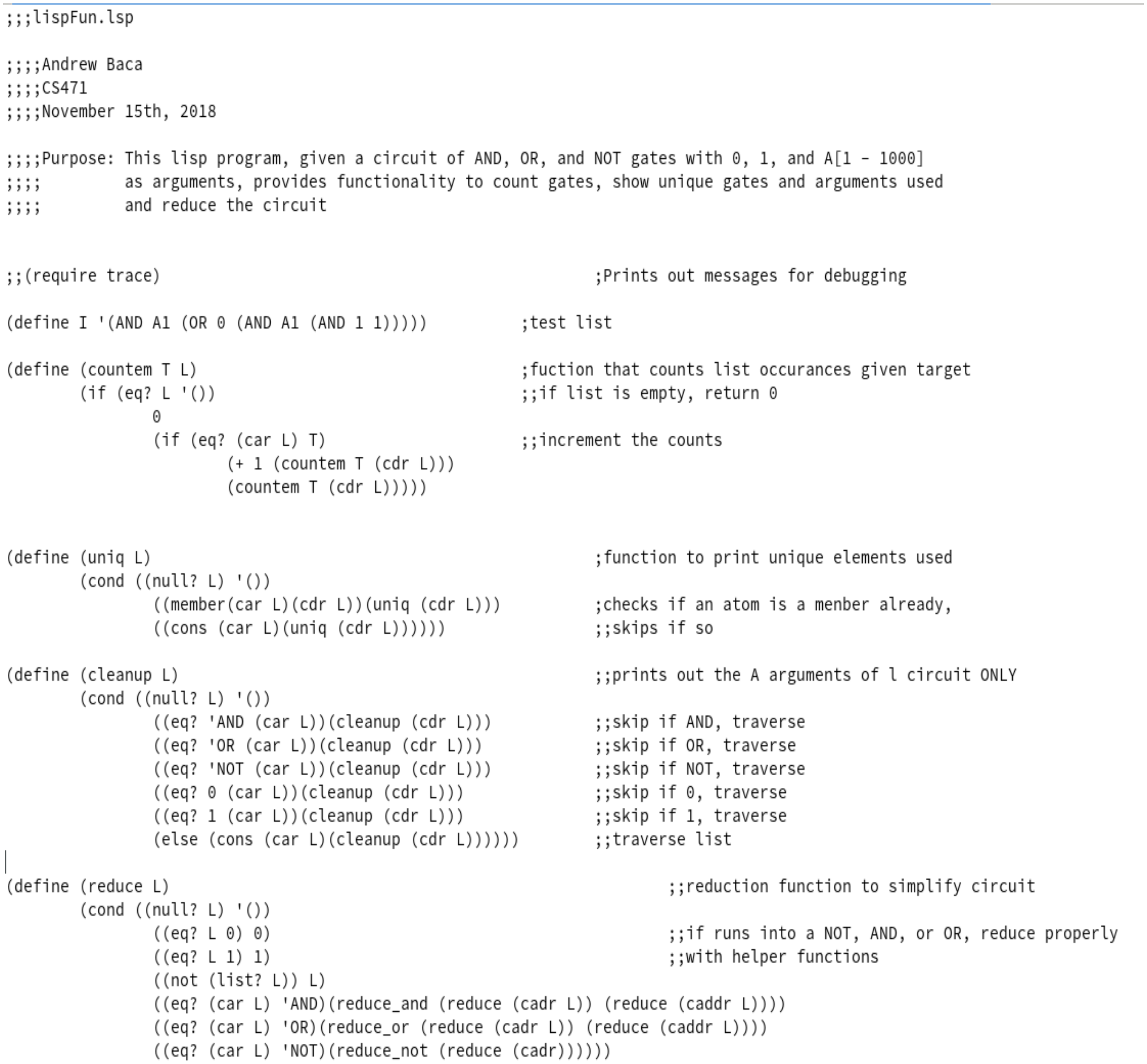
Andrew Baca
LISP Programming
CS 471
November 15th, 2018’

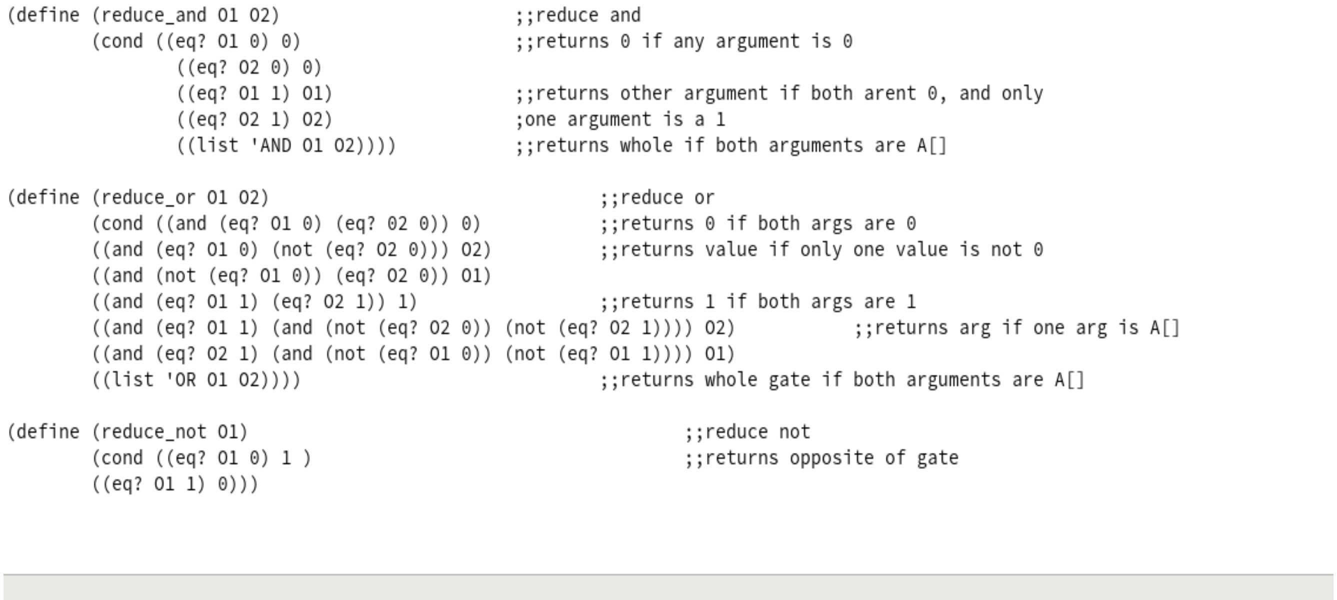
Purpose:

This program is meant to lest the LISP programming language using mzscheme. We will be making functions in lisp that given a circuit, we are to list how many occurrences there are in a circuit given a target, as well as list all the unique arguments and gates in a circuit and reduce circuits.

Code:



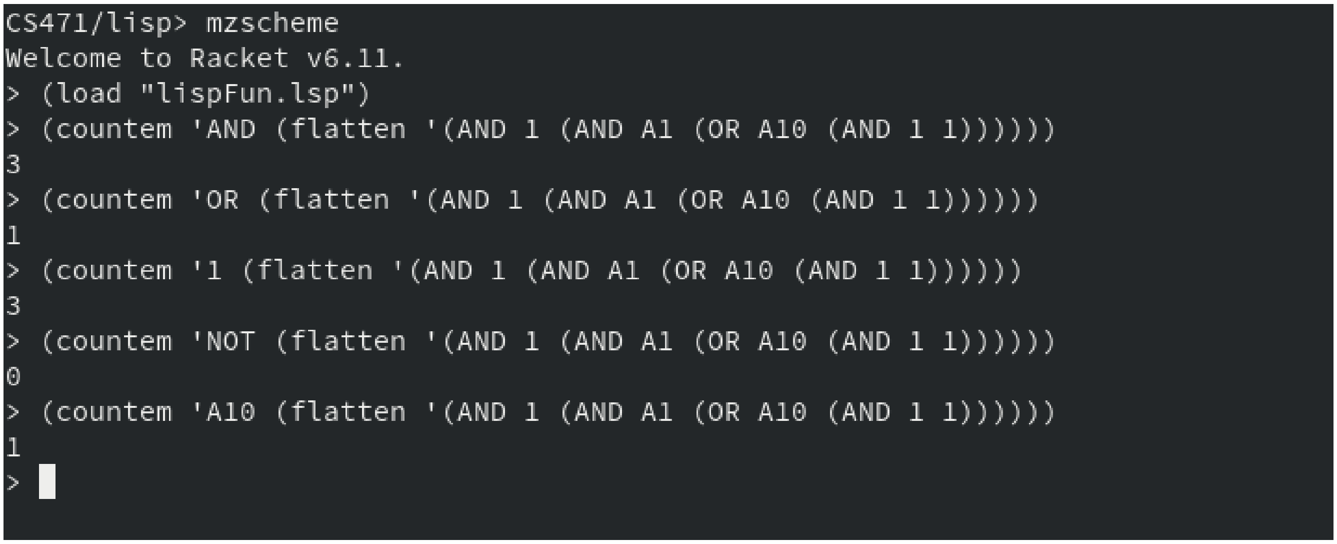
```
;;;lispFun.lsp

;;;;Andrew Baca
;;;;CS471
;;;;November 15th, 2018

;;;;Purpose: This lisp program, given a circuit of AND, OR, and NOT gates with 0, 1, and A[1 - 1000]
;;;;         as arguments, provides functionality to count gates, show unique gates and arguments used
;;;;         and reduce the circuit


;;(require trace)                                               ;Prints out messages for debugging

(define I '(AND A1 (OR 0 (AND A1 (AND 1 1)))))          ;test list

(define (countem T L)                                   ;fuction that counts list occurances given target
        (if (eq? L '())                                 ;;if list is empty, return 0
                0
                (if (eq? (car L) T)                     ;;increment the counts
                        (+ 1 (countem T (cdr L)))
                        (countem T (cdr L)))))


(define (uniq L)                                                ;function to print unique elements used
        (cond ((null? L) '())
                ((member(car L)(cdr L))(uniq (cdr L)))          ;checks if an atom is a menber already,
                ((cons (car L)(uniq (cdr L))))))                ;;skips if so

(define (cleanup L)                                             ;;prints out the A arguments of l circuit ONLY
        (cond ((null? L) '())
                ((eq? 'AND (car L))(cleanup (cdr L)))           ;;skip if AND, traverse
                ((eq? 'OR (car L))(cleanup (cdr L)))            ;;skip if OR, traverse
                ((eq? 'NOT (car L))(cleanup (cdr L)))           ;;skip if NOT, traverse
                ((eq? 0 (car L))(cleanup (cdr L)))              ;;skip if 0, traverse
                ((eq? 1 (car L))(cleanup (cdr L)))              ;;skip if 1, traverse
                (else (cons (car L)(cleanup (cdr L))))))        ;;traverse list

(define (reduce L)                                                      ;;reduction function to simplify circuit
        (cond ((null? L) '())
                ((eq? L 0) 0)                                           ;;if runs into a NOT, AND, or OR, reduce properly
                ((eq? L 1) 1)                                           ;;with helper functions
                ((not (list? L)) L)
                ((eq? (car L) 'AND)(reduce_and (reduce (cadr L)) (reduce (caddr L))))
                ((eq? (car L) 'OR)(reduce_or (reduce (cadr L)) (reduce (caddr L))))
                ((eq? (car L) 'NOT)(reduce_not (reduce (cadr)))))
```



```
(define (reduce_and O1 O2)                      ;;reduce and
        (cond ((eq? O1 0) 0)                    ;;returns 0 if any argument is 0
                ((eq? O2 0) 0)
                ((eq? O1 1) O1)                 ;;returns other argument if both arent 0, and only
                ((eq? O2 1) O2)                 ;one argument is a 1
                ((list 'AND O1 O2))))           ;;returns whole if both arguments are A[]

(define (reduce_or O1 O2)                               ;;reduce or
        (cond ((and (eq? O1 0) (eq? O2 0)) 0)           ;;returns 0 if both args are 0
        ((and (eq? O1 0) (not (eq? O2 0))) O2)          ;;returns value if only one value is not 0
        ((and (not (eq? O1 0)) (eq? O2 0)) O1)
        ((and (eq? O1 1) (eq? O2 1)) 1)                 ;;returns 1 if both args are 1
        ((and (eq? O1 1) (and (not (eq? O2 0)) (not (eq? O2 1)))) O2)          ;;returns arg if one arg is A[]
        ((and (eq? O2 1) (and (not (eq? O1 0)) (not (eq? O1 1)))) O1)
        ((list 'OR O1 O2))))                            ;;returns whole gate if both arguments are A[]

(define (reduce_not O1)                                         ;;reduce not
        (cond ((eq? O1 0) 1 )                                   ;;returns opposite of gate
        ((eq? O1 1) 0)))
```

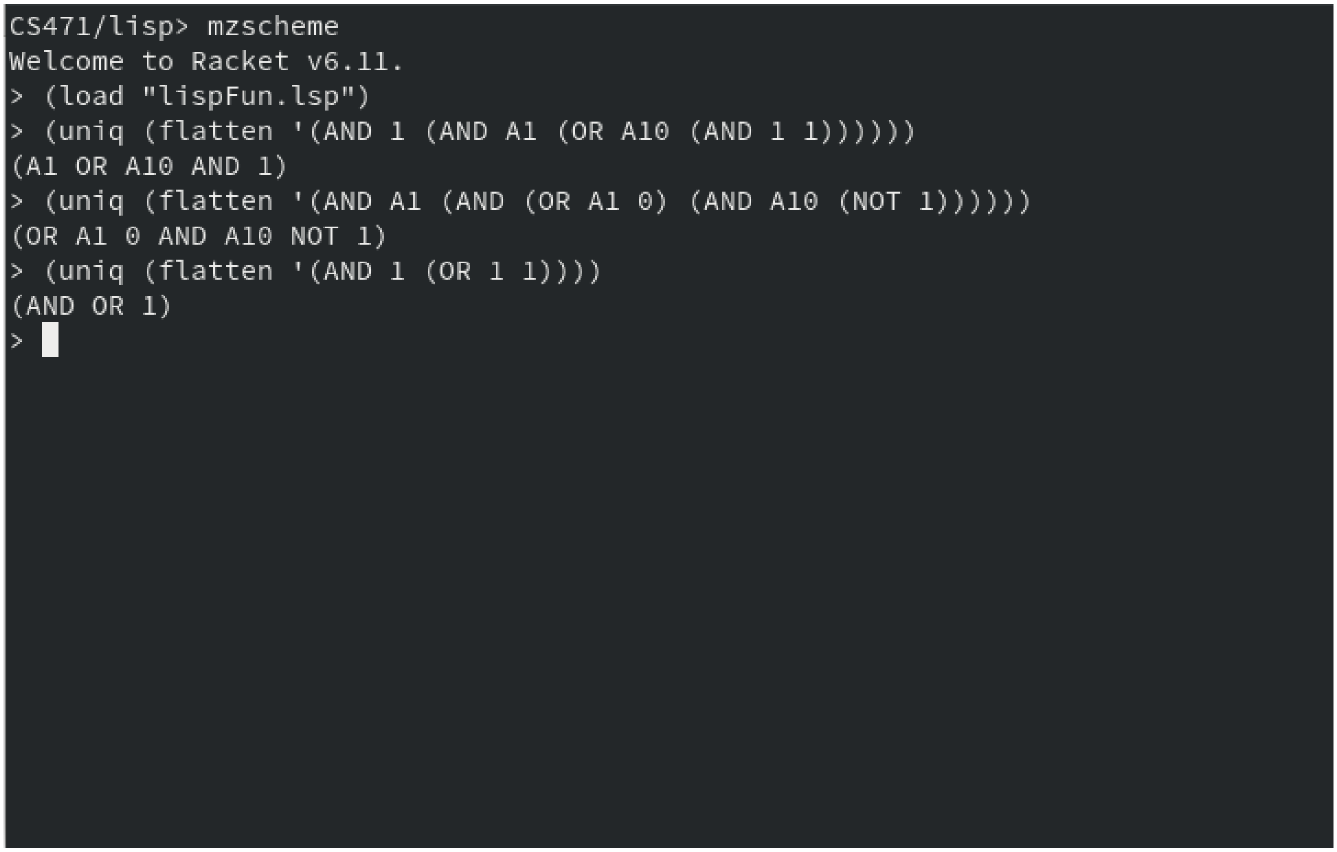
Output:

Countem:



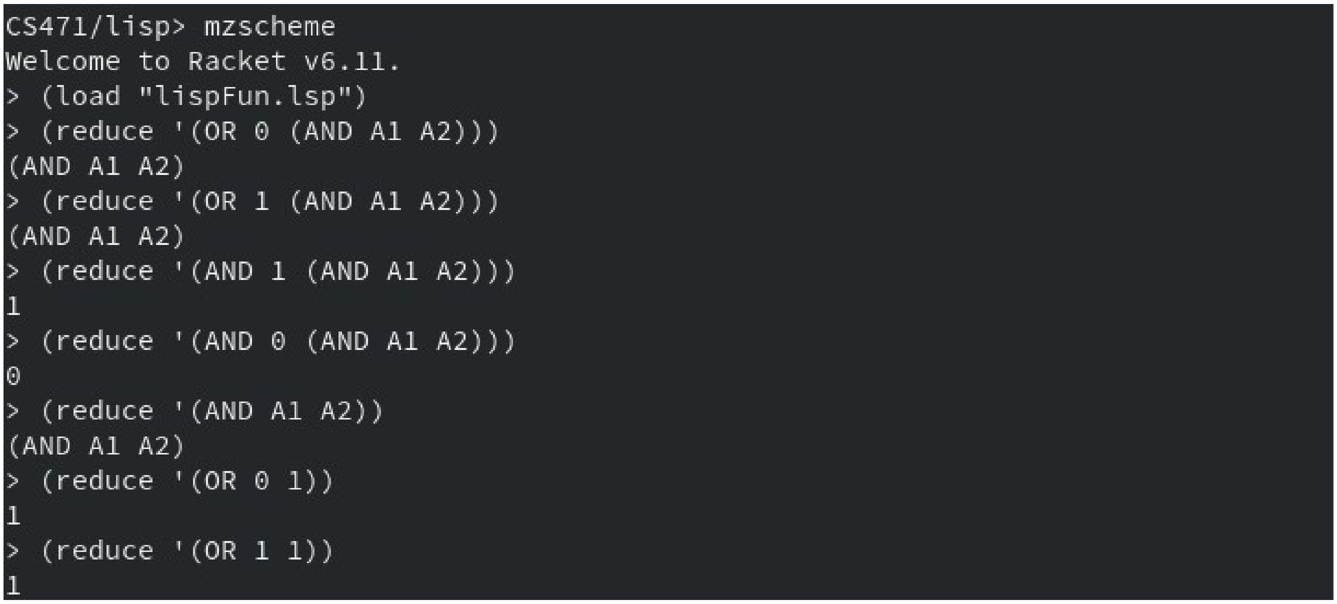
```
CS471/lisp> mzscheme
Welcome to Racket v6.11.
> (load "lispFun.lsp")
> (countem 'AND (flatten '(AND 1 (AND A1 (OR A10 (AND 1 1))))))
3
> (countem 'OR (flatten '(AND 1 (AND A1 (OR A10 (AND 1 1))))))
1
> (countem '1 (flatten '(AND 1 (AND A1 (OR A10 (AND 1 1))))))
3
> (countem 'NOT (flatten '(AND 1 (AND A1 (OR A10 (AND 1 1))))))
0
> (countem 'A10 (flatten '(AND 1 (AND A1 (OR A10 (AND 1 1))))))
1
> 
```

Uniq:



```
CS471/lisp> mzscheme
Welcome to Racket v6.11.
> (load "lispFun.lsp")
> (uniq (flatten '(AND 1 (AND A1 (OR A10 (AND 1 1))))))
(A1 OR A10 AND 1)
> (uniq (flatten '(AND A1 (AND (OR A1 0) (AND A10 (NOT 1))))))
(OR A1 0 AND A10 NOT 1)
> (uniq (flatten '(AND 1 (OR 1 1))))
(AND OR 1)
> 
```

reduce:



```
CS471/lisp> mzscheme
Welcome to Racket v6.11.
> (load "lispFun.lsp")
> (reduce '(OR 0 (AND A1 A2)))
(AND A1 A2)
> (reduce '(OR 1 (AND A1 A2)))
(AND A1 A2)
> (reduce '(AND 1 (AND A1 A2)))
1
> (reduce '(AND 0 (AND A1 A2)))
0
> (reduce '(AND A1 A2))
(AND A1 A2)
> (reduce '(OR 0 1))
1
> (reduce '(OR 1 1))
1
```