

Streams

Jorge Ortiz
CS94SI

What are Streams?

- Streams are lazy Lists
- Like Lists, two building blocks:

List	Stream
<code>::</code>	<code>Stream.cons</code>
<code>Nil</code>	<code>Stream.empty</code>
<code>1 :: 2 :: Nil</code>	<code>Stream.cons(1, Stream.cons(2, Stream.empty))</code>

What are Streams?

- Major difference:
 - Tails of Streams are lazy

`Stream[A] ==`
`Stream.empty`
or

`Stream.cons(head: A, tail: => Stream[A])`

Advantages of Streams

- Streams allow us to represent some algorithms more efficiently than Lists

Advantages of Streams

- Second prime between 1,000 and 10,000
- List version:
 - `List.range(1000, 10000).filter(isPrime).apply(1)`
 - $O(N)$ space, $O(N)$ time
- Stream version:
 - `Stream.range(1000, 10000).filter(isPrime).apply(1)`
 - $O(1)$ space, $O(1)$ time

Advantages of Streams

- Streams allow us to represent some data structures we can't represent with Lists
- For example, infinite data structures

Infinite Streams

- How do we define infinite streams?
- Infinite recursion, lazily evaluated

```
// Returns an infinite stream of Ints
// starting from the number i
def from(i: Int): Stream[Int] =
  Stream.cons(i, from(i + 1))
```

Fibonacci Stream

- How to define:
 - `fib: Stream[Int]`

```
fib: Stream[Int] ==  
0, 1, 1, 2, 3, ...
```

Fibonacci Stream

- How to define:
 - `fib: Stream[Int]`

```
fib:          Stream[Int] ===  
0, 1, 1, 2, 3, ...
```

```
fib.tail:     Stream[Int] ===  
1, 1, 2, 3, 5, ...
```

Fibonacci Stream

- How to define:
 - `fib: Stream[Int]`

`fib: Stream[Int] ===`
`0, 1, 1, 2, 3, ...`

`fib.tail: Stream[Int] ===`
`1, 1, 2, 3, 5, ...`

`fib.tail.tail: Stream[Int] ===`
`1, 2, 3, 5, 8, ...`

Fibonacci Stream

- How to define:
 - `fib: Stream[Int]`

```
fib:          Stream[Int] ===  
0, 1, 1, 2, 3, ...
```

```
fib.tail:     Stream[Int] ===  
1, 1, 2, 3, 5, ...
```

```
fib.tail.tail: Stream[Int] ===  
plus(fib, fib.tail)
```

Fibonacci Stream

```
// Takes each element of two Streams
// and adds them together
def plus(l: Stream[Int],
        r: Stream[Int]): Stream[Int] =
  l.zip(r).map {
    case(li, ri) => li + ri
  }
```

Fibonacci Stream

- How to define:
 - `fib: Stream[Int]`

`fib: Stream[Int] ===`
`0, 1, 1, 2, 3, ...`

`fib.tail: Stream[Int] ===`
`1, 1, 2, 3, 5, ...`

`fib.tail.tail: Stream[Int] ===`
`1, 2, 3, 5, 8, ...`

Fibonacci Stream

- How to define:
 - `fib: Stream[Int]`

```
fib:          Stream[Int] ===  
  0, fib.tail
```

```
fib.tail:     Stream[Int] ===  
  1, 1, 2, 3, 5, ...
```

```
fib.tail.tail: Stream[Int] ===  
  1, 2, 3, 5, 8, ...
```

Fibonacci Stream

- How to define:
 - `fib: Stream[Int]`

```
fib:          Stream[Int] ===  
  0, fib.tail
```

```
fib.tail:    Stream[Int] ===  
  1, fib.tail.tail
```

```
fib.tail.tail: Stream[Int] ===  
  1, 2, 3, 5, 8, ...
```

Fibonacci Stream

- How to define:
 - `fib: Stream[Int]`

```
fib:          Stream[Int] ===  
  0, fib.tail
```

```
fib.tail:    Stream[Int] ===  
  1, fib.tail.tail
```

```
fib.tail.tail: Stream[Int] ===  
  plus(fib, fib.tail)
```

Fibonacci Stream

- How to define:
 - `fib: Stream[Int]`

```
val fib: Stream[Int] =  
  Stream.cons(0,  
    Stream.cons(1,  
      plus(fib, fib.tail)))
```

Stream of Primes

- Sieve of Eratosthenes

Stream of Primes

- Start with a known prime (2) and every number after it

2, 3, 4, 5, 6, 7, 8, 9 ...

Stream of Primes

- Cross out every multiple of the known prime

2, 3, X, 5, X, 7, X, 9 ...

Stream of Primes

- The next number in the list is also a prime

2, 3, X, 5, X, 7, X, 9 ...

Stream of Primes

- Repeat

2, 3, X, 5, X, 7, X, X ...

Stream of Primes

```
def sieve(s: Stream[Int]): Stream[Int] =  
  Stream.cons(s.head,  
    sieve(s.tail.filter(_ % s.head != 0)))  
  
val primes = sieve(from(2))
```