Homework 04

● Graded

Student

Giancarlos Marte

Total Points

88 / 100 pts

Autograder Score

88.0 / 100.0

Failed Tests

Exercise 10. exp-to-string (0/12)

Passed Tests

Exercise 1. stream-skip (correctness) (5/5)
Exercise 1. stream-skip (eagerness) (3/3)
Exercise 2. stream-fold (correctness) (7/7)
Exercise 2. stream-fold (eagerness) (5/5)
Exercise 3. set-void (8/8)
Exercise 4. set-epsilon (8/8)
Exercise 5. set-char (8/8)
Exercise 6. set-prefix (correctness) (6/6)
Exercise 6. set-prefix (eagerness) (4/4)
Exercise 7. set-union (correctness) (6/6)
Exercise 7. set-union (eagerness) (5/5)
Exercise 8. set-concat (correctness) (6/6)
Exercise 8. set-concat (eagerness) (5/5)
Exercise 9. r:eval-exp (0 to +10 args) (6/6)
Exercise 9. r:eval-exp (nested expressions) (6/6)

Autograder Results

Exercise 1. stream-skip (correctness) (5/5)

Exercise 1. stream-skip (eagerness) (3/3)

Exercise 10. exp-to-string (0/12)

Exercise 2. stream-fold (correctness) (7/7)

Exercise 2. stream-fold (eagerness) (5/5)

Exercise 3. set-void (8/8)

Exercise 4. set-epsilon (8/8)

Exercise 5. set-char (8/8)

Exercise 6. set-prefix (correctness) (6/6)

Exercise 6. set-prefix (eagerness) (4/4)

Exercise 7. set-union (correctness) (6/6)

Exercise 7. set-union (eagerness) (5/5)

Exercise 8. set-concat (correctness) (6/6)

Exercise 8. set-concat (eagerness) (5/5)

Exercise 9. r:eval-exp (0 to +10 args) (6/6)

Exercise 9. r:eval-exp (nested expressions) (6/6)

Submitted Files

hw4.rkt Download

```
#lang errortrace typed/racket
#|
   ===> PLEASE DO NOT DISTRIBUTE SOLUTIONS NOR TESTS PUBLICLY <===

  We ask that solutions be distributed only locally -- on paper, on a
  password-protected webpage, etc.

  Students are required to adhere to the University Policy on Academic
  Standards and Cheating, to the University Statement on Plagiarism and the
  Documentation of Written Work, and to the Code of Student Conduct as
  delineated in the catalog of Undergraduate Programs. The Code is available
  online at: http://www.umb.edu/life_on_campus/policies/code/

|#
(require "hw4-util.rkt")
(provide (all-defined-out))

; stream get & next
(: stream-get : (All [Elem] (stream Elem) -> Elem))
(define (stream-get strm)
  (match (strm)
    [(stream-add first rest) first]))

(: stream-next : (All [Elem] (stream Elem) -> (stream Elem)))
(define (stream-next strm)
  (match (strm)
    [(stream-add first rest) rest]))


(: stream-skip : (All [Elem] Real (stream Elem) -> (stream Elem)))
; Parameterized on the type of the elements of the stream
; The first argument is the number of elements we wish to skip,
; and the second argument is the stream.

(define (stream-skip n s)
  (cond
    [(equal? 0 n) s]
    [else
     (stream-skip (- n 1) (stream-next s))]))


(: stream-fold : (All [Elem Accum]
                  (Elem Accum -> Accum)  ;; "step" function
                  Accum                  ;; initial accumulator
                  (stream Elem)          ;; stream to process
                  ->
```

```
                    (stream Accum)))
; We have 2 type parameters,
; 1. the type of elements of the stream
; 2. the type of the result being accumulated

(define (stream-fold f a s)
  (thunk
   (stream-add
    a
    (stream-fold
     f
     (f (stream-get s) a)
     (stream-next s)))))

; set get & next (not used)
(: set-get : set -> String)
(define (set-get s)
  (match s
    [(set-add first rest) first]))

(: set-next : set -> set)
(define (set-next s)
  (match s
    [(set-add first rest) rest]))


(: set-void : set)
(define set-void
  set-empty)

(: set-epsilon : set)
(define set-epsilon
  (thunk (set-add "" set-empty)))

(: set-char : Char -> set)
(define (set-char x)
  (thunk (set-add (string x) set-empty)))

(: set-prefix : String set -> set)
(define (set-prefix s p)
  (thunk
   (match (p)
     [(set-empty) (p)]
     [(set-add f r)
      (set-add
       (string-append s f)
       (set-prefix s r))])))

(: set-union : set set -> set)
```

```
(define (set-union p1 p2)
  (thunk
   (match (p1)
     [(set-empty) (p2)]
     [(set-add f r)
      (set-add
       f
       (set-union p2 r))])))


(: set-concat : set set -> set)
(define (set-concat p1 p2)
  (thunk
   (match (p1)
     [(set-empty) (p1)]
     [(set-add f r)
      ((set-union
        (set-prefix f p2)
        (set-concat r p2)))])))


(: r:eval-exp : r:expression -> Number)
(define (r:eval-exp exp)
  (match exp

    ; If it's a number, return that number
    [(r:number v) v]
    ; If it's a function with 2 arguments
    [(r:apply (r:variable f) (list arg1 arg2))
     (define func (r:eval-builtin f))
     (func (r:eval-exp arg1) (r:eval-exp arg2))]

    ; If it's a function with multiple arguments
    [(r:apply (r:variable f) (list x ...))
     (define func (r:eval-builtin f))
     (define y (map r:eval-exp x))
     (apply func y)]))

(: r:exp-to-string : r:expression -> String)
(define (r:exp-to-string exp)
  (match exp
    ; convert num to string
    [(r:number v) (number->string v)]
    ;convert variable to string for single variables
    [(r:variable v) (symbol->string v)]
    ; convert apply to string
    [(r:apply (r:variable f) (list argR ...))
     ; Listof expr -> Listof String
     (define strList (map r:exp-to-string argR))
```

```
    ; add */+ to Listof String
    (define temp (append (list (symbol->string f)) strList))
    ; add spaces while turning to list to a single string
    (define temp2 (join " " temp))
    ; add parenthesis
    (string-append "(" temp2 ")")]))
```