## Contents

1. **Preamble**  
   2

2. **Signature**  
   3

3. **Documentation**  
   
   3.1 Lazy Stream Delay and Exposure  
       5
   3.2 Stream Construction  
       6
   3.3 Deconstructing a Stream  
       7
   3.4 Simple Transformations  
       8
   3.5 Combinators and Higher-Order Functions  
       9
1 Preamble

Streams are potentially infinite lists whose elements are computed lazily. This means that the elements of a stream are determined by suspended computations that generate the next element only when forced to do so. We achieve lazy evaluation using “thunks,” which are function values whose bodies only evaluate when applied to the argument unit.

The type ′a Stream.stream represents a suspension of the type ′a Stream.front, through which the elements of the stream can be exposed.

Unless noted otherwise, all functions in this library are maximally lazy, meaning their values are not computed until absolutely necessary.
2 Signature

signature STREAM =

sig
  type 'a stream (* abstract *)
  datatype 'a front = Empty | Cons of 'a * 'a stream

  exception EmptyStream

  (* Lazy Stream Delay and Exposure *)

  val delay : (unit -> 'a front) -> 'a stream
  val expose : 'a stream -> 'a front

  (* Stream Construction *)

  val empty : 'a stream
  val cons : 'a * 'a stream -> 'a stream
  val fromList : 'a list -> 'a stream
  val tabulate : (int -> 'a) -> 'a stream

  (* Deconstructing a Stream *)

  val null : 'a stream -> bool
  val hd : 'a stream -> 'a
  val take : 'a stream * int -> 'a list
  val toList : 'a stream -> 'a list

  (* Simple Transformations *)

  val tl : 'a stream -> 'a stream
  val drop : 'a stream * int -> 'a stream
  val append : 'a stream * 'a stream -> 'a stream

  (* Combinators and Higher-Order Functions *)

  val map : ('a -> 'b) -> 'a stream -> 'b stream
  val filter : ('a -> bool) -> 'a stream -> 'a stream
  val zip : 'a stream * 'b stream -> ('a * 'b) stream
end
3 Documentation

**Constraint:** Whenever you use these stream functions, please make sure you meet the specified preconditions.

If you do not meet the precondition for a function, it may not behave as expected.
3.1 Lazy Stream Delay and Exposure

\begin{verbatim}
delay : (unit -> 'a front) -> 'a stream
ENSURES: delay f evaluates to a stream version of the suspended front.
\end{verbatim}

\begin{verbatim}
expose : 'a stream -> 'a front
ENSURES: expose s evaluates to a front version of the stream.
\end{verbatim}

Note that expose (delay f) \(\Rightarrow f\).
### 3.2 Stream Construction

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>empty : 'a stream</code></td>
<td>ENSURES: <code>empty</code> is a stream with no elements.</td>
</tr>
<tr>
<td><code>cons : 'a * 'a stream -&gt; 'a stream</code></td>
<td>ENSURES: <code>cons x s</code> evaluates to a stream whose first item is <code>x</code> and whose remaining items are exactly the stream <code>s</code>.</td>
</tr>
<tr>
<td><code>fromList : 'a list -&gt; 'a stream</code></td>
<td>ENSURES: <code>fromList L</code> returns a stream consisting of the elements of <code>L</code>, preserving order. This function is intended primarily for debugging purposes.</td>
</tr>
<tr>
<td><code>tabulate : (int -&gt; 'a) -&gt; 'a stream</code></td>
<td>ENSURES: <code>tabulate f n</code> evaluates to a stream <code>s</code> of length <code>n</code>, where the <code>i</code>(^{th}) element of <code>s</code> is equal to <code>f i</code>. Note that indices are zero-indexed.</td>
</tr>
</tbody>
</table>
### 3.3 Deconstructing a Stream

<table>
<thead>
<tr>
<th>Function</th>
<th>Signature</th>
<th>Ensures</th>
</tr>
</thead>
<tbody>
<tr>
<td>null</td>
<td>`'a stream -&gt; bool</td>
<td><code>null s</code> evaluates to <code>true</code> if <code>s</code> is an empty stream, and <code>false</code> otherwise.</td>
</tr>
<tr>
<td>hd</td>
<td>`'a stream -&gt; 'a</td>
<td><code>hd s</code> evaluates to the first element of <code>s</code> if <code>s</code> is non-empty. <code>hd s</code> raises <code>EmptyStream</code> otherwise.</td>
</tr>
<tr>
<td>take</td>
<td>`'a stream * int -&gt; 'a list</td>
<td><code>take (s, i)</code> evaluates to the list containing exactly the first <code>i</code> elements of <code>s</code> if <code>s</code> contains at least <code>i</code> elements, and raises <code>Subscript</code> otherwise.</td>
</tr>
<tr>
<td>toList</td>
<td>`'a stream -&gt; 'a list</td>
<td><code>toList s</code> returns a list consisting of the elements of <code>s</code>, preserving order. This function will loop forever if <code>s</code> is an infinite stream. This function is intended primarily for debugging purposes.</td>
</tr>
</tbody>
</table>
3.4 Simple Transformations

\[ \text{tl : } 'a \text{ stream} \rightarrow 'a \text{ stream} \]

ENSURES: \( \text{tl } s \) evaluates to the stream containing all but the first element of \( s \) if \( s \) is non-empty. \( \text{tl } s \) raises \texttt{EmptyStream} otherwise.

\[ \text{drop : } 'a \text{ stream} * \text{ int} \rightarrow 'a \text{ stream} \]

REQUIRES: \( i \geq 0 \)

ENSURES: \( \text{drop } (s, i) \) evaluates to the stream containing all but the first \( i \) elements of \( s \) if \( s \) contains at least \( i \) elements, and raises \texttt{Subscript} otherwise.

\[ \text{append : } 'a \text{ stream} * 'a \text{ stream} \rightarrow 'a \text{ stream} \]

ENSURES: \( \text{append } (s1, s2) \Rightarrow s \), where \( s \) is \( s2 \) appended to \( s1 \).
3.5 Combinators and Higher-Order Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>map</td>
<td>('a -&gt; 'b) -&gt; 'a stream -&gt; 'b stream</td>
<td>Ensures: map f s (\Rightarrow s'), where each element (x) in (s) corresponds to (f \ x) in (s').</td>
</tr>
<tr>
<td>filter</td>
<td>('a -&gt; bool) -&gt; 'a stream -&gt; 'a stream</td>
<td>Ensures: filter (p) (s) evaluates to a stream containing all of the elements (x) of (s) such that (p \ x) (\Rightarrow) true, preserving element order.</td>
</tr>
<tr>
<td>zip</td>
<td>'a stream * 'b stream -&gt; ('a * 'b) stream</td>
<td>Ensures: zip ((s1, s2)) (\Rightarrow s'), where the (i^{th}) element of (s') is the pair of the (i^{th}) element of (s1) and the (i^{th}) element of (s2). (s') contains as many elements as the stream ((s1\ or\ s2)) with fewer elements.</td>
</tr>
</tbody>
</table>