



# CS 3110

## Streams and Laziness

Prof. Clarkson  
Fall 2019

Today's music: "Lazy Days" by Enya

# **CLICKER QUESTION 1**

# Review

**Final unit of course:** Advanced functional programming

**Today:**

- Streams
- Laziness

# RECURSIVE VALUES

# “Infinite” lists

How can an infinite length list fit in a finite computer memory?



## **CLICKER QUESTION 2**

aka **infinite lists**, sequences, delayed lists, lazy lists

# **STREAMS**

# List representation

```
(** An ['a mylist] is a finite  
   list of values of type  
   ['a]. *)
```

```
type 'a mylist =  
  | Nil  
  | Cons of 'a * 'a mylist
```



# Stream representation?

```
(** An ['a stream] is an infinite  
   list of values of type  
   ['a]. *)
```

```
type 'a stream =  
  | Nil  
  | Cons of 'a * 'a stream
```

# Stream representation?

```
(** An ['a stream] is an infinite  
   list of values of type  
   ['a]. *)
```

```
type 'a stream =  
  | Nil  
  | Cons of 'a * 'a stream
```

# Stream representation?

```
type 'a stream =  
  | Cons of 'a * 'a stream
```

**Let's try coding these:**

- the stream of 1's
- the stream of natural numbers

Key idea of this entire lecture:

Be lazy:  
delay evaluation

# thunk

**fun () -> (\* a delayed computation \*)**

# Stream representation

(\*\* An ['a stream] is an infinite list  
of values of type ['a].

AF: [Cons (x, f)] is the stream  
whose head is [x] and tail is  
[f()].

RI: none \*)

**type** 'a stream =

Cons **of** 'a \* (**unit** -> 'a stream)

# Notation

Write

$\langle a; b; c; \dots \rangle$

to mean stream whose first elements are a, b, c.

# Stream sum

```
(** [sum <a1; a2; ...> <b1; b2; ...>]  
    is [<a1 + b1; a2 + b2; ...>] *)
```

```
let rec sum
```

```
  (Cons (h_a, tf_a))
```

```
  (Cons (h_b, tf_b))
```

```
=
```

```
?
```



**LAZINESS**

Demo

# Lazy

- Syntax: **lazy** e
- Static semantics:  
if  $e : t$  then **lazy** e : t lazy\_t
- Dynamic semantics:
  - **lazy** e evaluates to a *delayed value*
  - does not evaluate e to a value yet
  - when forced for the first time, evaluates e to a value v
  - if forced again, return v without evaluating e again

# Lazy

Standard library module for

- delaying evaluation
- remembering results once computed

```
module Lazy :  
  sig  
    type 'a t = 'a lazy_t  
    val force : 'a t -> 'a  
  end
```

Type constructor  
[lazy\_t] is built-in to  
language

# Implementing Lazy

- **force**: can implement yourself with references
- **lazy**: can't implement yourself

# Streams and laziness

- Can implement streams with **Lazy**
- See textbook exercise **lazy stream**

# Upcoming events

- N/A

*This is happily lazy.*

**THIS IS 3110**