COMP2212 Programming Language Concepts Coursework

Julian Rathke

Semester 2 2019/20

Introduction

Streams are potentially unbounded sequences of data. They are common in mathematics, computer science and everyday life. Examples of streams include *streaming* a movie on the web, a sequence of bits flowing through a wire in a digital circuit, or temperature measurements outside of Building 53 taken every hour. The theory of (discrete) signal processing deals with infinite sequences of reals. Similarly, in computer science, dataflow is the study of networks of nodes and channels through which a potentially unbounded number of data elements flow. A particular feature of streamed data is that streams may have different rates of accessing data associated with them.

Your task is to *design and implement* a domain specific programming language that performs certain simple computations on potentially unbounded integer sequences. You are required to

1. Invent an appropriate syntax,
2. Write an interpreter for your language using Haskell, and
3. Document your language in a written report.

The coursework will consist of two submissions. More details, including advice and deadlines can be found at the end of this document.

Submission 1

For the five example problems listed below, you will be required to produce a program, in your language, that solves each of the problems. Therefore, your design should be guided by the five problems, since you are required to produce *source files in your own language* that solve them.

The first submission consists of the source for your interpreter together with solutions written in your language to the first five problems.

Submission 2

Please keep a record of all the sources you consult that influence your design, and include them in your programming language report. The report will be required for the second submission, along with programs for five additional *unseen* problems, which we will make public *after* the deadline for the first submission.

The five additional problems will not ask you to reinvent the wheel. Instead, they will consist of variations and combinations of the kinds of computations that appear in the first five problems. The idea is to test the *orthogonality* of your design.

The specification is deliberately loose. If we haven't specified something, it is a design decision for you: e.g. how to handle syntax errors, illegal inputs, whether to allow comments, support for syntax highlighting, compile time warnings, any type systems etc. A significant part of the mark (25%) will be awarded for these qualitative aspects, where we will also take into account the design of the syntax and reward particularly creative, clean and concise solutions with additional marks. The remaining 75% of the mark will be awarded for solving the problems correctly.

First Five Problems

Input and output conventions

We will model streams using sequences of integers and, in general, inputs will take the following form:

```
a11 a21 a31 a41 ... an1
a12 a22 a32 a42 ... an2
.
.
.
a1k a2k a3k a4k ... ank
.
.
.
```

where every line is terminated with the new line character `EOL` and the file ends with `EOF`. The as are arbitrary (positive or negative) 32 bit integers (you can safely use a 64 bit implementation, we will not test for architecture specific overflow behaviour). The number n is the number of input sequences, which are the columns in the input. You can assume that all of the input streams will have the same length, or in other words, that all of the columns will be of the same size. This length will be finite in all tests but may be arbitrarily large. Note that empty inputs *are* allowed: if you are expecting n sequences and are given an empty input, you should assume that your input is n empty sequences.

The values in each row are separated by a single space character. For example, the following is an input that describes 2 sequences of length 3:

```
2 1
3 −1
4 1
```

For each problem you are expected to produce a single output on `stdout`. Your output is expected be of the form:

```
o1
o2
o3
.
.
.
oN
```

where the length of the output—that is, the number of lines—will depend on the length of the input. Where the output depends on an input stream that has exhausted its input then no further outputs are required. Every line must be terminated with the Unix new line character `EOL` (i.e. **not** Windows `CR+LF`).

Problem 1 - Double Speed Shuffle

Take two sequences $a_1\ a_2\ a_3\ a_4\ a_5\ \ldots$ and $b_1\ b_2\ b_3\ b_4\ \ldots$ as an input and output the sequence $a_1\ \ a_2\ \ b_1\ \ a_3\ \ a_4\ \ b_2\ \ \ldots$, that is, the output sequence alternates between input sequences, but reads twice from input one for ever read from input two.

Example input:	Expected output:
	1
1 5	2
2 6	5
3 7	3
4 8	4
	6

Note that input values 7,8 in the second input stream are never consumed as the first input stream has been exhausted.

Problem 2 - Shuffling and Arithmetic

Take three sequences $a_1\ a_2\ a_3\ a_4\ a_5\ \ldots$, $b_1\ b_2\ b_3\ b_4\ b_5\ \ldots$ and $c_1\ c_2\ c_3\ c_4\ c_5\ \ldots$ as inputs and output the following sequence

$$c_1 \quad b_1 \quad a_1 \quad a_1 + b_1 \quad b_1 + c_1 \quad c_2 \quad b_2 \quad a_2 \quad a_2 + b_2 \quad b_2 + c_2 \quad \ldots$$

This is the values from input stream three, then two then one, followed by the sum of the same values read from streams one and two and finally the sum of the values read from streams two and three. This is then repeated for the next entries in the input streams.

Example input:	Expected output:
	8
	2
	1
	10
	3
	−4
1 2 8	7
0 7 −4	0
3 5 −2	3
	7
	−2
	5
	3
	3
	8

Problem 3 - Skip and Prefix

Take two sequences $a_1\ a_2\ a_3\ a_4\ \ldots$ and $b_1\ b_2\ b_3\ b_4\ \ldots$, and produce the sequence

$$0 \quad b_2 \quad a_1 \quad b_4 \quad a_2 \quad b_6 \quad a_3 \quad b_8 \quad \ldots$$

That is a shuffle of the first input stream prefixed by a fixed value 0, and the second input stream where every other value is skipped over.

Example input:	Expected output:
1 5	0
2 4	4
3 3	1
4 2	2
5 1	2
6 0	0
	3

Problem 4 - More Stream Arithmetic and Local Reverse

Take a sequence $a_1\ a_2\ a_3\ a_4\ a_5\ a_6\ \ldots$ and produce the sequence

$$a_3 \quad 2 * a_2 \quad 3 * a_1 - 1 \quad a_6 \quad 2 * a_5 \quad 3 * a_4 - 1 \quad \ldots$$

That is at least three values are taken from the single input sequence, the third of these is output first, the second is doubled and then output, finally the first value is trebled, decremented and output. This sequence repeats.

Example input:	Expected output:
1	3
2	4
3	2
4	6
5	10
6	11

Problem 5 - Accumulator

Take a sequence $a_1\ a_2\ a_3\ a_4\ \ldots$ and output the sequence

$$a_1 \quad a_1 + a_2 \quad a_1 + a_2 + a_3 \quad a_1 + a_2 + a_3 + a_4 \quad \ldots$$

where each term of the output is the sum of all the input terms up to that point.

Example input:	Expected output:
1	1
2	3
3	6
4	10

First submission - due Monday April 20 4pm

Interpreter and the first five problems

You are required to submit a zip file containing:

- the source for the interpreter for your language, written in Haskell
- five programs, written in **YOUR** language, that solve the five problems specified above. The programs should be in files named `pr1.spl`, `pr2.spl`, `pr3.spl`, `pr4.spl`, `pr5.spl`.

Implementation and compilation. We will compile by using the command `make`, so you will need to include a Makefile in your zip file that produces an executable named `myinterpreter`. Prior to submission, you are required to make sure that your interpreter compiles **on a Unix machine with a standard installation of GHC (Version 8.4.3) or earlier**: if your code does not compile then you will be awarded 0 marks. Before submission, we will release a simple "automarking" script that will allow you to check if your code compiles and whether each of your programs passes a basic test.

You can use `Alex` and `Happy` for lexing and parsing. Alternatively you can use any other Haskell compiler construction tools such as parser combinator libraries. You are welcome to use any other Haskell libraries, as long as this is clearly acknowledged and the external code is bundled with your submission, so that it can compile on a vanilla Haskell installation.

Interpreter spec. Your interpreter is to take a file name (the program in your language) as a single command line argument. The interpreter should expect its input on standard input (`stdin`), produce output on standard output (`stdout`) and error messages on standard error (`stderr`).

For each problem, we will test whether your code performs correctly by using a number of tests. We only care about correctness and performance will not be assessed (within reason - our marking scripts will timeout after a generous period of time). You can assume that for the tests we will use correctly formatted input. For example, when assessing your solution for Problem 1 we will run

```
myinterpreter pr1.spl < input.txt
```

where `input.txt` will contain the input streams for a particular test, following the input conventions described earlier.

Second submission - due Thursday April 30th 4pm

Written Report and the second five problems

Shortly after the first deadline we will release a further five problems. Although they will be different from Problems 1-5, you can assume that they will be closely related, and follow the same input/output conventions. You will be required to submit two separate files.

First, you will need to submit a zip file containing programs (`pr6.spl`, `pr7.spl`, `pr8.spl`, `pr9.spl`, `pr10.spl`) written in your language that solve the additional problems. We will run our tests on your solutions and award marks for solving the additional problems correctly.

Second, you will be required to submit a 3 page report on your language **in pdf format** that explains the main language features, its syntax, including any scoping and lexical rules as well as additional features such as syntax sugar for programmer convenience, type checking, informative error messages, etc. In addition, the report should explain the execution model for the interpreter, e.g. what the states of the runtime are comprised of and how they are transformed during execution. This report, together with the five programs will be evaluated qualitatively and your marks will be awarded for the elegance and flexibility of your solution and the clarity of the report.

Please note: **there is only a short period between the first and second submission**. I strongly advise preparing the report well in advance throughout the development of the coursework.

The coursework is to be done in pairs. As part of the second submission we will require a declaration of how marks are to be distributed amongst the members of your group. All submissions will be done through handin. Please make sure that there is only one submission per group – there will be a marks penalty if both members submit separately.

Marks. This coursework counts for 40% of the total assessment for COMP2212. There are a total of 40 marks available. These are distributed between the two submissions as follows:

Submission one has 10 marks available. There are 2 marks available for functional correctness of each of the first five problems. You will receive the results of Submission 1 prior to the second deadline.

Submission two has 30 marks available. We will award up to 10 marks for the qualitative aspects of your solution, as described in your programming language report. We will award up to 20 marks for your solutions to the second five problems. For each problem there will be 4 marks available for functional correctness only. You have the option of resubmitting the interpreter, for a 50% penalty on this component. If you decide to resubmit your interpreter in the second submission the maximum possible total coursework mark is therefore capped at 30 marks.

Any late submission to either component will be treated as a late submission overall and will be subject to the standard university penalty of 10% per working day late.