QStream: a Suite of Streams

Joost Winter

Centrum Wiskunde & Informatica

September 3, 2013

Streams and Haskell

- Thanks to lazy evaluation, streams and other coinductive data types are easily implemented in Haskell
- Existing implementations by e.g. Hinze, McIlroy
- Connected to/based on Rutten's coinductive stream calculus

This talk

- qStream: somewhere between a hack and a tool
- Building further on earlier interpretations
- Providing the theory of behavioural differential equations with concrete examples
- . . . with a 'quasi-empirical' flavour thanks to OEIS integration
- . . . and close connections to (coalgebraic) automata theory
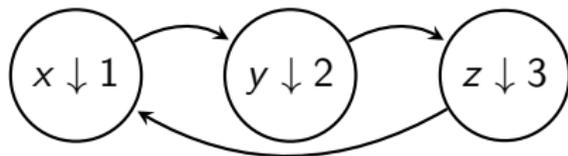
Representations of (simple) streams
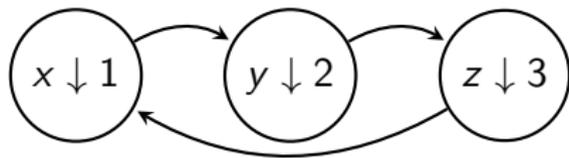
Representations of (simple) streams
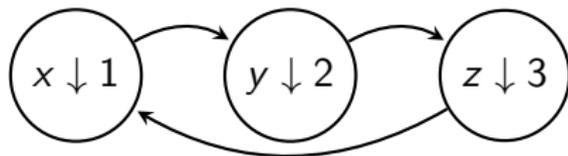
$$
\begin{array}{rcl l}
o(x) &=& 1 \qquad & x' &=& y \\
o(y) &=& 2 \qquad & y' &=& z \\
o(z) &=& 3 \qquad & z' &=& x
\end{array}
$$

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
```

# Towards algebra, or: streams as a `Num` type

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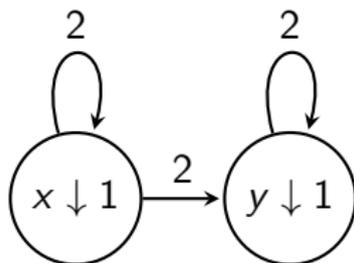
Note: with the exception of the (convolution) product, all
operators are straight liftings from the underlying type.
The last line corresponds to the (Brzozowski) product rule:

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

Weighted automata — rational streams
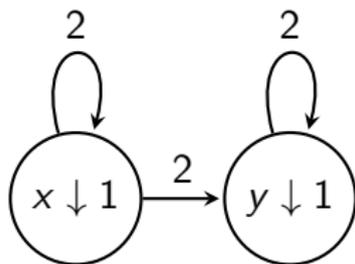
Weighted automata — rational streams

$$
\begin{array}{rclcrcl}
o(x) &=& 1 & \quad & x' &=& 2x + 2y \\
o(y) &=& 1 & \quad & y' &=& 2y
\end{array}
$$

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

. . . and their representation in Haskell

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

gives:

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

. . . and their representation in Haskell

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

gives:

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

The scalar product is defined with

```
(*!) k = map ((*) k)
```

Interfacing with the OEIS

A couple of helper functions to interface with the OEIS:

- `info`
- `rangeinfo`
- `maketable`
- `oeistable`

Modelling recurrences as streams...

We start from a simple (and familiar) recurrence

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

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:

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

Modelling recurrences as streams...

We start from a simple (and familiar) recurrence

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

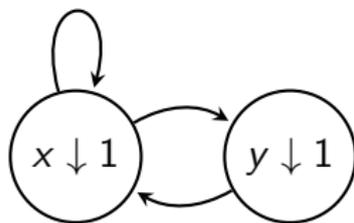...and note the last equation certainly holds if the following stream differential equation holds:

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

This now directly gives us:

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

. . . and weighted automata

The corresponding weighted automaton:

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

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

Stream differential equation:

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

Haskell code:

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

Algebraic systems

- Correspond to the counting function of unambiguous CFGs in GNF in the case of coefficients $\in \mathbb{N}$.

Algebraic systems

- Correspond to the counting function of unambiguous CFGs in GNF in the case of coefficients $\in \mathbb{N}$.
- Given an unambiguous 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

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

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 number of *m*-ary search trees on *n* keys

Haskell:

```haskell
st m = take (m - 1) ones ++ st m ^ m
```

The number of *m*-ary search trees on *n* keys

Haskell:

```
st m = take (m - 1) ones ++ st m ^ m
```

G.f.:

$$A(\mathfrak{X}) = \mathfrak{X}^{m-1} A^m(\mathfrak{X}) + \frac{1 - \mathfrak{X}^{m-1}}{1 - \mathfrak{X}}$$

The number of *m*-ary search trees on *n* keys

Haskell:

```
st m = take (m - 1) ones ++ st m ^ m
```

G.f.:

$$A(\mathfrak{X}) = \mathfrak{X}^{m-1} A^m(\mathfrak{X}) + \frac{1 - \mathfrak{X}^{m-1}}{1 - \mathfrak{X}}$$

Explicit formula:

Theorem 1 *The number of m-ary search trees on n keys is given by*

$$\tau_{m,n} = \sum \binom{k_+}{k_0, \ldots, k_{m-2}} \frac{[(m/(m-1))(k_+ - 1)]!}{[(k_+ - 1)/(m-1)]! k_+!}, \tag{1}$$

where the sum is over all $(m - 1)$-tuples $(k_0, \ldots, k_{m-2})$ such that: (i) $k_i \geq 0$ for $0 \leq i \leq m-2$, (ii) $m-1$ divides $k_+ - 1$, and (iii) $\sum_{j=0}^{m-2}(j+1)k_j = n+1$.

(from Fill/Dobrow: *The number of m-ary searchtrees on n keys*)

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

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

Consider a recurrence of the type

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

This gives stream equations

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

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:

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

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:

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

Stream differential equation:

$$x = 0 : \mathsf{zip}(x + \mathbf{ones}, -x')$$

Haskell:

```
x = 0 : xzip [x + ones, -d x]
```

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.

URL: `http://homepages.cwi.nl/~winter/qstream`

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