greater?-exp (exp1 exp2)

(let ((val1 (value-of exp1 env))

(val2 (value-of exp2 env)))

(let ((num1 (expval->num val1))

(num2 (expval->num val2)))

(bool-val (> num1 num2)))))

(less?-exp (exp1 exp2)

(let ((val1 (value-of exp1 env))

(val2 (value-of exp2 env)))

(let ((num1 (expval->num val1))

(num2 (expval->num val2)))

(bool-val (< num1 num2)))))

(cons-exp (exp1 exp2)

(let ((val1 (value-of exp1 env))

(val2 (value-of exp2 env)))

(cons-val val1 val2)))

(car-exp (body)

(expval->car (value-of body env)))

(cdr-exp (body)

(expval->cdr (value-of body env)))

(null?-exp (exp1)

(let ((val1 (value-of exp1 env)))

(let ((bool1 (expval->emptylist? val1)))

(bool-val bool1))))

(emptylist-exp ()

(emptylist-val))

(proc-exp (vars body)

(proc-val (procedure vars body env)))

(call-exp (rator rands)

(let ((proc (expval->proc (value-of rator env)))

(args (map (lambda (rand) (value-of rand env)) rands)))

(apply-procedure proc args)))

)))

(define the-grammar

'((program (expression) a-program)

(expression (number) const-exp)

(expression

("zero?" "(" expression ")")

zero?-exp)

(expression

("if" expression "then" expression "else" expression)

if-exp)

(expression (identifier) var-exp)

(expression

("let" identifier "=" expression "in" expression)

let-exp)

(expression

("minus" "(" expression ")") minus-exp)

(expression

("-" "(" expression "," expression ")") diff-exp)

(expression

("+" "(" expression "," expression ")") add-exp)

(expression

("\*" "(" expression "," expression ")") mul-exp)

(expression

("/" "(" expression "," expression ")") div-exp)

(expression

("equal?" "(" expression "," expression ")") equal?-exp)

(expression

("greater?" "(" expression "," expression ")") greater?-exp)

(expression

("less?" "(" expression "," expression ")") less?-exp)

(expression

("cons" "(" expression "," expression ")") cons-exp)

(expression

("car" "(" expression ")") car-exp)

(expression

("cdr" "(" expression ")") cdr-exp)

(expression

("null?" "(" expression ")") null?-exp)

(expression

("emptylist") emptylist-exp)

(expression

("proc" "(" (separated-list identifier ",") ")" expression) proc-exp)

(expression

("(" expression (arbno expression) ")") call-exp)

))

(define-datatype expval expval?

(num-val

(value number?))

(bool-val

(bool boolean?))

(cons-val

(first expval?) (rest expval?))

(proc-val

(proc proc?))

(emptylist-val))

(define-datatype proc proc?

(procedure

(vars (list-of symbol?))

(body expression?)

(saved-env environment?)))

(define extend-env\*

(lambda (vars vals env)

(cond

((null? vars) env)

(else (extend-env\* (cdr vars) (cdr vals) (extend-env (car vars) (car vals) env))))))

(define-datatype environment environment?

(empty-env)

(extend-env

(saved-var var?)

(saved-val scheme-value?)

(saved-env environment?)))

(define apply-procedure

(lambda (proc1 vals)

(cases proc proc1

(procedure (vars body saved-env)

(value-of body (extend-env\* vars vals saved-env))))))

(define expval->car

(lambda (val)

(cases expval val

(cons-val (car cdr) car)

(else (expval-extractor-error 'cons val)))))

(define expval->cdr

(lambda (val)

(cases expval val

(cons-val (car cdr) cdr)

(else (expval-extractor-error 'cons val)))))

(define expval->emptylist?

(lambda (val)

(cases expval val

(emptylist-val () #t)

(cons-val (first rest) #f)

(else (expval-extractor-error 'cons-or-emptylist val)))))

(define expval->num

(lambda (val)

(cases expval val

(num-val (num) num)

(else (expval-extractor-error 'num val)))))

(define expval->bool

(lambda (val)

(cases expval val

(bool-val (bool) bool)

(else (expval-extractor-error 'bool val)))))

(define expval->proc

(lambda (val)

(cases expval val

(proc-val (proc) proc)

(else (expval-extractor-error 'proc val)))))

(define var? symbol?)

(define scheme-value? (lambda (s) #t))

(define apply-env

(lambda (env search-var)

(cases environment env

(empty-env ()

(report-no-binding-found search-var))

(extend-env (saved-var saved-val saved-env)

(if (eqv? search-var saved-var)

saved-val

(apply-env saved-env search-var))))))

(define init-env

(lambda ()

(extend-env

'i (num-val 1)

(extend-env

'v (num-val 5)

(extend-env

'x (num-val 10)

(empty-env))))))

(define expval-extractor-error

(lambda (variant value)

(eopl:error 'expval-extractors "Looking for a ~s, found ~s"

variant value)))

(define value-of-program

(lambda (pgm)

(cases program pgm

(a-program (exp1)

(value-of exp1 (init-env))))))

(define-datatype program program?

(a-program(expl1 expression?)))

(define report-no-binding-found

(lambda (search-var)

(eopl:error 'apply-env "No binding for ~s" search-var)))

(define run

(lambda (string)

(value-of-program (scan&parse string))))

(define the-lexical-spec

'((whitespace (whitespace) skip)

(comment ("%" (arbno (not #\newline))) skip)

(identifier

(letter (arbno (or letter digit "\_" "-" "?")))

symbol)

(number (digit (arbno digit)) number)

(number ("-" digit (arbno digit)) number)

))

(define scan&parse

(sllgen:make-string-parser the-lexical-spec the-grammar))