CS 450

Lecture 11: Infinite Streams

Carl D. Offner

1 Infinitely long streams

We can use streams to represent infinitely long sequences:

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

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

and then we can filter these infinite sequences, just as before:

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

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

so if we define

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

then `(stream-ref no-sevens 100)` will evaluate to 117.

Here is a neat way to produce the Fibonacci sequence:

```
(define (fibgen a b)
  (cons-stream a (fibgen b (+ a b))))

(define fibs (fibgen 0 1))
```

And here is how we get the primes, using a form of the sieve of Eratosthenes:

```
(define (sieve stream)
  (cons-stream
   (stream-car stream)
   (sieve (stream-filter
           (lambda (x)
             (not (divisible? x (stream-car stream))))
           (stream-cdr stream)))))

(define primes (sieve (integers-starting-from 2)))
```

And you can try evaluating `(stream-ref primes 50)`.

2 Defining streams implicitly

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

(define (add-streams s1 s2)
  (stream-map + s1 s2))

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

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

(define (scale-stream stream factor)
  (stream-map (lambda (x) (* x factor)) stream))

(define double (cons-stream 1 (scale-stream double 2)))

;;; prime numbers -- really clever, and efficient because it only
;;; checks divisibility by numbers less than sqrt(n).

(define primes
  (cons-stream
   2
   (stream-filter prime? (integers-starting-from 3))))

(define (prime? n)
  (define (iter ps)
    (cond ((> (square (stream-car ps)) n) #t)
          ((divisible? n (stream-car ps)) #f)
          (else (iter (stream-cdr ps)))))
  (iter primes))
```