Xiana Lara
November 18, 2020
CS 471

Lisp Programming

Using lisp, this lab required us to create digital circuit designs. Given circuit descriptors we had to write routines for them. These routines included a function which counts the number of times a logical operator is used in a CD, a function which uniquely lists all of the input variables, and a function that given a CD, reduces, the CD to a simpler form by using tautologies.

Code:

```
;; Xiana Lara
;; 11 18 2020
;; Running a writing a lisp program
;; CODE APAPTED FROM SHAUN COOPER
;; input :
;; output :

;; evaluates a circuit design

;; 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))
    )
)
;; PRE:MUST be a (NOT CD) form (CAR CD) => NOT
;; reduce the Argument and see if we can reduce it
(define (evalcd_not CD)
    (cond ((eq? (evalcd (cadr CD)) 0) 1)
          ((eq? (evalcd (cadr CD)) 1) 0)
          (else (cons 'NOT (list (evalcd (cadr CD)))))
    )
)
;; PRE:MUST be (AND CD1 CD2) format
;; post apply simple tautologies to the CD1 and CD2 and may be reduced
;; AND
(define(evalcd_and CD)
    (cond ((eq? (evalcd (cadr CD)) 0) 0)
          ((eq? (evalcd (caddr CD)) 0) 0)
```

```scheme
          ((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))
                        )
                  )
          )
    )
)

;; PRE:Must be (OR CD1 CD2) format
;; post apply simple tautologies to the CD1 and CD2 and may be reduced
;; 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))
                        )
                  )
          )
    )
)

;; a function which uniquely lists all of the input VARIABLES
(define(unique lst)
    (cond
        ((null? lst) '())
        ((not (list? lst)) '())
        ((member(car lst)(cdr lst))(unique(cdr lst)))
        (else
            (cons
                (car lst)(unique(cdr lst))
            )
        )
    )
)

;; a function that given a CD, reduces, the CD
;; to a simpler form by using tautologies
```

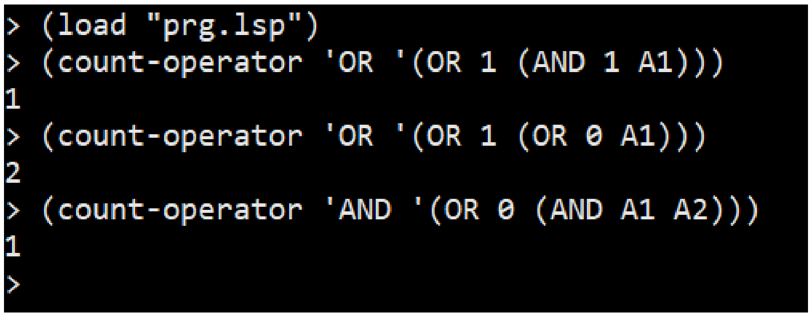
Xiana Lara
November 18, 2020
CS 471

```scheme
(define (findinvars lst)
    (cond
        ((null? lst) '())
        ((not(list? lst)) '())
        ((or (eq? (car lst) 1)
             (eq? (car lst) 0)
             (eq? (car lst) 'AND)
             (eq? (car lst) 'OR)
             (eq? (car lst) 'NOT)
             )
            (findinvars (cdr lst))
        )
        (else
            (cons (car lst) (findinvars (cdr lst)))))
)

;; a function which counts the number of times a logical
;; operator is used in a a CD
(define(count-operator x lst)
    (cond
        ((null? lst) 0)
        ((not (list? lst))
            (if(eq? x lst) 1 0)
        )
        (else
            (+
                (count-operator x (car lst))(count-operator x (cdr lst))
            )
        )
    )
)
```

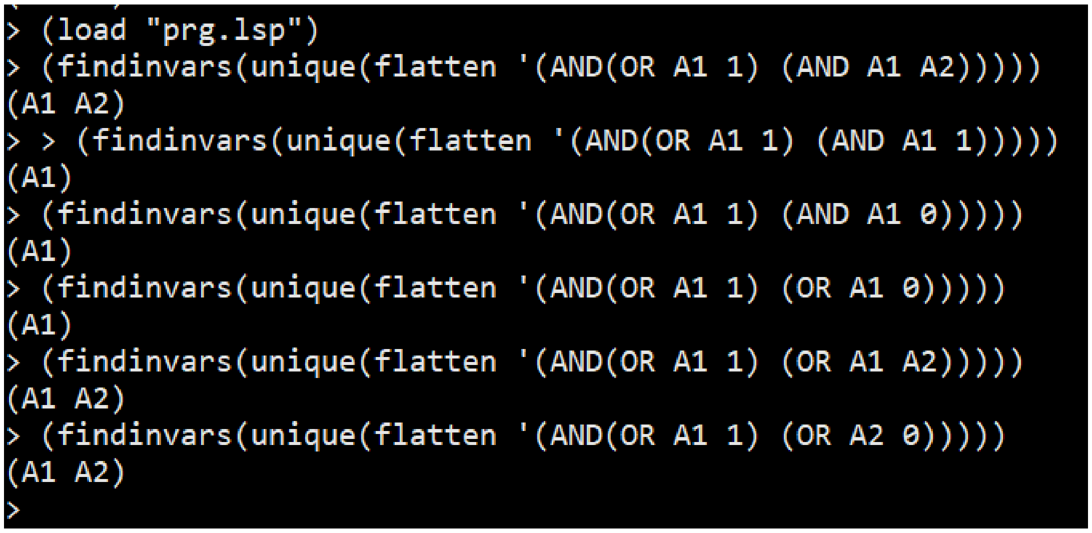
Xiana Lara
November 18, 2020
CS 471

Count-operator function Output:



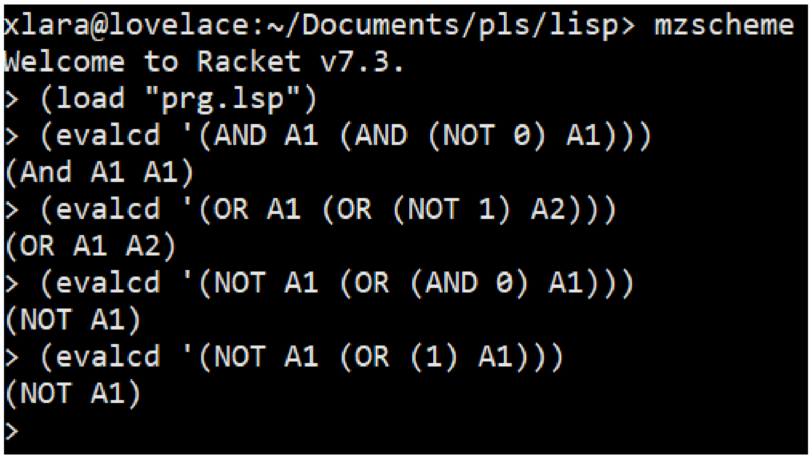
```
> (load "prg.lsp")
> (count-operator 'OR '(OR 1 (AND 1 A1)))
1
> (count-operator 'OR '(OR 1 (OR 0 A1)))
2
> (count-operator 'AND '(OR 0 (AND A1 A2)))
1
>
```

UNIQUE function Output:



```
> (load "prg.lsp")
> (findinvars(unique(flatten '(AND(OR A1 1) (AND A1 A2)))))
(A1 A2)
> > (findinvars(unique(flatten '(AND(OR A1 1) (AND A1 1)))))
(A1)
> (findinvars(unique(flatten '(AND(OR A1 1) (AND A1 0)))))
(A1)
> (findinvars(unique(flatten '(AND(OR A1 1) (OR A1 0)))))
(A1)
> (findinvars(unique(flatten '(AND(OR A1 1) (OR A1 A2)))))
(A1 A2)
> (findinvars(unique(flatten '(AND(OR A1 1) (OR A2 0)))))
(A1 A2)
>
```

Reduce Function Output:



```
xlara@lovelace:~/Documents/pls/lisp> mzscheme
Welcome to Racket v7.3.
> (load "prg.lsp")
> (evalcd '(AND A1 (AND (NOT 0) A1)))
(And A1 A1)
> (evalcd '(OR A1 (OR (NOT 1) A2)))
(OR A1 A2)
> (evalcd '(NOT A1 (OR (AND 0) A1)))
(NOT A1)
> (evalcd '(NOT A1 (OR (1) A1)))
(NOT A1)
>
```