#lang racket

(define sudoku1 '((2 1 4 3) (4 3 2 1) (1 2 3 4) (3 4 1 2)))

(define sudoku2 '((2 1 4 3) (4 3 2 1) (1 2 3 3) (3 4 1 2)))

;(a)different ;

(define (different L)

(cond ((null? L) #t)

((member (car L) (cdr L)) #f)

(else (different (cdr L)))))

;(b) extract4Columns;

(define (extractColumns matrix index)

(cond ((null? matrix) '())

(else (cons(list-ref (car matrix) index) (extractColumns (cdr matrix) index))))) ;(extract4Columns (cdr matrix) index ))))))

; (extractColumns sudoku1 0) ;

(define(extract4Columns matrix )

(do( (i 0 (+ i 1))

(L '() (append L(list(extractColumns matrix i)))))

((= i (length matrix)) L)))

;(extract4Columns sudoku1);

;(extract4Columns sudoku2)

;(c) extract4Quadrants;

(define (getElem matrix i j)

(list-ref(list-ref matrix i )j))

;(getElem '((1 2)(3 4)) 0 0)

(define (extract4Quadrants matrix)

(let ((Q1(list (getElem matrix 0 0) (getElem matrix 0 1) (getElem matrix 1 0) (getElem matrix 1 1)))

(Q2(list (getElem matrix 0 2) (getElem matrix 0 3) (getElem matrix 1 2) (getElem matrix 1 3)))

(Q3(list (getElem matrix 2 0) (getElem matrix 2 1) (getElem matrix 3 0) (getElem matrix 3 1)))

(Q4(list (getElem matrix 2 2) (getElem matrix 2 3) (getElem matrix 3 2) (getElem matrix 3 3)))

)

(list Q1 Q2 Q3 Q4)))

;(d)merge3

(define (merge3 x y z)

(append x y z))

; (e)checkSudoku

(define (checkSudoku L)

(let\* (

(mergedList (merge3 L (extract4Quadrants L) (extract4Columns L)))

(result( map different mergedList))

)

(cond((member #f result) #f) (else #t))

))

;(checkSudoku sudoku1)

;(checkSudoku sudoku2)

(different '(1 3 6 4 8 0))

(different '(1 3 6 4 1 8 0))

(extract4Columns sudoku1)

(extract4Quadrants sudoku1)

(merge3 '(1 3 6) '(5 4) '(1 2 3))

(checkSudoku sudoku1)

(checkSudoku sudoku2)