Tony Maldonado
CS 471
November 18, 2020

Program 9 – Lisp Programming: Circuit Descriptors

Problem Description:

In this program, we were to write multiple functions in Lisp that will help with evaluate circuit descriptors representing computer logic. One function was to count the number of times a logical operator appears in any inputted CD (logical) expression. Another function was to uniquely list all the variables that were used in any CD expression. Lastly, we were to write a function which would reduce any inputter CD to a simpler form by using tautologies. Essentially, to reduce a logical expression.

Lisp was a little tricky to get the upper hand on considering we've never had to use the Lisp language in any other CS class. However, after some experimenting and work, I was able to successfully implement the functions. Of course, with much help from our lectures.

Code:

```
;; Name: Tony Maldonado
;; Date: November 18, 2020
;; Input: A logical expression
;; Output: A reduced CD form, the number of specified operators or
;;        a list of all unique variables
;; Precondition: The user gives a valid CD as input
;; Postcondition: The program will give the correct output based
;;               on the function


;; This program evaluates a circuit design

;; Counts the number of specified operators in the argument
;; Format is: 'operator '(operator list)
(define (count_operator x oplist)
   (cond ((null? oplist) 0)
        ((not (list? oplist))
        (if (eq? x oplist) 1 0))
        (else (+ (count_operator x (car oplist)) (count_operator x (cdr oplist))))
   ) ;; end cond
) ;; end define

;; Lists all the unique variables in the argument
;; Format is: '(operator list)
(define (unique oplist)
   (cond ((null? oplist) '() )
        ((not (list? oplist)) '() )
        ((member (car oplist) (cdr oplist)) (unique (cdr oplist)))
        (else (cons (car oplist) (unique (cdr oplist))))
    ) ;; end cond
) ;; end define

;; Clean up all the other stuff in a CD
;; Assume is we get a flattened, unique CD
;; Format is: (findinputvars (unique (flatten '(oplist))))
(define (findinputvars oplist)
    (cond ((null? oplist) '() )
        ((not (list? oplist)) '() )
        ((or (eq? (car oplist) 1)
            (eq? (car oplist) 0)
            (eq? (car oplist) 'AND)
            (eq? (car oplist) 'NOT)
            (eq? (car oplist) 'OR))
            (findinputvars (cdr oplist)))
        (else (cons (car oplist) (findinputvars (cdr oplist))))
    ) ;; end cond
) ;; end define

;; NOT CD1
(define (evalcd CD)
    ;; Base Case
    (cond ((null? CD) '())
        ;; True, False, or A1....A1000
        ((not (list? CD)) CD)
        ((eq? (car CD) 'NOT) (evalcd_not CD))
        ((eq? (car CD) 'AND) (evalcd_and CD))
        ((eq? (car CD) 'OR)  (evalcd_or CD))
    ) ;; end cond
) ;; end define

;; PRE: MUST be a (NOT CD) form (CAR CD) => NOT
;; POST: Reduce the Argument AND see if we can reduce it
;; NOT
(define (evalcd_not CD)
    (cond ((eq? (evalcd (cadr CD)) 0) 1)
        ((eq? (evalcd (cadr CD)) 1) 0)
        (else (cons 'NOT (list (evalcd (cadr CD)))))
    ) ;; end cond
) ;; end define
```

Tony Maldonado
CS 471
November 18, 2020

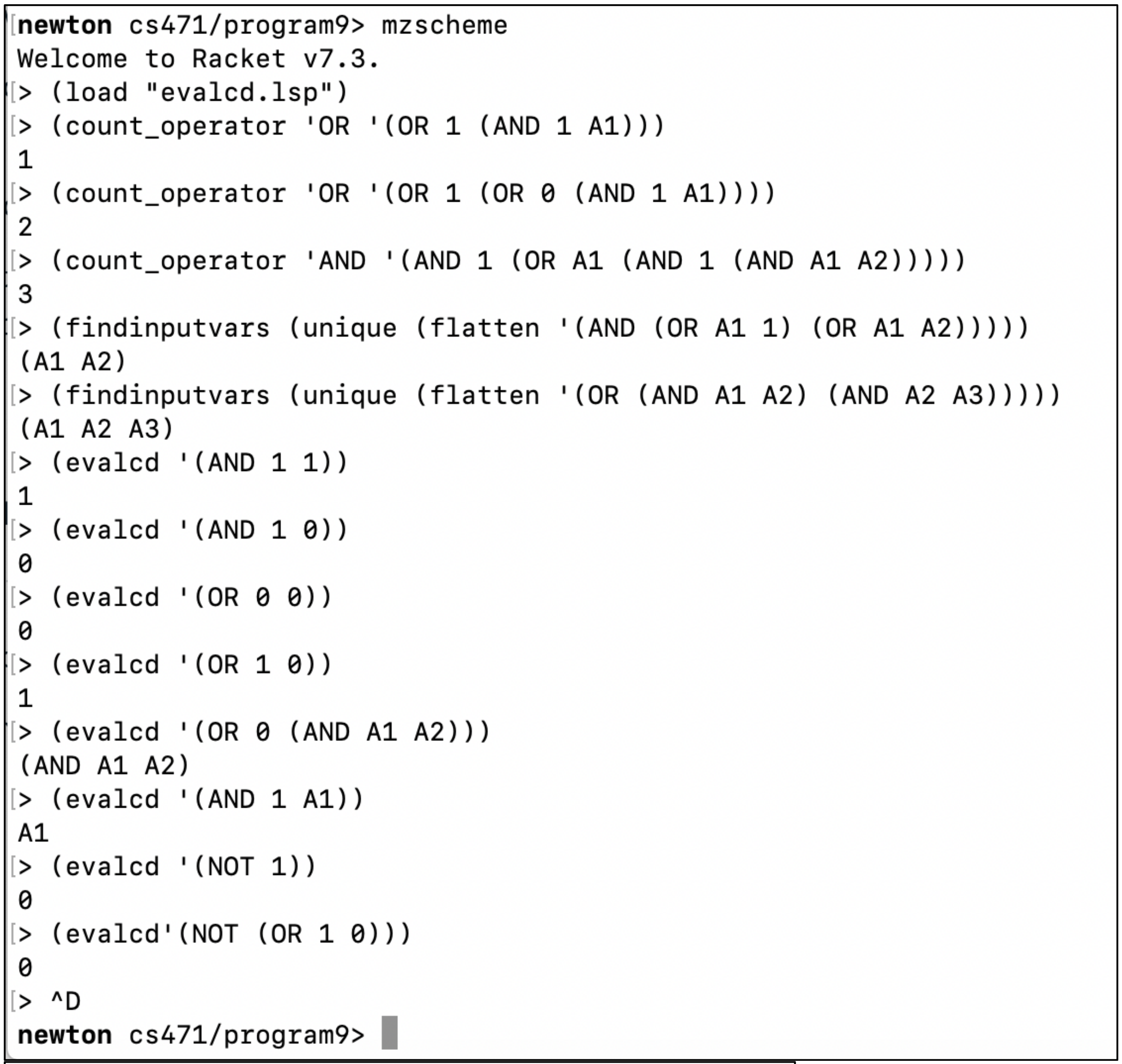
```
;; PRE: MUST be (AND CD1 CD2) format
;; POST: Apply simple tautologies to the CD1 and CD2 and maybe reduce
;; AND
(define (evalcd_and CD)
   (cond ((eq? (evalcd (cadr CD)) 0) 0)
        ((eq? (evalcd (caddr CD)) 0) 0)
        ((eq? (evalcd (cadr CD)) 1) (evalcd (caddr CD)))
        ((eq? (evalcd (caddr CD)) 1) (evalcd (cadr CD)))
        (else (cons 'AND (list (evalcd (cadr CD)) (evalcd (caddr CD)))))
   ) ;; end cond
) ;; end define

;; PRE: MUST be a (OR CD1 CD2) format
;; POST: Apply simple tautologies to the CD1 and CD2 and maybe reduce
;; OR
(define (evalcd_or CD)
   (cond ((eq? (evalcd (cadr CD)) 1) 1)
        ((eq? (evalcd (caddr CD)) 1) 1)
        ((eq? (evalcd (cadr CD)) 0) (evalcd (caddr CD)))
        ((eq? (evalcd (caddr CD)) 0) (evalcd (cadr CD)))
        (else (cons 'OR (list (evalcd (cadr CD)) (evalcd (caddr CD)))))
   ) ;; end cond
) ;; end define
```

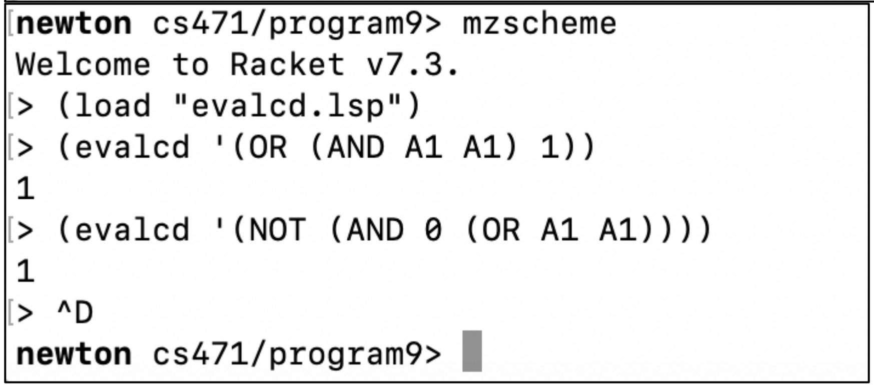
Tony Maldonado
CS 471
November 18, 2020

Output/ Screenshots:

- Here are a couple of screenshots showing the 'count_operator', 'unique', and reduce functions in action



```
newton cs471/program9> mzscheme
Welcome to Racket v7.3.
> (load "evalcd.lsp")
> (count_operator 'OR '(OR 1 (AND 1 A1)))
1
> (count_operator 'OR '(OR 1 (OR 0 (AND 1 A1))))
2
> (count_operator 'AND '(AND 1 (OR A1 (AND 1 (AND A1 A2)))))
3
> (findinputvars (unique (flatten '(AND (OR A1 1) (OR A1 A2)))))
(A1 A2)
> (findinputvars (unique (flatten '(OR (AND A1 A2) (AND A2 A3)))))
(A1 A2 A3)
> (evalcd '(AND 1 1))
1
> (evalcd '(AND 1 0))
0
> (evalcd '(OR 0 0))
0
> (evalcd '(OR 1 0))
1
> (evalcd '(OR 0 (AND A1 A2)))
(AND A1 A2)
> (evalcd '(AND 1 A1))
A1
> (evalcd '(NOT 1))
0
> (evalcd'(NOT (OR 1 0)))
0
> ^D
newton cs471/program9>
```



```
newton cs471/program9> mzscheme
Welcome to Racket v7.3.
> (load "evalcd.lsp")
> (evalcd '(OR (AND A1 A1) 1))
1
> (evalcd '(NOT (AND 0 (OR A1 A1))))
1
> ^D
newton cs471/program9>
```