MASSACHVSETTS INSTITVTE OF TECHNOLOGY
Department of Electrical Engineering and Computer Science
6.001—Structure and Interpretation of Computer Programs
Fall Semester, 1996

Lecture Notes, November 7 – Infinite Streams

The stream data abstraction:

```
(define (integers-from n)
  (cons-stream n
               (integers-from (+ 1 n))))

(define integers (integers-from 1))
```

Implementation Strategies

- Use lazy evaluator declarations
- Extend eval with delays as "thunks" or "promises"
- Extend eval with delays as procedures (or memoized procedures)

Stream Higher Order Procedures

```
(define (stream-ref s n)
  (if (= n 0)
      (stream-car s)
    (stream-ref (stream-cdr s) (- n 1))))

(define (stream-map proc stream)
  (if (stream-null? stream)
      the-empty-stream
      (cons-stream (proc (stream-car stream))
                   (stream-map proc (stream-cdr stream)))))

(define (stream-filter pred s)
  (cond ((stream-null? s) the-empty-stream)
        ((pred (stream-car s))
         (cons-stream (stream-car s)
                      (stream-filter pred (stream-cdr s))))
        (else (stream-filter pred (stream-cdr s)))))

(define (enumerate-interval low high)
  (if (> low high)
      the-empty-stream
      (cons-stream low
                   (enumerate-interval (+ low 1) high))))
```

Examples

```
(define no-sevens
  (stream-filter (lambda (n) (not (divisible? n 7)))
                 integers))

(define (divisible? x y) (= (remainder x y) 0))

(stream-ref no-sevens 100)
; Value: 117
```

The Sieve of Erasthosthenes - Prime Number Generation

```
(define (sieve s)
  (cons-stream (stream-car s)
               (sieve (stream-filter
                       (lambda (x) (not (divisible? x (stream-car s))))
                       (stream-cdr s)))))
(define primes (sieve (integers-from 2)))
(stream-ref primes 200)
```

Other Stream Utilities

```
(define (show-stream s n)
  (cond ((= n 0) 'done)
        (else (write-line (stream-car s))
              (show-stream (stream-cdr s) (- n 1)))))

(define (add-streams s1 s2)
  (cond ((stream-null? s1) s2)
        ((stream-null? s2) s1)
        (else
         (cons-stream (+ (stream-car s1) (stream-car s2))
                      (add-streams (stream-cdr s1) (stream-cdr s2))))))

(define (scale-stream c s)
  (stream-map (lambda (x) (* x c)) s))

(define (stream-map2 proc s1 s2)
  (if (stream-null? s1)
      the-empty-stream
      (cons-stream (proc (stream-car s1) (stream-car s2))
                   (stream-map2 proc (stream-cdr s1) (stream-cdr s2)))))
```

Examples

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

(define integers (cons-stream 1 (add-streams ones integers)))
```

A worksheet or table for the stream:

Stream of Fibonacci Numbers

```
(define fibs
  (cons-stream 0
               (cons-stream 1
                            (add-streams (stream-cdr fibs)
                                         fibs))))
```

A worksheet or table for the stream:

Stream of Square Roots

```
(define (sqrt-improve guess x)
  (average guess (/ x guess)))
(define (average a b) (/ (+ a b) 2))

(define (sqrt-stream x)
  (cons-stream 1.0
               (stream-map (lambda (g)
                             (sqrt-improve g x))
                           (sqrt-stream x))))

;; Follow the stream until desired tolerance is reached
(define (stream-limit s tol)
  (define (iter s)
    (let ((f1 (stream-car s))
          (f2 (stream-car (stream-cdr s))))
      (if (close-enuf? f1 f2 tol)
          f2
          (iter (stream-cdr s)))))
  (iter s))

(define (close-enuf? x y tol)
  (< (abs (- x y)) tol))

(stream-limit (sqrt-stream 2) 1.e-5)
```

Trapezoidal Integration

```
(define (trapezoid f a b h)
  (let ((dx (* (- b a) h))
        (n (/ 1 h)))
    (define (iter i sum)
      (let ((x (+ a (* i dx))))
        (if (>= i n)
            sum
            (iter (+ i 1) (+ sum (f x))))))
    (* dx (iter 1 (+ (/ (f a) 2) (/ (f b) 2))))))
```

The Witch of Agnesi and Approximations to π

```
(define (witch x)
  (/ 4 (+ 1 (* x x))))

(trapezoid witch 0. 1. .1)
;Value: 3.1399259889071587

(trapezoid witch 0. 1. .01)
;Value: 3.141575986923129
```

To learn more about Maria Agnesi, see `http://www.agnesscott.edu/lriddle/women/agnesi.htm`.

Stream of Approximations to π

```
(define (keep-halving R h)
  (cons-stream (R h)
               (keep-halving R (/ h 2))))

(show-stream (keep-halving (lambda (h) (trapezoid witch 0 1 h)) 0.1)
             10)

(stream-limit
 (keep-halving (lambda (h) (trapezoid witch 0 1 h)) 0.5)
 1.e-9)
```

Accelerating the Approximation

Suppose we want to approximate a function $R(0)$ and we have the sequence of values $R(h), R(h/2), R(h/4), \cdots$. Suppose we also know that R has the form $R(h) = A + Bh^p + Ch^{2p} + Dh^{3p} + \cdots$. Then

$$\frac{2^p R(h/2) - R(h)}{2^p - 1} = A + C_2 h^{2p} + D_2 h^{3p} + \cdots \tag{1}$$

That is to say, this new sequence converges to the same value as the original, but it converges faster. We can formulate this as a stream process:

```
(define (accel-halving-seq s p)
  (let ((2**p (expt 2 p)))
    (let ((2**p-1 (- 2**p 1)))
      (stream-map2 (lambda (Rh Rh/2)
                     (/ (- (* 2**p Rh/2)
                           Rh)
                        2**p-1))
                   s
                   (stream-cdr s)))))
```

Make a table, where each row is the accelerated version of the previous row.

```
(define (make-tableau s p)
  (define (rows s order)
    (cons-stream s
                 (rows (accel-halving-seq s order)
                       (+ order p))))
  (rows s p))
```

Finally, take just the first element of each row to get the "Richardson acceleration" of the original series:

```
(define (richardson-accel s p)
  (stream-map stream-car
              (make-tableau s p)))

(show-stream  (richardson-accel
               (keep-halving (lambda (h) (trapezoid witch 0 1 h)) .1)
               2)
              6)

(stream-limit (richardson-accel
               (keep-halving (lambda (h) (trapezoid witch 0 1 h)) .1)
               2)
              1.e-9)

; Value: 3.1415926536207945
```

This requires only 73 evaluations of the witch to get π to 9 decimal places!