CS450: Structure of Higher Level Languages

Fall 2020 Assignment 5, part 2

Due: Monday, November 16

Taken from assignments by Profs. Carl Offner and Ethan Bolker

Computing the digits of π

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 here: http://www.cs.umb.edu/cs450/homwork/pi.pdf. 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, the paper mentions "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`. This file begins with the same definitions as the file `streams.scm` from Part 1. `pi.scm` should not load any other file and you should not modify the header.

Also, you should answer the questions below in a separate file to be submitted with your homework (can be the same file as part 1). (Of course, the code should still be commented in place.)

1. Read all of Section 8.1 of the paper.

   (a) Before writing any code answer the questions stated in the last section, to be put in a separate file.

   (b) 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.

Important! Even though this particular case the produced stream is finite, make your function be able to handle infinite streams as well.

2. Read all of Section 8.2 of the paper.

3. 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 four 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 $\begin{pmatrix} 1 & 4 \\ 0 & 2 \end{pmatrix}$ 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.)

   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?

Theoretical/Written questions

Finally, a word about your separate answer 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 a separate discussion.

Believe it or not, according to Prof. Offner people have written things like:

"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 answers informative and useful.

Specifically, please answer the following:

1. The completed table in Figure 18 of Prof. Offner's paper (see above).

2. 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?)

   Notice: 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.

3. 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?

4. In the middle of page 55 is a statement that you are invited to try to prove.