Collin Gros
11-18-2020
cs471

LISP

PROBLEM DESCRIPTION

Experiment with LISP and write 3 programs to complete the following objectives (given a CD (Defined as follows)):

C -> (AND C C) | (OR C C)
| (NOT C)
| A[1-1000]
| 1
| 0):

1) Write a function which counts the number of times a logical operator is used in a a CD.

2) Write a function which uniquely lists all of the input VARIABLES.

3) Write a function that given a CD, reduces, the CD to a simpler form by using tautologies.

COUNTER-OPERATOR + EXAMPLES

```
;; collin gros
;; 11-18-2020
;; cs471
;; lisp
;;
;; FOR THIS ASSIGNMENT: we don't check for correctness, and we assume
;;                                    that we are given a CD beforehand.
;; (code from lecture by Shaun Cooper)
;; tells you how many of an element, x, are in a list, L,
;; including nested lists
;;
;; example i/o:
;;    i: `a `(a b (c a d))
;;    o: 2
;;

(define (howmany x L)
    (cond    ((null? L) 0)
               ((not (list? L))
                      (if (eq? x L) 1 0))
                      ;; if atom is x, 1 otherwise 0
                      (else (+ (howmany x (car L))
                                    ;; count of x in head of list
                                    (howmany x (cdr L))))))
                                    ;; count of x in remainder of list
```

EXAMPLES

```
> (load "howmany.lsp")
> (howmany `OR `(OR 0 (AND A1 (OR 1 0))))
2
> (howmany `OR `(AND A1 (OR 1 (OR A1 (OR 0)))))
3
```

```
> (howmany `AND `(OR 0 (AND A1 (OR 1 0))))
1
> (howmany `AND `(AND A1 (OR 1 (OR A1 (OR 0)))))
1
> (howmany `NOT `(AND A1 (OR 1 (NOT (OR 1 0)))))
1
```

UNIQUE + EXAMPLES

```
;; collin gros
;; 11-18-2020
;; cs471
;; lisp
;;
;; findinputvars - clean up all other stuff in a CD
;; modified version of Shaun Cooper's code from lecture
;;
;; WE GET A FLATTENED UNIQ CD
;;
;; pre: takes a list L
;; post: returns all input VARIABLES in L
(define (findinputvars L)
    (cond ((null? L) `())
        ;; if list is null return ()
        ((not (list? L)) `())
        ;; if list is an atom return ()
        ((or (eq? (car L) 1)
                (eq? (car L) 0)
                (eq? (car L) `AND)
                (eq? (car L) `OR)
                (eq? (car L) `NOT))
               ;; we want to ignore all of these so we want to examine the
               ;; rest of the list
               (findinputvars (cdr L)))
        (else (cons (car L) (findinputvars (cdr L))))))


;; uniq - extracts unique atoms from a FLATTENED list or nested lists
;;                      ---note: THE LIST MUST BE FLATTENED
;;
;; (used some modified Shaun Cooper's code from lecture)
;; pre: takes a list L
;; post: returns uniq elements in list L
(define (uniq L)
    (cond    ((null? L) `())
               ;; uniq of empty list is empty list
               ((not (list? L)) `())
               ;; uniq of junk is empty list (given is not a list)
               ((member (car L) (cdr L)) (uniq (cdr L)))
               ;; if car L is in the rest of the list, we can ignore car L
               (else (cons (car L) (uniq (cdr L))))))
               ;; if not, we need to add this element to the uniq of the
               ;; rest of the list
               ;;
               ;; e.g., (a b c) == (cons `a (uniq `(b c))
               ;;                                  (cons `b (uniq `(c)))
               ;;                                         (cons `c (uniq `())))
               ;;                                         `()


;; uniq_findinputvars
```

```
;; pre: takes a list L
;; post: returns uniq input variables in list L
(define (uniq_findinputvars L)
    (findinputvars (uniq (flatten L))))
```

EXAMPLES

```
> (uniq_findinputvars `(NOT OR AND A1 A2 NOT A3 A3 A1))
(A2 A3 A1)
> (uniq_findinputvars `(NOT AND A3 A3 A1 A2 A5 A1 (A2 A3 NOT OR A1 A7)))
(A5 A2 A3 A1 A7)
```

REDUCE + EXAMPLES

```
;; collin gros
;; 11-18-2020
;; cs471
;; lisp
;;
;; evaluates a circuit design (CD)
;; (modified code from Shaun Cooper's lecture)
;;
;; must be given a CD
;;
;; NOT CD1

(define (evalcd CD)
    (cond ((null? CD) `())
        ;; if the CD is null we have no more work to do
        ;; base case
        ((not (list? CD)) CD)
        ;; can't break atoms up; return it
        ((eq? (car CD) `NOT) (evalcd_not CD))
        ;; evaluate NOT
        ((eq? (car CD) `AND) (evalcd_and CD))
        ;; evaluate AND
        ((eq? (car CD) `OR) (evalcd_or CD))
        ;; evaluate OR

        ))


;; pre: must be in (NOT CD) form, (CAR CD) -> NOT
;; post: apply simple tautologies to CD
(define (evalcd_not CD)
    (cond ((eq? (evalcd (cadr CD)) 0) 1)
                ;; not of 0 is 1
                ((eq? (evalcd (cadr CD)) 1) 0)
                ;; not is 1 is 0
        (else (cons `NOT (list (evalcd (cadr CD)))))))
                ;; deal with rest of CD (we took care of the NOT)


;; pre: must be in (AND CD1 CD2) form
;; post: apply simple tautologies to CD1 and CD2, and may reduce AND
(define (evalcd_and CD)
    (cond ((eq? (evalcd (cadr CD)) 0) 0)
                ;; if first arg is 0 then return 0
                ((eq? (evalcd (caddr CD)) 0) 0)
                ;; if second arg is 0 then return 0
```

```
                ((eq? (evalcd (cadr CD)) 1) (evalcd (caddr CD)))
                ;; if first arg is 1, return whatever the second arg is
                ((eq? (evalcd (caddr CD)) 1) (evalcd (cadr CD)))
                ;; if second arg is 1, return whatever the first arg is

        (else (cons `AND
                      (list (evalcd (cadr CD)) (evalcd (caddr CD)))
                      ;; evaluate the rest of the list after the part we
                      ;; already examined
        ))))


;; pre: must be in (OR CD1 CD2) form
;; post: apply simple tautologies to CD1 and CD2, and may reduce OR
(define (evalcd_or CD)
    (cond ((eq? (evalcd (cadr CD)) 1) 1)
                ;; if first arg is 1 then return 1
                ((eq? (evalcd (caddr CD)) 1) 1)
                ;; if second arg is 1 then return 1
                ((eq? (evalcd (cadr CD)) (evalcd (caddr CD)))
                              (evalcd (cadr CD)))
                ;; if both args are equal, return their value (handles 0 0)

                ((eq? (evalcd (cadr CD)) 0) (evalcd (caddr CD)))
                ;; if first arg is 0, return whatever the second arg is
                ((eq? (evalcd (caddr CD)) 0) (evalcd (cadr CD)))
                ;; if second arg is 0, return whatever the first arg is

        (else (cons `OR
                      (list (evalcd (cadr CD)) (evalcd (caddr CD)))
                      ;; evaluate the rest of the list after the part we
                      ;; already examined
        ))))
```

EXAMPLES

```
> (evalcd `(AND A1 (NOT (OR 0 1))))
0
> (evalcd `(NOT (OR A1 (NOT 0))))
0
> (evalcd `(NOT (AND A5 (OR 0 (NOT 1)))))
1
```