CS 450 Homework 5

Carl Offner

Spring, 2022

Due Wednesday, March 9, 2022

These exercises concern streams and delayed evaluation, and provide some practice using them in a remarkable and non-trivial example —computing the decimal digits of π.

Part 1: Elementary computations with streams

For this first part of the assignment, you will hand in one file: **streams.scm**, in your new project directory `.../cs450/hw5`, with discussion in the same file, as Scheme comments.

This file should begin with some definitions from the textbook, which are included here. You should **copy** these definitions into the top of your file `streams.scm` (in addition to any other definitions you might find useful before starting the code for the various problems below). This is important: `streams.scm` should not load any other file—it should be a self-contained file that contains any code it needs.

Also please note: nowhere in the code for this assignment do you need to use `set!` or any other operator that changes the value of a variable. Sometimes informally, I write in the paper of "changing the value" of some variable. What that always really means is a recursive call in which the new value is passed. But no variable actually changes its value.

Please be careful about this. The whole point of this assignment is to use streams and recursion. If you read what I have written here carefully and do what I suggest, I think you will be amazed at how simple and clear the code for these procedures is.

1. Define a function

   ```
   (display-n stream n)
   ```

   that prints the first `n` elements of `stream`, each on a separate line.

2. Exercise 3.50 (page 324)

This exercise is pretty simple, provided you follow the suggestion in the book. In fact, you can forget about checking for the empty stream—we will only use this procedure for infinite streams.

3. As explained in section 3.5.2, we can define an infinite stream of ones and use this to define the stream of positive integers:

   ```
   (define ones (cons-stream 1 ones))
   ```

   ```
   (define integers (cons-stream 1 (add-streams ones integers)))
   ```

   Type in these definitions and verify that they work by using the `display-n` procedure. Generate the stream `notdiv-235` of all integers that are not divisible by any of the numbers 2, 3, or 5. (Use `stream-filter`.)

Part 2: Computing the digits in pi

There are two problems in this part of the assignment. They are both long and difficult. They are based on an expository paper that you will find on my web site: Computing the Digits in π. Actually, all you really need to read in that paper is Section 8, although you may be interested in looking at other parts as well But you will really need to read and understand all of Section 8. (Well, you don't actually have to read the two proofs in Sections 8.1.1 and 8.2.3, but you might want to at least skim those two sections to get an understanding of what is going on.)

Section 8 of the paper is adapted from the paper ``Unbounded Spigot Algorithms for the Digits of Pi", by Jeremy Gibbons, in the American Mathematical Monthly (Vol. 113, Number 4; April 2006).

As in Part 1 of the assignment, please note that nowhere in the code for this assignment do you need to use `set!` or any other operator that changes the value of a variable. Sometimes informally, I write in the paper of "changing the value" of some variable. What that always really means is a recursive call in which the new value is passed. But no variable actually changes its value.

Put your code for this part of the assignment in the file **pi.scm** in your `hw5` directory. This file should begin with the same definitions as the file `streams.scm` from Part 1. `pi.scm` should not load any other file.

Also, you should create a separate text file **notes.txt** to hold more extended discussions of the problems and the code you wrote. (Of course, the code should still be commented in place.)

1. Read all of Section 8.1 of the paper.

   1. Before actually writing any code, finish filling in the table of Figure 18. (Put your table in `notes.txt`.)

2. Produce a function `mult-stream`, based on the explanation in Section 8.1, that takes as input two arguments:

   - The first argument is a multiplier `m`, which is a positive integer.
   - The second argument is a stream `strm` of digits, representing the digits in the decimal representation of a number between 0 and 1. You may assume that the number does not end in an infinite string of 9's.

   The function `mult-stream` produces a stream which is the decimal representation of the product of `m` with the number represented by `strm`. Don't worry about where the decimal point goes for this assignment. (Of course, in practice that would be very important! It's not at all hard to figure out, but it's not what I'm concerned about right here.)

   Somewhere in the course of doing this, you will probably need a function which you might want to name

   ```
   number->list-of-digits
   ```

   which takes a non-negative integer as input and returns the list of single digits that make up the decimal representation of that integer. There are various ways to do this. One way is to use some of the built-in Scheme functions that operate on strings and characters, such as

   ```
   number->string
   string->list
   char->integer
   ```

   If you do this, please be sure to notice that `char->integer` returns the ASCII value of a character, so if the character is "3", for instance, what is returned from `char->integer` is not the number 3. So you'll have to adjust for that as well.

2. Read all of Section 8.2 of the paper.

   1. On page 54, just before Section 8.2.1, it is stated that we could have performed the sequence of computations there, but starting with a value different from 2 and still got the same result (in the limit, of course). Do the computations starting with 3 instead of 2 and show that this seems to be true. (This isn't a proof, of course, right?) Put this in `notes.txt`.

      And just to be completely clear about this: when I talk about "starting with 3 instead of 2", I'm talking about replacing only the rightmost 2 in the expressions, not all of them.

   2. Answer the following question (in `notes.txt`): What is the matrix corresponding to the fractional linear transformation that takes x as input, adds 3 to it, and then takes the reciprocal of the result?

   3. In the middle of page 55 is a statement that you are invited to try to prove. Put your proof in `notes.txt`

   4. Finally, based on what you have learned in this section, write a Scheme procedure named pi, which takes no arguments, and which returns the stream of decimal digits of π; that is, the stream (3 1 4 1 5 9 ...).

Please be careful. I want you to produce a **procedure** named pi, not a stream named pi. The procedure, when invoked, should produce a stream.

To do this, you will need to have a representation for a 2x2 matrix. A simple list of 4 elements will do. Write a constructor and selectors for this.

You will also need a procedure to multiply matrices. I suggest calling this procedure `compose`, since the term "multiply" is overloaded enough as it is.

You will need to produce the original input stream. For a hint on how to do this, look at how we produced the stream `integers` using `add-stream`. You will need to notice how each element of the stream is produced from the previous one: you just add the matrix

```
|  1  4  |
|        |
|  0  2  |
```

to each element of the stream to get the next one. (Make sure you understand this.) Just as the stream `integers` was produced from the initial stream `ones`, you can build up the stream `strm` by starting with the stream all of whose elements are this matrix, and you can construct this stream in turn the same way as the stream `ones` was constructed.

That should enable you to produce `strm` recursively. (You'll also need a procedure to add matrices. Remember that you add matrices by adding corresponding elements.)

5. In doing this assignment, you should not use floating-point computations. And in fact there is no need to, because the only thing you really need to deal with is integers. In such a case, integer operations are both faster and more accurate (because they are exact). So in particular, you don't need to use either of the functions `/` or `floor`. In fact,

   ```
   (quotient a b)
   ```

   is just the floor of `a/b`. Isn't that neat?

The file `notes.txt`

Finally, a word about your `notes.txt` file. I have asked some questions above, and I expect to see the answers in that file. However, if that is all you put in that file, I will be very disappointed. You don't want me to be disappointed. I expect that you will learn a lot by doing this assignment. I expect you to write about that in notes.txt.

Believe it or not, I have occasionally seen things like this:

"I was confused at first. But then I thought about it a lot, and now I understand it."

I hope you understand that this sort of thing is useless. It doesn't tell me or anyone else anything at all. Please spend some serious time making your `notes.txt` file informative and useful.