Coinductive Stream Calculus in Haskell

Joost Winter

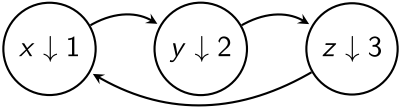
Centrum Wiskunde & Informatica

June 4, 2013

A short motivation

- Haskell has nice built-in support or infinitary, or coinductive data types, such as streams.
- I will present (another) implementation of this.
- A bridge between the theory of behavioural differential equations and 'reality'
- ... with a 'quasi-empirical' flavour thanks to OEIS integration

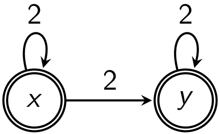
Representations of (simple) streams



$$\begin{aligned} o(x) &= 1 & x' &= y \\ o(y) &= 2 & y' &= z \\ o(z) &= 3 & z' &= x \end{aligned}$$

```
x = 1:2:3:x
```

Weighted automata...



$$\begin{aligned} o(x) &= 1 \qquad & x' &= 2(x+y) \\ o(y) &= 1 \qquad & y' &= 2y \end{aligned}$$

$$x \to 2x+2y \to 4x+8y \to 8x+24y \to 16x+64y \to 32x+160y \ldots$$

. . . and their representation in Haskell

$$\begin{aligned} o(x) &= 1 \qquad & x' &= 2(x+y) \\ o(y) &= 1 \qquad & y' &= 2y \end{aligned}$$

gives:

```
x = 1 : 2 *! x + 2 *! y
y = 1 : 2 *! y
```

Modelling recurrences as streams...

We start from a simple (and familiar) recurrence

$$a(0) = a(1) = 1 \qquad a(n+2) = a(n) + a(n+1)$$

...and note the last equation certainly holds if the following stream differential equation holds:

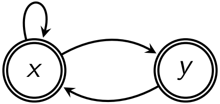
$$\sigma'' = \sigma + \sigma'$$

This now directly gives us:

```
fibs = 1 : 1 : fibs + d fibs
```

... and weighted automata

The corresponding weighted automaton:



Modelling recurrences as streams (generally)...

Recurrence (with $a(0), \dots a(k-1)$ given):

$$a(n+k) = \sum_{0 \leq i < k} b_i a(n+i)$$

Stream differential equation:

$$\sigma^{(k)} = \sum_{0 \leq i < k} b_i \sigma^{(i)}$$

Haskell code:

```
s = a 0:...:a (k-1):sum [b i *! dd s i | i <- 0..(k-1)]
```

Streams as a Num type

(a trick due to Douglas McIlroy)

```
instance Num a => Num [a] where
  fromInteger = i . fromInteger
  negate = map negate
  (+) = zipWith (+)
  s * t = o s * o t : d s * t + o s *! d t
```

Note: with the exception of the (convolution) product, all operators are straight liftings from the underlying type. The last line corresponds to the (Brzozowski) product rule:

$$o(st) = o(s)o(t) \qquad (st)' = s't + o(s)t'$$

Algebraic systems

- Correspond to derivation counts in unambiguous CFGs in GNF in the case of coefficients $\in \mathbb{N}$.
- Given a CFG in GNF, we can directly construct an algebraic system for its counting function.

Matching pairs of parentheses – Catalan numbers

The following CFG generates matching pairs of parentheses over an alphabet $\{a, b\}$:

$$x \to \epsilon \mid axbx$$

Corresponding system of bdes:

$$o(x) = 1 \quad x_a = xbx \quad x_b = 0$$

Transform this into a system over a single alphabet symbol $\mathcal{X}$:

$$o(x) = 1 \quad x' = \mathcal{X}x^2$$

Matching pairs of parentheses – Catalan numbers

$$o(x) = 1 \quad x' = \mathcal{X}x^2$$

In Haskell:

```
x = 1 : 0 : x^2
```

Gives:

$$(1, 0, 1, 0, 2, 0, 5, 0, 14, 0, 42, 0, 132, 0, 429, 0, 1430, 0, 4862, 0, \ldots)$$

The zip-operator. . .

The zip of two streams alternately takes an element of either stream. Zip can be defined as follows:

```
myzip s t = o s : myzip t (d s)
```

In the opposite direction, we have operators even and odd, satisfying

$$\mathrm{zip}(\mathrm{even}(\mathrm{x}), \mathrm{odd}(\mathrm{x})) = x$$

. . . and divide and conquer recurrences

Consider a recurrence of the type

$$a(2n) = ba(n) \qquad a(2n+1) = ca(n) + d$$

This gives stream equations

$$\text{even}(\sigma) = b\sigma \qquad \text{odd}(\sigma) = c\sigma + d \cdot \mathbf{ones}$$

or

$$\sigma = \text{zip}(b\sigma, c\sigma)$$

which always can be transformed into (guarded) systems of stream differential equations.

Per Nørgård's Infinity Sequence

The Danish composer Per Nørgård used a sequence of this type in several of his compositions. It is given by $a(0) = 1$ and

$$a(2n) = -a(n) \qquad a(2n+1) = a(n) + 1$$

It starts out as:

$$(0, 1, -1, 2, 1, 0, -2, 3, -1, 2, 0, 1, 2, -1, -3, 4, 1, 0, -2, 3, 0, 1, -1)$$

A stream differential equation for (the tail of) this sequence:

$$x = 1 : \text{zip}(-x, x + \textbf{ones})$$

Wrap-up

- Haskell provides a very nice setting for stream calculus in action.
- With QStream, we can inspect streams and look them up on OEIS.
- Stream differential equations (and the corresponding Haskell specifications) are often particularly concise and elegant.

Related work

- Earlier stream calculus implementations in Haskell by Douglas McIlroy and Ralf Hinze.
- Other stream tools (mostly for proving stream equality) include e.g. CIRC (Dorel Lucanu) and Streambox (Hans Zantema and Jörg Endrullis)
- Theoretical work on behavioural differential equations