Jeffrey Lansford

11/18/2020

Program 9

In this lab we are learning how to write Lisp by creating three functions to work with Circuit Design. 1) count how many times an operator is used in the CD (Circuit Design). 2) list all unique input variables. 3) reduce the CD using tautologies. Here is our defined Circuit:

C -> (AND C C) | (OR C C)

| (NOT C)

| A[1-1000]

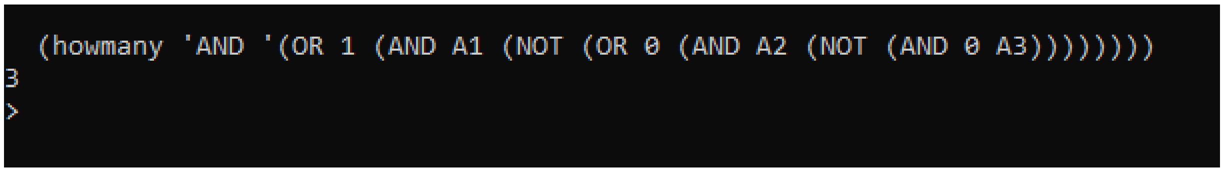
| 1

| 0

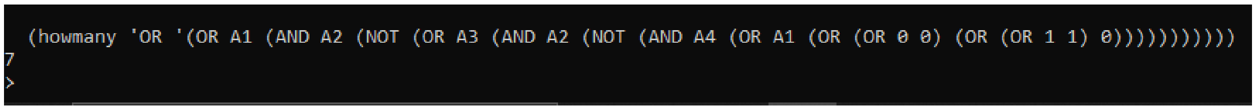
```
;; Jeffrey Lansford
;; LISP Program
;; 11/18/2020
;; Lisp program that can count how many operators or symbols occurs in CD,
;; can list the input varibles in a CD,
;; and reduce CD using tautologies

;; (require trace)

;; 1) Counts how many opertors in CD
;; PRE: L must be a vaild CD
(define (howmany x L )
        ;; base condition if null list, retrun 0
        (cond ((null? L) 0)
                ;; check L is list
                ((not (list? L))
                    ;; if atom x is equal to atom L, then return 1, if not, retur
n 0
                    (if (eq? x L) 1 0))
                ;; L is a list, do recursion on the head of the list and tail of
the list
                (else (+ (howmany x (car L))
                         (howmany x (cdr L))))))
```

```
;; 2) Main function to find unique varibles using the two helper functions uniq a
nd findinputvars
;; PRE: input must be a valid CD
(define (finduniqueinputvars L)
    ;; flatten list to be one big list, find uniq symbols and operators, and find
 input varibles
    (findinputvars (uniq (flatten L))))

;; Helper function to get uniq operators and symbols/varibles (0, 1, A1, A2, ...
A1000)
(define (uniq L)
    ;; base case, if we get a empty list, return a empty list
    (cond ((null? L ) '() )
        ;; if an atom, then return empty list
        ((not (list? L )) '() )
        ;; if head of list is in rest of the list, the recursion on the tail of t
he list (rest of the list)
        ((member (car L) (cdr L)) (uniq (cdr L)))
        ;; head of the list was not in rest of the list, the construct list with
head of list and recursion on tail of list
        (else ( cons (car L) (uniq (cdr L))))))

;; Helper function that finds input varibles (A1, A2, .. A1000)
(define (findinputvars L)
        ;; base case, if we get a empty list, return a empty list
        (cond ((null? L) '())
        ;; if L is not a list, the return empty list
        ((not (list? L)) '())
        ;; if head of list L is not a input varible, then do recursion on tail of
 list
        ((or (eq? (car L) 1)
             (eq? (car L) 0)
             (eq? (car L) 'AND)
             (eq? (car L) 'OR)
             (eq? (car L) 'NOT))
            (findinputvars (cdr L)))
        ;; head of list is a input varible, then construct a new list with head o
f list L and d orecursion on tail of list L
        (else (cons (car L) (findinputvars (cdr L))))))



;; 3) Main Function to reduce CD using tautologies
;; PRE: input must be a vaild CD
(define (evalcd CD)
```

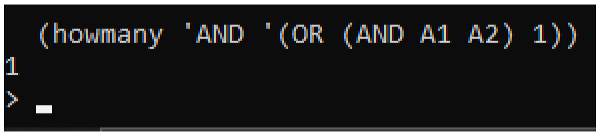
```scheme
    ;; if CD is a empty list, then return empty list
    (cond ((null? CD) '())
        ;; if CD is an atom , then return atom
        ((not (list? CD )) CD)
        ;; if head of list CD is a NOT operator, run function to reduce NOT
        ((eq? (car CD) 'NOT) (evalcd_not CD))
        ;; if head of list CD is a AND operator, run function to reduce AND
        ((eq? (car CD) 'AND) (evalcd_and CD))
        ;; if head of list CD is a OR operator, run function to reduce OR
        ((eq? (car CD) 'OR) (evalcd_or CD))
        ))

;; Helper function to reduce NOT
(define (evalcd_not CD)
    ;; if second element of list CD is a 0, then reduce to 1
    (cond ((eq? (evalcd (cadr CD)) 0) 1)
        ;; if second element of list CD is a 1, then reduce to 0
        ((eq? (evalcd (cadr CD)) 1) 0)
        ;; make a new list with NOT operator and evaluate CD on the second elemen
t of the list
        (else (cons 'NOT (list (evalcd (cadr CD)))))))

;; Helper function to reduce AND
(define (evalcd_and CD)
    ;; if second element is a 0, then reduce to 0
    (cond ((eq? (evalcd (cadr CD)) 0) 0)
        ;; if third element is a 0, then reduce to 0
        ((eq? (evalcd (caddr CD)) 0) 0)
        ;; if second element is a 1, then evaluate CD on third element
        ((eq? (evalcd (cadr CD)) 1) (evalcd (caddr CD)))
        ;; if third element is a 1, then evaluate CD on second element
        ((eq? (evalcd (caddr CD)) 1) (evalcd (cadr CD)))
        ;; make a new list with the AND operator and evaluate CD on both the seco
nd and third element
        (else (cons 'AND
            (list (evalcd (cadr CD))
            (evalcd (caddr CD)))))))

;; Helper function to reduce OR
(define (evalcd_or CD)
    ;; if second element is a 1, the nreduce to 1
    (cond ((eq? (evalcd (cadr CD)) 1) 1)
        ;; if third element is a 1, then reduce to 1
        ((eq? (evalcd (caddr CD)) 1) 1)
        ;; if second element is a 0, then evaluate on third element
        ((eq? (evalcd (cadr CD)) 0) (evalcd (caddr CD)))
        ;; if third element is a 0, then evaluate on second element
        ((eq? (evalcd (caddr CD)) 0) (evalcd (cadr CD)))
        ;; make a new list with the OR operator and evaluate on both the second a
nd third element
        (else (cons 'AND
            (list (evalcd (cadr CD))
            (evalcd (caddr CD)))))))
```



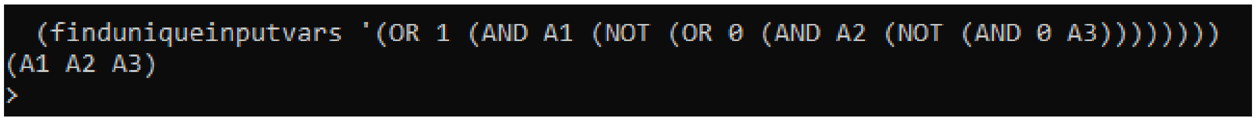
```
  (howmany 'AND '(OR 1 (AND A1 (NOT (OR 0 (AND A2 (NOT (AND 0 A3))))))))
3
>
```



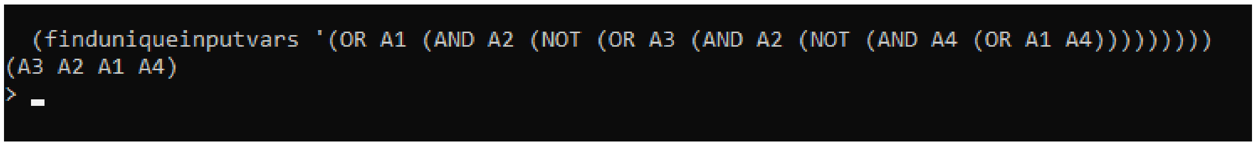
```
  (howmany 'OR '(OR A1 (AND A2 (NOT (OR A3 (AND A2 (NOT (AND A4 (OR A1 (OR (OR 0 0) (OR (OR 1 1) 0)))))))))))
7
>
```



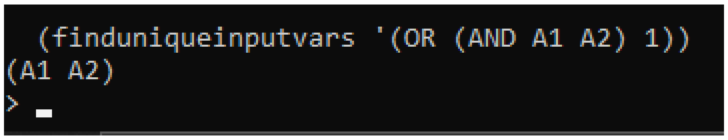
```
  (howmany 'AND '(OR (AND A1 A2) 1))
1
>
```



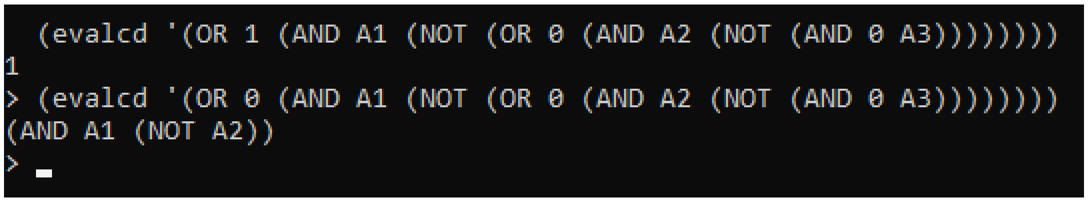
```
  (finduniqueinputvars '(OR 1 (AND A1 (NOT (OR 0 (AND A2 (NOT (AND 0 A3))))))))
(A1 A2 A3)
>
```



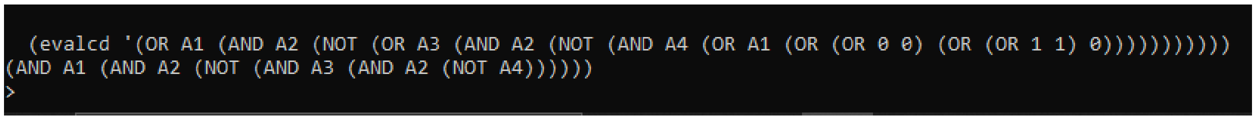
```
  (finduniqueinputvars '(OR A1 (AND A2 (NOT (OR A3 (AND A2 (NOT (AND A4 (OR A1 A4)))))))))
(A3 A2 A1 A4)
>
```



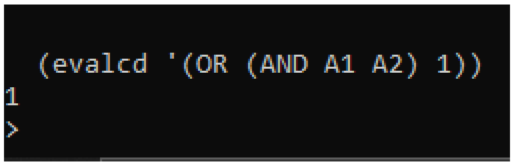
```
  (finduniqueinputvars '(OR (AND A1 A2) 1))
(A1 A2)
>
```



```
  (evalcd '(OR 1 (AND A1 (NOT (OR 0 (AND A2 (NOT (AND 0 A3))))))))
1
> (evalcd '(OR 0 (AND A1 (NOT (OR 0 (AND A2 (NOT (AND 0 A3))))))))
(AND A1 (NOT A2))
> 
```



```
  (evalcd '(OR A1 (AND A2 (NOT (OR A3 (AND A2 (NOT (AND A4 (OR A1 (OR (OR 0 0) (OR (OR 1 1) 0)))))))))))
(AND A1 (AND A2 (NOT (AND A3 (AND A2 (NOT A4))))))
>
```



```
  (evalcd '(OR (AND A1 A2) 1))
1
>
```