Unbounded Spigot Algorithms for $\pi$

Jeremy Gibbons
IIJ, March 2017
1. Spigot algorithms for $\pi$

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

```c
int 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

\[ \pi = g(1, 0, 1, 1, 3, 3) \text{ where} \]
\[ g(q, r, t, k, n, l) = \]
\[ \text{if } 4 \times q + r - t < n \times t \]
\[ \text{then } n : g(10 \times q, 10 \times (r - n \times t), t, k, \]
\[ \text{div} (10 \times (3 \times q + r)) t - 10 \times n, l) \]
\[ \text{else } g(q \times k, (2 \times q + r) \times l, t \times l, k + 1, \]
\[ \text{div} (q \times (7 \times k + 2) + r \times l) (t \times l), l + 2) \]

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

Familiar representations use a \textit{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 \(\mathcal{F}_{10} = \left( \frac{1}{10}, \frac{1}{10}, \frac{1}{10}, \ldots \right)\).

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 \textit{mixed-radix} base \(\mathcal{B} = \left( \frac{1}{3}, \frac{2}{5}, \frac{3}{7}, \ldots \right)\).

Computing the digits of \(\pi\) 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:

```
decimal x = d : decimal (10 * (x - fromIntegral d))
where d = floor x
```

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 \( B \),

- \( \lfloor x \rfloor \) 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 \textit{irregular} representation

- for \textit{finite} number, regularize from right to left, carrying leftwards

For \textit{infinite} number, regularization needs to be left to right.
This can be done by \textit{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,

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

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

For example, \( \frac{1}{e} = 0.100221 \ldots \) in \( \mathcal{F}_3 \), and \( 0.240 \ldots \) in \( \mathcal{F}_7 \).

First three input digits 100 determine first output digit 2:

\[
0.27 < 0.100_3 < 0.101_3 < 0.257
\]

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

First few states of the conversion:

<table>
<thead>
<tr>
<th>( a_i )</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>2</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>( u, v )</td>
<td>( 0, \frac{1}{1} )</td>
<td>( 1, \frac{1}{1} )</td>
<td>( 1, \frac{1}{3} )</td>
<td>( 3, \frac{1}{9} )</td>
<td>( 9, \frac{1}{27} )</td>
<td>( 9, \frac{7}{27} )</td>
</tr>
<tr>
<td>( b_j )</td>
<td>2</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

\[
[7 \times v \times u] = [7 \times \frac{1}{27} \times \frac{9}{1}] = 2 = [7 \times \frac{1}{27} \times (\frac{9}{1} + 1)] = [7 \times v \times (u + 1)]
\]
8. Streaming: the pattern

\[
\text{stream} :: (b \to \text{Bool}) \to (b \to c) \to (b \to b) \to (b \to a \to b) \to b \to [a] \to [c]
\]

\[
\text{stream safe next prod cons } z (x : xs)
\]

\[
= \text{if } \text{safe } z \text{ then } y : \text{stream safe next prod cons } (\text{prod } z) (x : xs)
\]

\[
\text{else } \text{stream safe next prod cons } (\text{cons } z x) xs
\]

where \( y = \text{next } z \)

In particular,

\[
\text{convert } (m, n) xs = \text{stream safe next prod cons init } xs \text{ where }
\]

\[
(m', n') = (\text{fromInteger } m, \text{fromInteger } n)
\]

\[\text{init} = (0 \% 1, 1 \% 1)\]

\[\text{next } (u, v) = \text{floor } (u \times v \times n')\]

\[\text{safe } (u, v) = (\text{next } (u, v) == \text{floor } ((u + 1) \times v \times n'))\]

\[\text{prod } (u, v) = (u - \text{fromInteger } (\text{next } (u, v)) \div (v \times n'), v \times n')\]

\[\text{cons } (u, v) x = (\text{fromInteger } x + u \times m', v \div m')\]
9. Back to $\pi$

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

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

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

—composition of infinite series of linear fractional transformations $\left( \frac{q}{s}, \frac{r}{t} \right)$.

- fixpoint of $\left( 2 + \frac{1}{3} \times \right)$ is 3, fixpoint of $\left( 2 + \frac{1}{2} \times \right)$ 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
  
  \[
  \left( \begin{array}{cc}
  1 & 6 \\
  0 & 3 \\
  \end{array} \right), \quad
  \left( \begin{array}{cc}
  2 & 10 \\
  0 & 5 \\
  \end{array} \right), \quad
  \left( \begin{array}{cc}
  3 & 14 \\
  0 & 7 \\
  \end{array} \right), \ldots
  \]

- State is another LFT \( z \)

- Initial state is identity LFT, \( \left( \begin{array}{cc} 1 & 0 \\ 0 & 1 \end{array} \right) \)

- \( z \) is safe if image under \( z \) of \([3, 4]\) all has same integer part, \( n \)
  
  \[
  \left( \begin{array}{cc}
  q & r \\
  s & t \\
  \end{array} \right) \times [3, 4] = \left[ \frac{3q+r}{3s+t}, \frac{4q+r}{4s+t} \right]
  \]

- Then produce digit \( n \), and multiply state by \( \left( \begin{array}{cc} 10 & -10n \\ 0 & 1 \end{array} \right) \), inverse of the LFT \( x \to n + \frac{x}{10} \)

- Otherwise consume next LFT, by matrix multiplication
11. Program for $\pi$

$$\pi = \text{stream safe next prod cons init lfts where}$$

$$\begin{align*}
\text{init} & = \text{unit} \\
\text{lfts} & = [(k, 4 \times k + 2, 0, 2 \times k + 1) | k \leftarrow [1..]] \\
\text{next z} & = \text{floor} (\text{extr z} 3) \\
\text{safe z} & = (\text{next z} == \text{floor} (\text{extr z} 4)) \\
\text{prod z} & = \text{comp} (10, -10 \times \text{next z}, 0, 1) z \\
\text{cons z z'} & = \text{comp} z z'
\end{align*}$$

where $\text{comp}$ is matrix multiplication, and $\text{extr}$ extracts the LFT from a matrix $\left( \begin{array}{cc} q & r \\ s & t \end{array} \right)$, 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 $\left( \begin{array}{cc} q & r \\ s & t \end{array} \right)$. 
12. Reasoning about stream

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

\[
\text{foldl} :: (b \to a \to b) \to b \to [a] \to b
\]

\[
\text{foldl } h \ z \ (x : xs) = \text{foldl} \ h \ (h \ z \ x) \ xs
\]

\[
\text{foldl } h \ z \ [] = z
\]

followed by reification, producing the other:

\[
\text{unfoldr} :: (b \to \text{Bool}) \to (b \to c) \to (b \to b) \to b \to [c]
\]

\[
\text{unfoldr} \ p \ f \ g \ z = \text{if } p \ z \ \text{then } f \ z : \text{unfoldr} \ p \ f \ g \ (g \ z) \ \text{else }[
\]

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

Sometimes this process can be streamed: if state \( z \) satisfies

\[
\exists y \cdot \forall xs \cdot \text{convert } p \ f \ g \ h \ z \ xs = y : \ldots
\]

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 \( a \mapsto [0, \frac{1}{2}], b \mapsto \left[\frac{1}{2}, \frac{2}{3}\right], c \mapsto \left[\frac{2}{3}, 1\right] \):

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1/2</td>
<td>2/3</td>
<td>1</td>
</tr>
</tbody>
</table>

and text *abacab*:

\[
[0, 1] \rightarrow [0, \frac{1}{2}] \rightarrow \left[\frac{1}{4}, \frac{1}{3}\right] \rightarrow \left[\frac{1}{4}, \frac{7}{24}\right] \rightarrow \left[\frac{5}{18}, \frac{7}{24}\right] \rightarrow \left[\frac{5}{18}, \frac{41}{144}\right] \rightarrow \left[\frac{9}{32}, \frac{61}{216}\right]
\]

and \( \left[\frac{9}{32}, \frac{61}{216}\right] \) 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


My papers are available from my webpage:

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