Streams

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).

```python
def sum_primes(a, b):
    total = 0
    x = a
    while x < b:
        if is_prime(x):
            total += x
        x += 1
    return total
```

```python
def sum_primes(a, b):
    return sum(filter(is_prime, range(a, b)))
```

Space: Constant
Also Constant

Streams are Lazy Scheme Lists

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

```scheme
(car (cons 1 nil)) -> 1
(car (cons-stream 1 nil)) -> 1
(cdr (cons 1 nil)) -> ()
(idr (cons-stream 1 nil)) -> ()
(cons 1 (cons 2 nil))
(cons-stream 1 (cons-stream 2 nil))
```

Errors only occur when expressions are evaluated:

```scheme
(car (cons 1 (cons nil 2)))  -> 1
(car (cons-stream 1 (cons-stream nil 2)))  -> 1 . #promise [not forced]!
(car (cons-stream 1 (cons-stream 1 2)))  -> ERROR
(idr-stream (cons-stream 1 (cons-stream 1 nil)))  -> ERROR
(idr-stream (cons-stream 1 (cons-stream 1 nil)))  -> ERROR
```
Stream Ranges are Implicit

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

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

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

Infinite Streams

Infinite streams are not yet computed when the stream is created.

```
(define (int-stream start)
  (cons-stream start (int-stream (+ start 1))))
```

Integer Stream

An integer stream is a stream of consecutive integers.

The rest of the stream is not yet computed when the stream is created.

```
(define (int-stream start)
  (cons-stream start (int-stream (+ start 1))))
```

Stream Processing

Combine two streams by separating each into car and cdr:

```
(define (add-streams s t)
  (cons-stream (+ (car s) (car t))
               (add-streams (cdr-stream s) (cdr-stream t))))
```

Integer + Integer Stream

```
(define ints (cons-stream 1 (add-streams ones ints))), 1 2 3 4 5 6 7 ...
```

Recursively Defined Streams

The rest of a constant stream is the constant stream.

```
(define ones (cons-stream 1 ones))
```

Higher-Order Stream Functions

Combine two streams by separating each into car and cdr:

```
(define (add-streams s t)
  (cons-stream (+ (car s) (car t))
               (add-streams (cdr-stream s) (cdr-stream t))))
```

Integer + Integer Stream

```
(define ints (cons-stream 1 (add-streams ones ints))), 1 2 3 4 5 6 7 ...
```
Higher-Order Functions on Streams

Implementations are identical, but change cons to cons-stream and change cdr to cdr-stream:

```scheme
(define map-stream f s)
  (if (null? s)
      nil
      (cons-stream (f (car s))
        (map-stream f (cdr-stream s))))

(define filter-stream f s)
  (if (null? s)
      nil
      (if (f (car s))
          (cons-stream (car s)
                        (filter-stream f (cdr-stream s)))
          (filter-stream f (cdr-stream s))))

(define reduce-stream f s start)
  (if (null? s)
      start
      (reduce-stream f (cdr-stream s)
                    (f start (car s))))
```

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

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

(Demo)