Computer Science 131, Spring 2001
Assignment 11: Streams

Out: Wednesday, April 18, 2001

Due: Wednesday, April 25, 11:00am

For this assignment, retrieve `stream.sml` and `assign11.sml` from the Assignments web page. (No special version of the compiler is required; just `use` the file `stream.sml`.)

Complete the functions in `assign11.sml`. As always, your file should contain no syntax or type errors. When you are ready, run the command

```
cs131submit assign11.sml
```

In all cases, your code should be as lazy as possible. Delay computations whenever feasible.

1 Tree Comparisons

Consider the following representation for binary trees containing an integer at each leaf:

```
datatype itree = Leaf of int
               | Node if itree*itree
```

1. Write a function

   ```
   sappend : 'a stream * 'a stream -> 'a stream
   ```

   to append two streams into a single stream: all the elements of the first stream, followed by all the elements of the second stream.. (If the first stream turns out to be infinite then the values of the second stream will never be used.)

2. Write the function

   ```
   fringe : itree -> int stream
   ```

   which creates a stream of all the integers in the given tree from left to right. Remember to make your code lazy!

3. Write the function

   ```
   samefringe : itree * itree -> bool
   ```

   to test whether the two ordered trees contain the same set of integers. Your code should halt immediately when a difference is detected without entirely traversing either tree.

2 Hamming's Problem

1. Write the function

   ```
   sscale : int -> int stream -> int stream
   ```

   such that `sscale n` takes a stream of integers and multiplies them all by `n`. Thus if `nats` is the stream of natural numbers defined in class, then `sscale 5 nats` would return a stream containing all multiples of `5`.

2. Write the function

   ```
   smerge : int stream * int stream -> int stream
   ```

   which takes two strictly increasing sequences of integers and merges them into a single increasing stream of integers *with no duplicates*. If a number appears in both input streams, it should appear exactly once in the output stream.

3. Define the stream

   ```
   seq235 : int stream
   ```

   which contains the elements of the set $\{2^i 3^j 5^k \mid i, j, k \geq 0\}$ without repetitions, sorted in increasing order. The following observations may be useful:

   - The sequence begins with `1`.
   - The elements of `sscale 2 seq235` form a subsequence of `seq235`.
   - So do the elements of `sscale 3 seq235` and `sscale 5 seq235`.
   - Every element in `seq235` is either `1` or a member of at least one of these subsequences (because every element is either `1`, or divisible by `2` or `3` or `5`).

3 Digital Logic

1. Define the function

```
map2 : ('a*'b->'c) -> ('a stream * 'b stream) -> 'c stream
```

   that returns returns a stream of values computed by applying the given function to pairs of elements from the two given *infinite* streams. Remember to make the code as lazy as possible.

2. We can represent the state of a digital circuit at discrete time intervals as an infinite stream of booleans. For example, an infinite sequence containing only `true` would represent a signal that was always active, and stream that repeats the sequence `true, false, false, false` could describe a circuit clock output that goes high once every four time intervals.

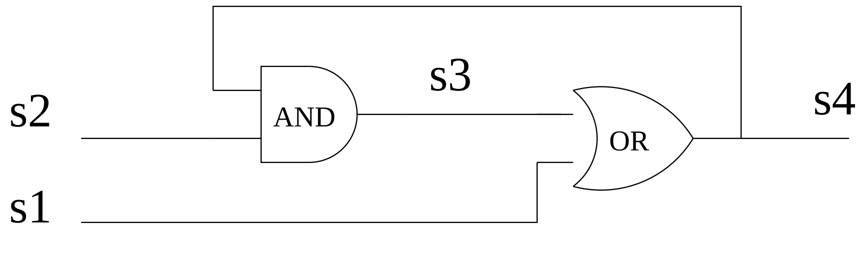
   Using `map2`, define the functions

```
andgate : bool stream * bool stream -> bool stream
orgate  : bool stream * bool stream -> bool stream
```

   These functions should should have the following behavior:

   - At the first instant, these gates should output `false`
   - At each successive instant, the gates should output the logical-and or logical-or, respectively, of the inputs at the previous time instant. That is, the gate's outputs lag the inputs by one time interval.

3. Finally, implement a function to simulate the following circuit:



   Your function

```
circuit : bool stream * bool stream -> bool stream
```

   Should take the sequence of inputs `s1` and `s2`, and yield the sequence `s4` of outputs from the OR gate. Note that this circuit contains a feedback loop: the output of the OR gate is also an input to the AND gate, while the output of the AND gate is an input to the OR gate.