CSE 305: Infinite Lists

Lukasz Ziarek

Due: May 12th, at 11:59 pm

1 Overview

In this homework you will be exploring infinite lists in SML. This homework is divided into 3 parts, each of which has an associated point total. The total of all parts combines to 50 points.
Part 1: Infinite Lists (30 Points)
Part 2: Generalized Printing (10 Points)
Part 3: ZIP and Evaluation (10 Points)

Note: Please adhere to the type signatures that we request you to follow as you will be graded exactly based on them. If you do not follow the type signatures you will be assigned 0 points.

2 Requirements and Submission

Submissions must be made on Autolab. You do not need to worry about the file name if you are only submitting a single SML program, the autograder will handle that. If you have multiple files, you may zip or tar them and submit the archive to Autolab, but there must be a file named `inf_student.sml` as the main SML program. Do not put any required files in folders, otherwise your submission will be assigned 0 points.

For generalized printing, your program must open an outstream to a file and close it after the list has been printed. Otherwise, you will receive no points for the section.

Your submission must include the contents of `inf_student.sml` provided on Piazza.

Autolab will grade your submission as follows:

Part 1 - Checks for correct `fibs/evenFibs/oddFibs`. Your submission does not need to print them. You do not need the functionality of Part 2 or Part 3 to receive credit.

Part 2 - `printList` and `printPairList` must open an outstream, print to the outstream, and then close the outstream. You do not need to print to terminal. Autolab will check the file for correct output. You do not need the functionality of Part 1 or Part 3 to get credit.

Part 3 - Checks for a correct `ZIP` function. You do not need the functionality of Part 2 to get credit for Part 3.

3 Infinite Lists

Start with the following datatype and function bindings given in `inf_student.sml` under the General Resources tab in Piazza:

```
datatype 'a inflist = NIL
   | CONS of 'a * (unit -> 'a inflist);
exception Empty;
exception Subscript;
fun HD (CONS(a,b)) = a
   | HD NIL = raise Empty;
fun TL (CONS(a,b)) = b()
   | TL NIL = raise Empty;
fun NUL NIL = true
   | NUL _ = false;
fun NTH 0 L = HD L
   | NTH n L = NTH (n-1) (TL L);
fun TAKE (xs, 0) = []
   | TAKE (NIL, n) = raise Subscript
   | TAKE (CONS(x, xf), n) = x::TAKE(xf(), n-1);
fun FROMN n = CONS(n, fn () => FROMN (n+1));
fun FIB n m = CONS(m, fn () => FIB m (n+m));
fun FILTER f l =
   if NUL l
   then NIL
   else if f (HD l)
   then CONS(HD l, fn() => (FILTER f (TL l)))
   else FILTER f (TL l);
fun SIFT NIL = NIL
   | SIFT l =
   let val a = HD l
   in CONS(a, fn () => SIFT(FILTER (fn x => x mod a <> 0) (TL l)))
end;
```

You are tasked with creating 3 infinite lists, as well as appropriate helper functions with the following type signatures:

```
val even = fn :  int -> bool
val odd = fn :  int -> bool
val fibs = CONS (0,fn) :  int inflist
val evenFibs = CONS (0,fn) :  int inflist
val oddFibs = CONS (1,fn) :  int inflist
```

Write a function called `even` which will test the argument if it is even. Write a function called `odd` which will test the argument if it is odd. Write a function call `fib` that will generate an infinite list containing the Fibonacci sequence (hint: take a look at the FROMN function in the class slides as an example of a generating function). `fib` should be a curried function that takes two arguments, the first two Fibonacci numbers (ex: fib 0 1) Create an infinite list called `fibs` that has the entire Fibonacci sequence. Create an infinite list called `evenFibs` that has only the even numbers in the entire Fibonacci sequence. Create an infinite list called `oddFibs` that has only the odd numbers in the entire Fibonacci sequence. You should use the FILTER function and the infinite sequence `fibs` to create `evenFibs` and `oddFibs` by filtering the odd and even Fibonacci numbers respectively.

4 Generalized Printing

You are tasked with creating generalized list printing functions. Write a function called `printGenList` which takes a function f and a list l and applies the function f to each element of the list l recursively. The function should have the following type signature:
`val printGenList = fn : ('a -> 'b) -> 'a list -> unit`
Create a function called `printList` that will pretty print an integer list. You will find the string concatenation operator in SML useful (^).

Example: `print("hello" ^ " world");`

`printList` will take in a 2-tuple, the first is the name of a file where the output is to be printed, and the second is the list to print. This function should leverage `printGenList` and provide an anonymous function (the fn ... =>... construct) that will do the appropriate pretty printing to the output file. This anonymous function should print the element of the list and then a space character. printList should have the following type signature:
`val printList = fn : (string * int list) -> unit`

Create a function called `printPairList` that will pretty print a list consisting of integer pairs. The function will also take in a 2-tuple, the first is the name of a file where the output is to be printed, and the second is the list to print. `printPairList` should leverage `printGenList` and provide an anonymous function (the fn ... =>... construct) that will do the appropriate pretty printing. This anonymous function should print an open parenthesis the first element of the pair, a comma, a space, the second element of the pair, and then a close parenthesis followed by a space. `printPairList` should have the following type signature:
`val printPairList = fn : (string * (int * int) list) -> unit`

For both `printList` and `printPairList`, you may want to use SML's `let-in-end` construct. Open the file in the `let` declarations, and in the `in` expression, you would print the list to the file and close the outstream. (Hint: remember that in parenthesized expressions, `(<expr>)`, you can separate multiple expressions with a semicolon, `(<expr1>; <expr2>)`. The first expression can print the list, the second can close the outstream.)

5 ZIP and Evaluation

Write a function called `ZIP` which will zip together two infinite lists (hint: take a look at how the zip function works that we went over in class). ZIP should have the following type signature:
`val ZIP = fn : 'a inflist * 'b inflist -> ('a * 'b) inflist`
You may wish to test this by using the functions from part 2 and 3 to print the first 20 Fibonacci numbers from the infinite list `fibs` by zipping the first 10 even Fibonacci numbers from the infinite list `evenFibs` with the first 10 odd Fibonacci numbers from the infinite list `oddFibs`. Additionally, test by printing the first 10 pairs from an infinite list created by zipping `evenFibs` and `oddFibs`. You should use the `TAKE` function to get the required amount of elements from each infinite list. Autolab will be leveraging your ZIP function for grading; you do not need to invoke the function. As long as the behavior of the function is correct, you will be awarded points.

6 Example output for the entire homework

```
use "inf_student.sml";
[opening inf_student.sml]
datatype 'a inflist = CONS of 'a * (unit -> 'a inflist) | NIL
val HD = fn :  'a inflist -> 'a
val TL = fn :  'a inflist -> 'a inflist
val NULL = fn :  'a inflist -> bool
val FILTER = fn :  ('a -> bool) -> 'a inflist -> 'a inflist
val TAKE = fn :  'a inflist * int -> 'a list
val even = fn :  int -> bool
val odd = fn :  int -> bool
val fib = fn :  int -> int -> int inflist
val fibs = CONS (0,fn) :  int inflist
val evenFibs = CONS (0 ,fn) :  int inflist
val oddFibs = CONS (1,fn) :  int inflist
val printGenList = fn :  ('a -> 'b) -> 'a list -> unit
val printList = fn :  (string * int list) -> unit
val printPairList = fn :  (string * (int * int) list) -> unit
val ZIP = fn :  'a inflist * 'b inflist -> ('a * 'b) inflist

EXAMPLE fibs OUTPUT: 0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181
EXAMPLE evenFibs OUTPUT: 0 2 8 34 144 610 2584 10946 46368 196418
EXAMPLE oddFibs OUTPUT: 1 1 3 5 13 21 55 89 233 377
EXAMPLE ZIP OUTPUT: (0, 1) (2, 1) (8, 3) (34, 5) (144, 13) (610, 21) (2584, 55) (10946,
89) (46368, 233) (196418, 377)
```