Fast Sequence Combinators

Anna Bolotina

Programming Research Laboratory
Czech Technical University in Prague
ann-bolotina@yandex.ru

RacketFest 2020
Programming quiz

- What is the sum of squares of primes up to 10000?
- What is the product of squares of primes up to 10000?
- What is the first square of a prime that ends in 1?
(define squares-of-primes-up-to-10000
  (map sqr
    (filter prime?
      (range 0 10000)))))

(for/sum ([((x) squares-of-primes-up-to-10000)]
  x)

(for/product ([((x) squares-of-primes-up-to-10000)]
  x)

(for/first ([((x) squares-of-primes-up-to-10000]
    #:when (ends-in-1? x))
  x)
(define squares-of-primes-up-to-10000
  (sequence-map
   sqr
   (sequence-filter
    prime?
    (in-range 0 10000))))

(for/sum ([[(x) squares-of-primes-up-to-10000]]) x)

(for/product ([[(x) squares-of-primes-up-to-10000]]) x)

(for/first ([[(x) squares-of-primes-up-to-10000]]
            #:when (ends-in-1? x)) x)
(for/sum ([(x) (in-range 0 10000)])
  (if prime?
    (sqr x)
    0))

(for/or ([(x) (in-range 0 10000)])
  (if (prime? x)
    (let ([x* (sqr x)])
      (if (ends-in-1? x*)
        x*
        #f))
    #f))
(define-syntax-rule (squares-of-primes-up-to-10000)
  (fast-sequence-map
    sqr
    (fast-sequence-filter
      prime?
      (in-range 0 10000))))
for loop structure

(for ([x (in-list '(1 2 5 3 10))])
  (println x))

⇒

(let loop ([lst '(1 2 5 3 10)])
  (when (pair? lst)
    (let ([x (car lst)]
           [rest (cdr lst)])
      (begin
        (println x)
        (loop rest)))))
Loop with two for clauses

(for [[(x) (in-list '(1 2 5 3 10))] [(y) (in-range 0 7 1)]
  (println (list x y)))

⇒

(let loop ([lst '(1 2 5 3 10)] [pos 0])
  (when (and
    (pair? lst)
    (< pos 7))
    (let ([x (car lst)]
          [rest (cdr lst)]
          [y pos]
          [next (+ 1 pos)])
      (begin
        (println (list x y))
        (loop rest next)))))
fast-sequence-map

(for ([x] (fast-sequence-map
    sqr
    (in-list '(1 2 5 3 10))))
  (println x))

⇒

(let loop ([lst '(1 2 5 3 10)])
  (when (pair? lst)
    (let ([x* (car lst)]
        [rest (cdr lst)])
      (let ([x (sqr x*)])
        (begin
          (println x)
          (loop rest)))))
fast-sequence-filter: Motivating a structure

(for/list ([x (in-list '(a b c d e))]
            [y (fast-sequence-filter
                odd?
                (in-list '(1 2 3 4 5 6)))]
          (list x y))

Wrong

\[
\begin{array}{cccccc}
  x & a & b & c & d & e \\
  y & 1 & 3 & 5 \\
\end{array}
\]

\[
\Rightarrow
\begin{array}{ccc}
  '(a 1) & '(c 3) & '(e 5) \\
\end{array}
\]

Right

\[
\begin{array}{cccccc}
  x & a & b & c & d & e \\
  y & 1 & 3 & 5 & \_ \\
\end{array}
\]

\[
\Rightarrow
\begin{array}{ccc}
  '(a 1) & '(b 3) & '(c 5) \\
\end{array}
\]
(for ([x (in-list '(a b c d e))] [y (fast-sequence-filter odd? (in-list '(1 2 3 4 5 6)))]
  (println (list x y)))
⇒
(let loop ([lst1 '(a b c d e)] [lst2 '(1 2 3 4 5 6)])
  (when (and (pair? lst1) #t)
    (let-values
      ([x (car lst1)]
       [rest1 (cdr lst1)]
       #| find the next y and rest2, or else y is done |#
      
      (when y-is-found
        (println (list x y))
        (loop rest1 rest2))))))
fast-sequence-filter

(for ([x (in-list '(a b c d e))]
    [y (fast-sequence-filter
        odd?
        (in-list '(1 2 3 4 5 6)))]
    (println (list x y)))

⇒

(let loop ([lst1 '(a b c d e)] [lst2 '(1 2 3 4 5 6)])
  (when (and (pair? lst1) #t)
    (let-values
      ([x (car lst1)]
        [(rest1) (cdr lst1)]
        [y rest2 y-is-found]
        (let loop ([lst lst2])
          (cond [(pair? lst)
                 (let ([y (car lst)]
                        [rest (cdr lst)])
                   (cond
                     [(odd? y) (values y rest2 #t)]
                     [else (loop rest2)])])]
                [else (values #f #f #f)]))))
  (when y-is-found
    (println (list x y))
    (loop rest1 rest2)))