**Structures and Interpretation of Computer Program**

**Exercise Chapter 2.1 Name:** Wan Huzaifah bin Wan Azhar

**Exercise 2.1.3 What is meant by data**



(define (lower-bound x)

(min (car x) (cdr x)))

(define (upper-bound x)

(max (car x) (cdr x)))



* Interval addition results in the maximum upper and lower-bound value that both interval can have.
* Interval subtraction thus result in the minimum upper and lower-bound value that both interval can have.
* As such, to subtract an interval, an interval must subtract with minimal value of the interval to be subtracted
* Assume interval A = (x1, x2), B= (y1, y2) with x1 < x2 and y1 < y2
* A – B. -B is = (-y1, -y2) or (-y2, -y1) since -y2 is now lower than -y1.
* A – B = A + (-B) = ((x1 – y2), (x2 – y1))

(define (sub-interval x y)

(make-interval

(- (lower-bound x) (upper-bound y))

(- (upper-bound x) (lower-bound y))))

(define (interval-a) (make-interval 4 5))

(define (interval-b) (make-interval 5 3))

(display (sub-interval (interval-b) (interval-a)))



* Width is half of difference between upper bound and lower bound of an interval.
* Prove that width of addition or subtraction of two interval is addition of subtraction of two width interval.
* A = (x1, x2) x1 < x2 and B = (y1, y2) y1 < y2.
* A + B = x1 + y1, x2 + y2
* Width(A+B) = (x2 + y2 – x1 – y1)/2
* Width(A+B) = (x2 – x1)/2 + (y2 – y1)/2
* From definition, width(A) = (x2 – x1)/2
* So, Width(A+B) = width(A) + width(B)
* Proves that width of addition of two interval is addition of two width interval.
* It should also proves the same for substraction.
* With multiplication or division, the hypothesis is false.
* Recall that multiplication of two interval, AB = min{x1y1,x1y2,x2y1,x2y2}, max{x1y1,x1y2,x2y1,x2y2}
* Width(AB) is then (max{x1y1,x1y2,x2y1,x2y2} - min{x1y1,x1y2,x2y1,x2y2})/2
* There can be many possibility of the result as multiplication use max and min function.
* As such, multiplication of width of two interval can result in width of multiplication of two interval but the opposite can also be true.
* However, as not all multiplication of width of two interval can result in width of multiplication of two interval, the hypothesis is false for multiplication.
* It is also the same for division, as it also uses multiplication.



(define (div-interval x y)

(if (or (= 0 (lower-bound y)) (= 0 (upper-boud y))) (error "Attempt to divide interval by zero")

(mul-interval x

(make-interval (/ 1.0 (upper-bound y))

(/ 1.0 (lower-bound y))))))



Exactly 9 cases with only one case have more than two multiplication (both interval are opposite sign)

(define (make-interval a b)

(if (< a b)

(cons a b)

(cons b a)))

(define (lower-bound x)

(min (car x) (cdr x)))

(define (upper-bound x)

(max (car x) (cdr x)))

(define (mul-interval x y)

(let ((p1 (\* (lower-bound x) (lower-bound y)))

(p2 (\* (lower-bound x) (upper-bound y)))

(p3 (\* (upper-bound x) (lower-bound y)))

(p4 (\* (upper-bound x) (upper-bound y))))

(make-interval (min p1 p2 p3 p4)

(max p1 p2 p3 p4))))

(define (new-mul-interval x y)

(define (check-interval i)

(cond ((and (< 0 (lower-bound x)) (< 0 (upper-bound x))) -1)

((and (> 0 (lower-bound x)) (> 0 (upper-bound x))) 1)

(else 0)))

(cond ((and (= 1 (check-interval x))(= 1 (check-interval y))) (make-interval (\* (lower-bound y) (lower-bound x)) (\* (upper-bound y) (upper-bound x))))

((and (= -1 (check-interval x))(= -1 (check-interval y))) (make-interval (\* (lower-bound y) (lower-bound x)) (\* (upper-bound y) (upper-bound x))))

((and (= 1 (check-interval x))(= -1 (check-interval y))) (make-interval (lower-bound y) (upper-bound x)))

((and (= -1 (check-interval x))(= 1 (check-interval y))) (make-interval (lower-bound x) (upper-bound y)))

((and (= 0 (check-interval x))(= 1 (check-interval y))) (make-interval (lower-bound x) (upper-bound y)))

((and (= 0 (check-interval x))(= -1 (check-interval y))) (make-interval (lower-bound y) (upper-bound x)))

((and (= -1 (check-interval x))(= 0 (check-interval y))) (make-interval (lower-bound x) (upper-bound y)))

((and (= 1 (check-interval x))(= 0 (check-interval y))) (make-interval (lower-bound y) (upper-bound x)))

((and (= 0 (check-interval x))(= 0 (check-interval y))) (mul-interval x y))

))

(define (interval-a) (make-interval 4 -3))

(define (interval-b) (make-interval -5 3))

(display (mul-interval (interval-a) (interval-b)))

(newline)

(display (new-mul-interval (interval-a) (interval-b)))



(define (make-center-percent c p)

(let ((additive-tolerance (\* c (/ p 100))))

(make-interval (- c additive-tolerance) (+ c additive-tolerance))))

;Not using center and width selector to avoid div by zero

(define (percent i)

(\* 100 (/ (- (upper-bound i) (lower-bound i)) (+ (upper-bound i) (lower-bound i)))))

1. Skipped because of difficulty

(define (make-interval a b)

(if (< a b)

(cons a b)

(cons b a)))

(define (lower-bound x)

(min (car x) (cdr x)))

(define (upper-bound x)

(max (car x) (cdr x)))

(define (add-interval x y)

(make-interval (+ (lower-bound x) (lower-bound y))

(+ (upper-bound x) (lower-bound y))))

(define (mul-interval x y)

(let ((p1 (\* (lower-bound x) (lower-bound y)))

(p2 (\* (lower-bound x) (upper-bound y)))

(p3 (\* (upper-bound x) (lower-bound y)))

(p4 (\* (upper-bound x) (upper-bound y))))

(make-interval (min p1 p2 p3 p4)

(max p1 p2 p3 p4))))

(define (div-interval x y)

(mul-interval x

(make-interval (/ 1.0 (upper-bound y))

(/ 1.0 (lower-bound y)))))

(define (width i)

(/ (- (lower-bound i) (upper-bound i)) 2))

(define (center i)

(/ (+ (lower-bound i) (upper-bound i)) 2))

(define (make-center-percent c p)

(let ((additive-tolerance (\* c (/ p 100))))

(make-interval (- c additive-tolerance) (+ c additive-tolerance))))

;Not using center and width selector to avoid div by zero

(define (percent i)

(\* 100 (/ (- (upper-bound i) (lower-bound i)) (+ (upper-bound i) (lower-bound i)))))

(define (par1 r1 r2)

(div-interval (mul-interval r1 r2)

(add-interval r1 r2)))

(define (par2 r1 r2)

(let ((one (make-interval 1 1)))

(div-interval one

(add-interval (div-interval one r1)

(div-interval one r2)))))

(display (par1 (make-interval 0 2) (make-interval 3 5)))

(newline)

(display (par2 (make-interval 0 2) (make-interval 3 5)))

(newline)

(define (interval-a) (make-center-percent 6 0.001))

(define (interval-b) (make-center-percent 2 0.001))

(define (A/A) (div-interval (interval-a) (interval-a)))

(define (A/B) (div-interval (interval-a) (interval-b)))

(newline)

(display "Center A/A ")

(display (center (A/A)))

(newline)

(display "Percentage Tolerance A/A ")

(display (percent (A/A)))

(newline)

(display "Center A/B ")

(display (center (A/B)))

(newline)

(display "Percentage Tolerance A/B ")

(display (percent (A/B)))

* Insight on A/A
  + A/A will always return 1 if the width percentage tolerance is small
  + As division is defined as (x1, x2) \* (1/(y1,y2)
  + A/A then = (x1, x2) \* (1/(x1,x2)
  + = [min{x1 \* 1/x1, x1 \* 1/x2 …}, max{ x1 \* 1/x1, x1 \* 1/x2 …}]
  + = [min{x1/x1, x1/x2, x2/x1, x2/x2}, max{x1/x1, x1/x2, x2/x1, x2/x2}]
  + = [min{1, x1/x2, x2/x1, 1}, max{1, x1/x2, x2/x1, 1}]
  + x1= x2 as x1 and x2 has small width
  + Therefore,
  + = [min{1, 1, 1, 1}, max{1, 1, 1, 1}]
  + A/A will always equal 1 if width percentage tolerance is small
* Insight on A/B
  + A/B will return approximately (center(A)/center(B)) if width is small percentage of center value.
  + As division is defined as (x1, x2) \* (1/(y1,y2)
  + A/B then = (x1, x2) \* (1/(y1,y2)
  + = [min{x1 \* 1/x1, x1 \* 1/x2 …}, max{ x1 \* 1/x1, x1 \* 1/x2 …}]
  + = [min{x1/x1, x1/x2, x2/x1, x2/x2}, max{x1/x1, x1/x2, x2/x1, x2/x2}]
  + X1 and x2 is close to center(x) as both has small width. Same is for y1 and y2
  + Therefore [{center(x)/center(y)}, max{center(x)/center(y)}]
  + = [center(x)/center(y), center(x)/center(y)]