Question 1:

1.)

A or (A and B) <-> A

|  |  |  |  |
| --- | --- | --- | --- |
| **A** | **B** | **A and B** | **A or (A and B)** |
| F | F | F | F |
| F | T | F | F |
| T | F | F | T |
| T | T | T | T |

2.)

Minterms: AB’ + AB

A + AB = AB’ + AB

A + AB = A(B’ + B)

A + AB = A

A v (A ^ B) = (A ^ A) v (A ^ B)

A v (A ^ B) = (A v B) v (A ^ B)

A v (A ^ B) = A

Question 2:

1.)

- Input: A list of values

- Output: The last value of the list

- If the list is empty to begin with, outputs an error

- The program will iterate through every value of the list before reaching the last one.

2.)

(unknown ‘(1 2 3))

- checks if input is null, because it isn’t the program will continue

- checks if the rest of the list is null, because it isn’t the program will continue

- calls unknown rest of the list = (unknown ‘(2 3))

- repeats steps above = (unknown ‘(3))

- checks if input is null, it is not

- checks if the rest of the list is null, it is and outputs 3 and ends

3.)

(cons '() (cons '() '()))

Question 3:

(define (Union list1 list2)

(cond

[(null? list1) list2]

[(null? list2) list1]

[else (if (member? (first list1) list2)

(Union (rest list1) list2)

(cons (first list1) (Union (rest list1) list2)))]

)

)

Question 4:

1.)

(define (charfun pred?)

(cond

[(null? pred?) 0]

[else (if (and #t pred?)

1

0)]

)

)

2.)

predicateprime? Is not a function (I am assuming it is below. I have commented this part out of the submission)

(map (lambda (list)(predicateprime? list))(range 1 100))

- Will produce a list of #t and #f where all primes are #t

(foldr + 0 (map (lambda(x) (if (and #t x) 1 0)) (map (lambda (list)(predicateprime? list))(range 1 100))))

- foldr will iterate over a mapped list of 1s and 0s where 1 replaced the #t from the previous map. The output will be the total number of primes

3.)

(define (appendfoldr list1 list2)

(cond

[(null? list1) list2]

[(null? list2) list1]

[else (foldr cons list2 list1)]

)

)

Question 5:

(define (remove-iff expr)

(cond

[(constant? expr) expr]

[(variable? expr) expr]

[(negation? expr) expr]

[(disjunction? expr) expr]

[(conjunction? expr) expr]

[(implication? expr) expr]

[(iff? expr) (and (implies (op1 expr)(op2 expr)) (implies (op2 expr)(op1 expr)))]

))