Efficient Sequence Processing

Sequence Operations

Map, filter, and reduce express sequence manipulation using compact expressions

Example: Sum all primes in an interval from \( a \) (inclusive) to \( b \) (exclusive)

```scheme
(define (sum_primes a b)
  (let loop ((total 0) (x a))
    (if (< x b) (loop (+ total (if (is_prime x) x 0)) (+ x 1)) total)))
```

Streams are Lazy Scheme Lists

A stream is a list, but the rest of the list is computed only when needed:

```scheme
(define (range-stream a b)
  (if (> a b) nil (cons-stream a (range-stream (+ a 1) b))))

(define lots (range-stream 1 10000000000000000000))
```

Errors only occur when expressions are evaluated:

```scheme
(error (car (cdr-stream (range-stream 1 0))))
(error (car (cdr-stream (range-stream 1 1))))
(error (car (cdr-stream (range-stream 1 2))))
(error (car (cdr-stream (range-stream 1 3))))
```

Infinite Streams

Stream Ranges are Implicit

A stream can give on-demand access to each element in order

```scheme
(define (range-stream a b)
  (if (> a b) nil (cons-stream a (range-stream (+ a 1) b))))

(define lots (range-stream 1 10000000000000000000))
```

```scheme
(error (car (cdr-stream lots)))
(error (car (cdr-stream (cdr-stream lots))))
```
A Stream of Primes

For any prime k, any larger prime must not be divisible by k.

The stream of integers not divisible by any k \leq n is:
- The stream of integers not divisible by any k < n
- Filtered to remove any element divisible by n

This recurrence is called the Sieve of Eratosthenes

\[ 2, 3, 5, 7, 11, 13 \]