WELLESLEY

Programming Languages

CS 251

Fall 2021

Carolyn Anderson

Recap

Evaluation Strategies

Eager evaluation: evaluate expressions as soon as possible.

Lazy evaluation: evaluate expressions only when they are needed.

Streams

the power of laziness

Streams

As we saw last week, lazy evaluation allows us to define infinitely long collections of data.
These are called **streams**.

Streams

A **stream** is a sequence of data elements that are made available over time.

To define a stream, you need to know how to get the next item in the sequence, but you don't need to be able to enumerate all items.

Streams

For instance, we can define a stream of (positive) integers:

```
> (define (integers n)
      (cons n (integers (+ n 1))))

> (integers 0)
(cons 0 (cons 1 (cons 2 (cons 3 …
```

Exercise: endless string list

Exercise: write a function that takes a single string as an argument and creates an endless list of that string.

Call it ***endless-strings***.

Streams

```
> (define (endless-strings n)
        (cons str (endless-strings str)))

> (endless-strings “cat”)
(cons “cat” (cons “cat” (cons “cat” …
```

Streams

We were able to define streams because we were using lazy evaluation. If we tried to use these functions in ordinary Racket, they would never terminate.

However, there is an implementation of streams in base Racket as well.

You can load it using (require racket/stream)

Streams in Racket

We can define new streams using the (stream-cons) function.

This is equivalent to **cons** in the lazy version of Racket that we used last week.

```
(define (plus-1 n)
  (stream-cons n (plus-1 (+ n 1))))
```

Exercise: stream of longer strings

Define a stream where the first element is a string and each successive element contains one more copy of that string than the previous element:

“a”, “aa” “aaa”, “aaaa”, “aaaaa” , “aaaaaa” …

Higher-Order Functions on Streams

Map and **filter** are well-defined on streams, but **fold** only terminates on finite streams.

Map takes a stream and returns a new stream.
Just like in **map** for lists, there is a 1-to-1 correspondence between the items of the original stream and the return stream.

Higher-Order Functions on Streams

Map and **filter** are well-defined on streams, but **fold** only terminates on finite streams.

Filter takes a stream and function **f** and returns the stream consisting of all items that satisfy the function **f**.

It is well-defined if there is at least one item in the stream that satisfies **f**.

Exercise: factorial stream

Define the stream of factorials starting with 1:

1, 2, 6, 24, 92 , …