CS 320: Streams

Due: Thursday, December 12, at 11:59 pm

1 Overview

The solution to this assignment has to be in Ocaml.
In this assignment you are asked to implement a few streams and functions over streams.
You are provided with the file ***streams.ml*** which you can use as a basis for your development. In the file you are already given a type definition of a stream:

```
type 'a str = Cons of 'a * ('a stream)
   and
'a stream = unit -> 'a str
```

as well as the definition of basic combinators on streams whose types signatures are:

```
val head : 'a stream -> 'a
val tail : 'a stream -> 'a stream
val take : int -> 'a stream -> 'a list
val map : ('a -> 'b) -> 'a stream -> 'b stream
val zip : ('a -> 'b -> 'c) -> 'a stream -> 'b stream -> 'c stream
val filter : 'a stream -> ('a -> bool) -> 'a stream
val sieve : int stream -> int stream
val nums_from : int -> int stream
val nats : int stream
val primes : int stream
val fibs : int stream
val ones : int stream
val zeros : int stream
val even : int -> bool
val odd : int -> bool
```

The previous functions can be helpful to implement your solution. You may also find helpful the combinators `List.fold_left`, `List.filter`, and `List.rev` from the standard library.

2 Tasks specifications

1. (5 points). `odds: int stream`

   The first task you need to solve is to define the stream of odd natural numbers in increasing order. This value will have type `int stream`.
   For example: `odds = (1, 3, 5, 7, 9, ...)`

2. (5 points). `evens: int stream`

   As second task you need to define the stream of all the even natural numbers in increasing order. This value will have type `int stream`, remember that 0 is an even number.
   For example: `evens = (0, 2, 4, 6, 8, ...)`

3. (5 points). `squares : int stream`

   The third task you need to solve is to define the stream of perfect squares of natural numbers starting from 0 in increasing order. This value will have type `int stream`.
   For example: `squares = (0, 1, 4, 9, 16, ...)`

4. (5 points). `evenFibs : int stream`

   The fourth task you need to solve is to define the stream of even Fibonacci numbers in increasing order. This value will have type `int stream`.
   For example: `evenFibs = (0, 2, 8, 34, 144, ...)`

5. (5 points). `mulThrees: int stream`

   The fifth task you need to solve is to define the stream of multiples of three of natural numbers starting from 0 in increasing order. This value will have type `int stream`.
   For example: `mulThrees = (0, 3, 6, 9, 12, ...)`

6. (5 points). Define a function

   `split_on_p: 'a stream -> ('a -> bool) -> ('a stream) * ('a stream)`

   that takes as inputs a stream of elements of type `'a` and a predicate `p:'a -> bool` over `'a` and returns a pair of streams where the first stream contains the elements of the input stream that satisfy `p`, while the second stream contains the elements of the input stream that do not satisfy `p`. For instance, given in input the streams of natural numbers, and the predicate `even`, then `split_on_p` should return a pair of streams `(evens,odds)` where `evens` is the stream of even numbers and `odds` is the stream of odd numbers. That is `((0, 2, 4, 6, 8, 10, 12, ...), (1, 3, 5, 7, 9, 11, 13, ...))`.
   For example:
   `split_on_p nats even = ((0, 2, 4, 6, 8, 10, ...), (1, 3, 5, 7, 9, 11, ...))`
   `split_on_p nats odd = ((1, 3, 5, 7, 9, 11, ...), (0, 2, 4, 6, 8, 10, ...))`

7. (5 points). Define a function

   `stream_zip: 'a stream -> 'b stream -> ('a * 'b) stream`

   that takes as inputs two streams of elements and returns the stream of pairs of elements. For instance, given in input the streams of natural numbers greater than 0, `tail nats` and the streams `primes`, then `stream_zip (tail nats) primes` should return a stream of pairs `(i, pi)` where `pi` is the `i`-th prime number, for `i>0`. That is `(1, 2),(2, 3),(3, 5),(4, 7),...`.
   For example:
   `stream_zip (tail nats) primes = (1, 2), (2, 3), (3, 5), (4, 7), (5, 11)...`
   `stream_zip odds evens = (1, 0), (3, 2), (5, 4), (7, 6)...`

8. (5 points). Define a function

   `stream_unzip: ('a * 'b) stream -> ('a stream * 'b stream)`

   that takes as inputs a stream of pairs of elements of type `'a * 'b`, and produces a pair of streams, the first containing only elements of type `'a` and the second containing only elements

of type `'b`. For instance, given in input the stream of pairs of elements of int where the `i`th element in this stream is `(i, pi)` (`i` is `i`th natural number greater than `0`, and `pi` is the `i`th `prime` number, `(1, 2),(2, 3),(3, 5),(4, 7),(5, 11), ...`), then `stream_unzip` should return a pair of streams where the first stream is the natural numbers greater than `0` and the second stream is the `primes`. That is `((1, 2, 3, 4, 5, ...), (2, 3, 5, 7, 11, ...))`.
For example:

```
stream_unzip (stream_zip (tail nats) primes) = ((1,2,3,4,5,...), (2,3,5,7,11,...))
stream_unzip (stream_zip odds evens) = ((1, 3, 5, 7, 9, ...), (0, 2, 4, 6, 8, ...))
```

9. (5 points). Define a function

   `incremental_map : ('a -> 'a ) -> a' -> 'a stream`,

   that given in input a function `f`, and an element `a` of type `'a` produces the stream of elements `a, (f a), (f (f a)), (f (f (f a))), (f (f (f (f a))))...` . For instance, if `f` is the function that maps x to $x + 1$. Then `incremental_map f 0` should return the stream of ordered natural numbers.
   For example:

   ```
   incremental_map (fun x -> x + 1) 0 = (0, 1, 2, 3, 4, 5, 6, 7, ...)
   incremental_map (fun x -> x * 2) 1 = (1, 2, 4, 8, 16, 32, 64, ...)
   ```

10. (5 points). `exp_seq: float stream`

    Use `incremental_map` to define a stream of `floats` called `exp_seq` in which the $i-$th element is the $(i-1)$-th element squared, and where the first and second elements are `2, 4`. This stream can be seen as the sequence of values of the mathematical sequence $n \mapsto 2^{2^n}$, for $n = 0, 1, 2 \ldots$
    For example: `exp_seq = (2,4,16,256,65536,...)`

3 Submission Instructions

Late submissions will not be accepted. You can use Gradescope to confirm your program adheres to the specification outlined. Only your last Gradescope submission will be graded for you final grade. This means you can submit as many times as you want. If you have any questions please ask well before the due date.

3.1 Gradescope submission instructions

When you log into Gradescope and pick the CS 320 course you will see an assignment called stream. Stream has only one part for OCaml. You will need to submit a solution to this part to have full credit for the assignment. The total of points for this assignment is: 50. Click Submit and choose your source file. **The solution you submitted must be in an .ml file named `streams.ml`**. You can submit as many times as you wish before the deadline.

3.2 Additional Resources

For information on how to run OCaml please see the following references:

- `https://caml.inria.fr/pub/docs/manual-ocaml/stdlib.html`
- `https://caml.inria.fr/pub/docs/manual-ocaml/`

You will find resources for these languages on the Piazza course web site, under the Resources page.