CMSC 10500-1: Homework 2

(due on Monday June 28th)

Consider the following scheme definitions for representing quadratic equations and their solutions:

```
;; structure representing the equation
;; quadratic*x^2 + linear*x + constant = 0
(define-struct quad-eqn (constant linear quadratic))

;; structure representing the roots of an equation
;; type is 'Degenerate 'Zero 'One 'Two
;; first and second are numbers
(define-struct quad-roots (type first second))
```

1. **(5 pts)** Write a function `solve` which takes as input a quadratic equation, represented by the `quad-eqn` structure, and returns the roots represented by the `quad-roots` structure.

   The `type` should be set to `'Degenerate`, `'Zero`, `'One` or `'Two` depending on whether the equation is degenerate and the number of real roots of the equation. If there are no real roots, `first` and `second` should both be set to zero. In case of one real root `second` should be set to zero. In the degenerate case, the linear term may assumed to be non-zero.

2. **(5 pts)** Write a function `equation` which takes an input a `quad-roots` structure and returns an equation (represented by a `quad-eqn` structure) whose roots is the given input. If the given input has type `'Zero`, then the equation returned should correspond to the quadratic equation: $x^2 + 1 = 0$.

Solution for questions 1 and 2

Suppose the given equation is

$$ax^2 + bx + c = 0$$

We first need to check if the coefficient of the quadratic term (i.e. a) is zero. If so, then this is a degenerate equation whose only root is $\frac{-c}{b}$ (this is where we assume that $b \neq 0$).

If equation is not degenerate, we calculate the discriminant (i.e. b^2-4ac) and depending on whether it is zero, positive or negative the number of real roots is One, Two or Zero. In case of one root it is given by $(-b/2a)$ and for two roots it is given by

$$\frac{-b \pm \sqrt{b^2-4ac}}{2a}$$

Conversely, given the roots we first inspect the type. If the type is 'Zero we output the equation x^2+1 (as specified in the problem). If the type is 'Degenerate then the equation is $x-\alpha=0$ where α is the value of first root. If the type is 'One, then the equation is $x^2-2\alpha x+\alpha^2=0$. Finally if the type is 'Two, then the equation is $x^2-(\alpha+\beta)x+\alpha\beta=0$ where β is the second root.

The scheme program to solve and construct the equations are given below:

```
;; discriminant: quad-eqn -> number
;; evaluate the discriminant of a quadratic equation (b^2 - 4ac)
(define (discriminant e)
  (- (sqr (quad-eqn-linear e))
       (* 4 (* (quad-eqn-quadratic e)
               (quad-eqn-constant e))))
)

;; nondeg-solve: quad-eqn number -> quad-roots
;; Solve the non-degenerate case of a quadratic eqn given the discriminant
(define (nondeg-solve e disc)
  (cond
    [(= disc 0) (make-quad-roots 'One
                                 (/ (- (quad-eqn-linear e))
                                    (* (quad-eqn-quadratic e) 2))
                                 0
                                 )]
    [(< disc 0) (make-quad-roots 'Zero 0 0)]
    [else (make-quad-roots 'Two
                           (/ (+ (- (quad-eqn-linear e)) (sqrt disc))
                              (* 2 (quad-eqn-quadratic e)))
                           (/ (- (+ (quad-eqn-linear e) (sqrt disc)))
                              (* 2 (quad-eqn-quadratic e))))]
    )
  )
```

```
;; solve : quad-eqn -> quad-roots
;; Solve the given quadratic equation
(define (solve e)
  (cond
    [(= (quad-eqn-quadratic e) 0)
     (make-quad-roots 'Degenerate
                      (/ (- (quad-eqn-constant e))
                         (quad-eqn-linear e))
                      0)]
    [else (nondeg-solve e (discriminant e))]
   )
)

;; equation : quad-roots -> quad-eqn
;; Find an equation whose roots are given as input
(define (equation roots)
  (cond
    [(symbol=? (quad-roots-type roots) 'Degenerate)
     (make-quad-eqn (quad-roots-first roots) -1 0)]
    [(symbol=? (quad-roots-type roots) 'Zero)
     (make-quad-eqn 1 0 1)]
    [(symbol=? (quad-roots-type roots) 'Two)
     (make-quad-eqn (* (quad-roots-first roots)
                       (quad-roots-second roots))
                    (- (+ (quad-roots-first roots)
                          (quad-roots-second roots)))
                    1
               )]
    [else
     (make-quad-eqn (sqr (quad-roots-first roots))
                    (* -2 (quad-roots-first roots))
                    1)]
    )
  )
```

3. **(10 pts)** Write a function `configuration`, which takes three points on the plane as input (each point represented by a `posn` structure) and returns one of the following:

   (a) `'CoLinear` if the three points are on a straight line.

   (b) `'RightTriangle` if the three points form a right angled triangle.

   (c) `'Neither` if none of the above two conditions hold.

Hint: Calculate the pair-wise distances first.

Solution for problem 3

Let the three points given be P, Q and R. Let a, b and c denote the distance between Q and R, P and R, and P and Q respectively. P, Q and R lie on a straight line, exactly when the sum of two of these distances equals the third. Similarly P, Q and R form a right triangle exactly when the sum of the squares of two of these distances equals the square of the third (square of hypotenuse = sum of squares of other two sides).

To check if the three points are in a straight line, we need to check if the sum of two of a, b, c equals the third. It is almost trivial to see that if the sum of two equals the third, then the third has to be the largest of the three. Thus an equivalent way of stating the same is that $a + b + c = 2 * \max(a, b, c)$.

Similarly if sum of squares of two of a, b, c equals the square of the third, then the third has to be largest of the three. Hence this can be written as $a^2+b^2+c^2 = 2*\max(a, b, c)^2$.

The scheme program for this is given below:

```
;; distance: posn posn -> number
;; Returns the distance between the two points
(define (distance a b)
  (sqrt (+
     (sqr (- (posn-x a) (posn-x b)))
     (sqr (- (posn-y a) (posn-y b)))
     )))

;; sumsqr: number number number -> number
;; Returns the sum of squares
(define (sumsqr a b c)
      (+ (sqr a) (sqr b) (sqr c)))

;; config-helper: number number number -> symbol
;;   The three numbers are the pair wise distances
;; Return 'CoLinear, 'RightTriangle or 'Neither
(define (config-helper a b c)
  (cond
    [(= (+ a b c) (* 2 (max a b c))) 'CoLinear]
    [(= (sumsqr a b c) (* 2 (sqr (max a b c)))) 'RightTriangle]
    [else 'Neither]
    )
)
```

```
;; configuration: posn posn posn -> symbol
;;   Given three points on the plane return
;;   'CoLinear 'RightTriangle 'Neither depending on whether
;;   the three points are co-linear, form a right triangle or neither
(define (configuration P Q R)
  (config-helper (distance P Q) (distance P R) (distance Q R))
  )
```

If you did not know about the `max` operator in scheme, one can define a function `max3` for that as follows:

```
;; max2 : number number -> number
;;   Return the larger of the two numbers
(define (max2 x y)
  (cond
    [(> x y) x]
    [else y]
  )
)

;; max3 : number number number -> number
;;    Return maximum of three numbers
(define (max3 x y z)
  (max2 (max2 x y) z)
)
```

Comments

You can download the scheme code presented in the above solution from here.