Unbounded Spigot Algorithms for π

Jeremy Gibbons

IIJ, March 2017

UNIVERSITY OF OXFORD

1. Spigot algorithms for π

Rabinowitz & Wagon's algorithm, obfuscated by Winter & Flammenkamp:

```
a[52514], b, c = 52514, d, e, f = 1e4, g, h;
main(){
  for(; b = c -= 14; h = printf("%04d", e + d / f))
    for(e = d %= f; g = --b * 2; d /= g)
      d = d * b + f * (h ? a[b] : f / 5), a[b] = d % --g;
}
```

based on the expansion

$$\pi = \sum_{i=0}^{\infty} \frac{(i!)^2 2^{i+1}}{(2i+1)!} = 2 + \frac{1}{3}\left(2 + \frac{2}{5}\left(2 + \frac{3}{7}\left(\cdots\left(2 + \frac{i}{2i+1}\left(\cdots\right)\right)\right)\right)\right)$$

A *spigot algorithm*: digits 'drip' out, one by one (or here, four by four), with limited intermediate storage.

2. Finite versus infinite sequences

R&W's algorithm inherently *bounded*, committing initially to length:

> "One cannot simply [keep going], because memory allocations must be made in advance".

W&F's program operates on a *finite* array, generating just 15,000 digits.

This program

$$
\begin{array}{l}
pi = g\ (1,0,1,1,3,3)\ \textbf{where} \\
\quad g\ (q,r,t,k,n,l) = \\
\quad\quad \textbf{if}\ 4 \times q + r - t < n \times t \\
\quad\quad\quad \textbf{then}\ n : g\ (10 \times q, 10 \times (r - n \times t), t, k, \\
\quad\quad\quad\quad\quad\quad div\ (10 \times (3 \times q + r))\ t - 10 \times n, l) \\
\quad\quad\quad \textbf{else}\quad\ \ g\ (q \times k, (2 \times q + r) \times l, t \times l, k + 1, \\
\quad\quad\quad\quad\quad\quad div\ (q \times (7 \times k + 2) + r \times l)\ (t \times l), l + 2)
\end{array}
$$

is based on *infinite* sequences, and generates digits without bound.

3. Number representations

Familiar representations use a *fixed-radix* base; consider

$$\pi = 3 + \frac{1}{10}\left(1 + \frac{1}{10}\left(4 + \frac{1}{10}\left(1 + \frac{1}{10}\left(5 + \cdots\right)\right)\right)\right)$$

as number $(3; 1, 4, 1, 5, \ldots)$ in fixed-radix base $\mathcal{F}_{10} = (\frac{1}{10}, \frac{1}{10}, \frac{1}{10}, \ldots)$.

Similarly, think of expansion

$$\pi = 2 + \frac{1}{3}\left(2 + \frac{2}{5}\left(2 + \frac{3}{7}\left(\cdots\left(2 + \frac{i}{2i+1}\left(\cdots\right)\right)\right)\right)\right)$$

as number $(2; 2, 2, 2, \ldots)$ in *mixed-radix* base $\mathcal{B} = (\frac{1}{3}, \frac{2}{5}, \frac{3}{7}, \ldots)$.

Computing the digits of π is then radix conversion from $\mathcal{B}$ to $\mathcal{F}_{10}$.

Regular representations: digit i after the point is

- in $[0, 9]$, and 'maximal fraction' is $(0; 9, 9, 9 \ldots) = 1$, for $\mathcal{F}_{10}$;
- in $[0, 2i]$, and maximal fraction is $(0; 2, 4, 6 \ldots) = 2$, for $\mathcal{B}$.

4. Converting to fixed-radix base

Digits in base $\mathcal{F}_{10}$ of number x (assume $0 \leq x < 10$):

- first digit $d = \lfloor x \rfloor$
- remainder is $x - d$
- remaining digits obtained from $10 \times (x - d)$

In Haskell:

$$\begin{aligned}&\mathit{decimal}\ x = d : \mathit{decimal}\ (10 \times (x - \mathit{fromIntegral}\ d))\\&\quad \textbf{where}\ d = \mathit{floor}\ x\end{aligned}$$

We have to do this for number x represented in $\mathcal{B}$.

5. Operations in mixed-radix base

For number $x = (a_0; a_1, a_2, a_3 \ldots)$ in $\mathcal{B}$,

- $\lfloor x \rfloor$ is either a_0 or $a_0 + 1$, depending on whether remainder $(0; a_1, a_2, a_3 \ldots)$ is in $[0, 1)$ or $[1, 2)$
- (remainder cannot be 2, for irrational x)
- so need to buffer any 9s produced, in case of carries
- multiplying x by 10 can be achieved by multiplying each a_i by 10
- this typically yields an *irregular* representation
- for *finite* number, regularize from right to left, carrying leftwards

For *infinite* number, regularization needs to be left to right.
This can be done by *streaming*.

6. Streaming: the idea

Consider conversion of *infinite* representations from base $\mathcal{F}_m$ to $\mathcal{F}_n$.

Key idea:

first few input digits determine first few output digits.

So consume first few, produce first few, continue with remainder.

Maintain additional information, representing the function from the remaining inputs to the remaining outputs: with input

$$x = \frac{1}{m}\left(a_0 + \frac{1}{m}\left(a_1 + \cdots\right)\right)$$

after $a_0, a_1, \ldots, a_{i-1}$ have been consumed and $b_0, b_1, \ldots, b_{j-1}$ produced,

$$x = \frac{1}{n}\left(b_0 + \frac{1}{n}\left(b_1 + \cdots + \frac{1}{n}\left(b_{j-1} + v \times \left(u + \frac{1}{m}\left(a_i + \frac{1}{m}\left(a_{i+1} + \cdots\right)\right)\right)\right)\right)\right)$$

Represent that function by the pair (u, v) of rationals.
Initially, $i = j = 0$ and $(u, v) = (0, 1)$.
Commit when $v \times (u + 0)$ and $v \times (u + 1)$ have same first digit in base n.

7. Streaming: an example

For example, $1/e = 0.100221\ldots$ in $\mathcal{F}_3$, and $0.240\ldots$ in $\mathcal{F}_7$.

First three input digits 100 determine first output digit 2:

$$0.2_7 < 0.100_3 < 0.101_3 < 0.25_7$$

So consume first three input digits, produce first output digit; continue with remainder.

First few states of the conversion:

a_i	u, v	b_j
	$\frac{0}{1}, \frac{1}{1}$	
1	$\frac{1}{1}, \frac{1}{3}$	
0	$\frac{3}{1}, \frac{1}{9}$	
0	$\frac{9}{1}, \frac{1}{27}$	
	$\frac{9}{7}, \frac{7}{27}$	2
2	$\frac{41}{7}, \frac{7}{81}$	
2	$\frac{137}{7}, \frac{7}{243}$	
1	$\frac{418}{7}, \frac{7}{729}$	
	$\frac{10}{49}, \frac{49}{729}$	4

First *safe* state is $(u, v) = (\frac{9}{1}, \frac{1}{27})$, the first for which we have:

$$\lfloor 7 \times v \times u \rfloor = \lfloor 7 \times \tfrac{1}{27} \times \tfrac{9}{1} \rfloor = 2 = \lfloor 7 \times \tfrac{1}{27} \times (\tfrac{9}{1} + 1) \rfloor = \lfloor 7 \times v \times (u + 1) \rfloor$$

8. Streaming: the pattern

$$
\begin{array}{l}
\mathit{stream} :: (b \to \mathit{Bool}) \to (b \to c) \to (b \to b) \to (b \to a \to b) \to b \to [\,a\,] \to [\,c\,] \\
\mathit{stream\ safe\ next\ prod\ cons}\ z\ (x : \mathit{xs}) \\
\quad = \mathbf{if}\ \mathit{safe}\ z\ \mathbf{then}\ y : \mathit{stream\ safe\ next\ prod\ cons}\ (\mathit{prod}\ z)\ (x : \mathit{xs}) \\
\qquad\qquad\qquad \mathbf{else}\quad \mathit{stream\ safe\ next\ prod\ cons}\ (\mathit{cons}\ z\ x)\ \mathit{xs} \\
\qquad \mathbf{where}\ y = \mathit{next}\ z
\end{array}
$$

In particular,

$$
\begin{array}{l}
\mathit{convert}\ (m, n)\ \mathit{xs} = \mathit{stream\ safe\ next\ prod\ cons\ init\ xs}\ \mathbf{where} \\
\quad (m', n') \qquad\quad = (\mathit{fromInteger}\ m, \mathit{fromInteger}\ n) \\
\quad \mathit{init} \qquad\qquad\; = (0 \,\%\, 1, 1 \,\%\, 1) \\
\quad \mathit{next}\ (u, v) \quad\; = \mathit{floor}\ (u \times v \times n') \\
\quad \mathit{safe}\ (u, v) \quad\;\, = (\mathit{next}\ (u, v) == \mathit{floor}\ ((u + 1) \times v \times n')) \\
\quad \mathit{prod}\ (u, v) \quad\; = (u - \mathit{fromInteger}\ (\mathit{next}\ (u, v))\ /\ (v \times n'), v \times n') \\
\quad \mathit{cons}\ (u, v)\ x = (\mathit{fromInteger}\ x + u \times m', v\ /\ m')
\end{array}
$$

9. Back to π

Can use streaming to regularize an infinite representation.
But there is a more direct approach to computing the digits of π.

$$\begin{aligned}\pi &= 2 + \tfrac{1}{3}\left(2 + \tfrac{2}{5}\left(2 + \tfrac{3}{7}\left(\cdots\left(2 + \tfrac{i}{2i+1}\left(\cdots\right)\right)\right)\right)\right) \\ &= \left(2 + \tfrac{1}{3}\times\right)\left(2 + \tfrac{2}{5}\times\right)\left(2 + \tfrac{3}{7}\times\right)\cdots\left(2 + \tfrac{i}{2i+1}\times\right)\cdots\end{aligned}$$

—composition of infinite series of *linear fractional transformations* $\left(\begin{smallmatrix} q & r \\ s & t \end{smallmatrix}\right)$.

- fixpoint of $(2 + \frac{1}{3}\times)$ is 3, fixpoint of $(2 + \frac{1}{2}\times)$ is 4
- so each LFT maps interval $[3, 4]$ onto a subinterval of itself
- each LFT shrinks by at least a factor of 2
- so compositions of such LFTs converge to a point in $[3, 4]$.

Finding that point is another *change of representation*,
from infinite sequences of LFTs to infinite sequences of decimal digits.

10. Streaming π

- each *input* LFT is a 2-by-2 matrix of integers

  $$\left[\begin{pmatrix}1&6\\0&3\end{pmatrix},\begin{pmatrix}2&10\\0&5\end{pmatrix},\begin{pmatrix}3&14\\0&7\end{pmatrix},\ldots\right]$$

- *state* is another LFT z
- *initial* state is identity LFT, $\begin{pmatrix}1&0\\0&1\end{pmatrix}$
- z is *safe* if image under z of $[3,4]$ all has same integer part, n

  $$\begin{pmatrix}q&r\\s&t\end{pmatrix}\times[3,4]=\left[\frac{3q+r}{3s+t},\frac{4q+r}{4s+t}\right]$$

- then *produce* digit n, and multiply state by $\begin{pmatrix}10&-10n\\0&1\end{pmatrix}$, inverse of the LFT $x\mapsto n+\frac{x}{10}$
- otherwise *consume* next LFT, by matrix multiplication

11. Program for π

$$
\begin{array}{l}
\mathit{pi} = \mathit{stream}\ \mathit{safe}\ \mathit{next}\ \mathit{prod}\ \mathit{cons}\ \mathit{init}\ \mathit{lfts}\ \mathbf{where} \\
\quad \begin{array}{lcl}
\mathit{init} & = & \mathit{unit} \\
\mathit{lfts} & = & [\,(k, 4 \times k + 2, 0, 2 \times k + 1) \mid k \leftarrow [1\,..]\,] \\
\mathit{next}\ z & = & \mathit{floor}\ (\mathit{extr}\ z\ 3) \\
\mathit{safe}\ z & = & (\mathit{next}\ z == \mathit{floor}\ (\mathit{extr}\ z\ 4)) \\
\mathit{prod}\ z & = & \mathit{comp}\ (10, -10 \times \mathit{next}\ z, 0, 1)\ z \\
\mathit{cons}\ z\ z' & = & \mathit{comp}\ z\ z'
\end{array}
\end{array}
$$

where *comp* is matrix multiplication, and *extr* extracts the LFT from a matrix $\left(\begin{smallmatrix} q & r \\ s & t \end{smallmatrix}\right)$, taking x to $(q \times x + r)\,/\,(s \times x + t)$.

Obfuscated program obtained from this by inlining definitions,
and observing that invariant $s = 0$ holds in all our LFTs $\left(\begin{smallmatrix} q & r \\ s & t \end{smallmatrix}\right)$.

12. Reasoning about *stream*

For finite sequences, express change of representation by *abstraction*, consuming one representation:

$$
\begin{aligned}
&\mathit{foldl} :: (b \to a \to b) \to b \to [\,a\,] \to b \\
&\mathit{foldl}\ h\ z\ (x : xs) = \mathit{foldl}\ h\ (h\ z\ x)\ xs \\
&\mathit{foldl}\ h\ z\ [\,] \qquad\quad = z
\end{aligned}
$$

followed by *reification*, producing the other:

$$
\begin{aligned}
&\mathit{unfoldr} :: (b \to \mathit{Bool}) \to (b \to c) \to (b \to b) \to b \to [\,c\,] \\
&\mathit{unfoldr}\ p\ f\ g\ z = \mathbf{if}\ p\ z\ \mathbf{then}\ f\ z : \mathit{unfoldr}\ p\ f\ g\ (g\ z)\ \mathbf{else}\ [\,]
\end{aligned}
$$

and $\mathit{convert}\ p\ f\ g\ h\ z\ xs = \mathit{unfoldr}\ p\ f\ g\ (\mathit{foldl}\ h\ z\ xs)$.

Sometimes this process can be *streamed*: if state z satisfies

$$\exists y \bullet \forall xs \bullet \mathit{convert}\ p\ f\ g\ h\ z\ xs = y : \ldots$$

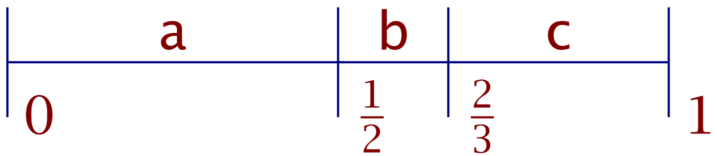
then it is safe to produce y from z before consuming any more of xs.

13. Arithmetic coding

Data compression, of a text to a bit sequence:

- *distribute alphabet* across unit interval
- *narrow* unit interval, character by character
- output *shortest binary fraction* in final interval

For example, with $\mathsf{a} \mapsto [0, \frac{1}{2}]$, $\mathsf{b} \mapsto [\frac{1}{2}, \frac{2}{3}]$, $\mathsf{c} \mapsto [\frac{2}{3}, 1]$



and text abacab:

$$[0, 1] \xrightarrow{\mathsf{a}} [0, \tfrac{1}{2}] \xrightarrow{\mathsf{b}} [\tfrac{1}{4}, \tfrac{1}{3}] \xrightarrow{\mathsf{a}} [\tfrac{1}{4}, \tfrac{7}{24}] \xrightarrow{\mathsf{c}} [\tfrac{5}{18}, \tfrac{7}{24}] \xrightarrow{\mathsf{a}} [\tfrac{5}{18}, \tfrac{41}{144}] \xrightarrow{\mathsf{b}} [\tfrac{9}{32}, \tfrac{61}{216}]$$

and $[\frac{9}{32}, \frac{61}{216}]$ contains 0.01001 and no shorter binary fraction.

For efficiency, we wish to *stream* the output.
Lecturing on arithmetic coding led us to the streaming abstraction.

14. Further reading

- "A Spigot Algorithm for the Digits of π", Stanley Rabinowitz and Stan Wagon, *American Mathematical Monthly*, **102**:195–203, 1995
- "Unbounded Spigot Algorithms for the Digits of π", Jeremy Gibbons, *American Mathematical Monthly*, **113**:318–328, 2006
- "Metamorphisms: Streaming Representation-Changers", Jeremy Gibbons, *Science of Computer Programming*, **65**:108–139, 2007
- "Arithmetic Coding with Folds and Unfolds", Richard Bird and Jeremy Gibbons, *Advanced Functional Programming*, LNCS **2638**:1–26, 2003

My papers are available from my webpage:

`http://www.cs.ox.ac.uk/jeremy.gibbons`