QStream: A Suite of Streams

Joost Winter[1,2⋆]

[1] Centrum Wiskunde & Informatica (CWI)
[2] LIACS – Leiden University

Abstract. We present a simple tool in Haskell, QStream[3], implementing the technique of *coinductive counting* by making use of Haskell's built-in coinduction capabilities. We furthermore provide a number of useful tools for stream exploration, including a number of pretty print functions and integration with the Online Encyclopedia of Integer Sequences.

1 Introduction

It has been observed before, for example in [McI01] and [Hin11], that Haskell's built-in coinduction capabilities allow for easy and simple specifications of *streams* making use of variants of the coinductive stream calculus presented in e.g. [Rut05].

Borrowing some terminology from [McI01], the present paper can perhaps be considered as presenting another variation on this theme. Compared to [McI01], in which expressions for generating functions are given a coinductive semantics directly, our main focus lies on the connection with weighted automata, and systems of behavioural differential equations, using which rational and algebraic (or context-free) streams (or formal power series) can be chacaterized. We have opted for using a set of operators which is minimal but still expressive enough to be able to classify the complete classes of rational and algebraic streams.

Often, the specifications obtained this way turn out to be surprisingly elegant, although with a different flavour from the more familiar generating function expressions. For example, given a fixed integer k, the generating function for the stream of powers of k is $1/(1-kX)$, whereas the corresponding system of behavioural differential equations consists of the equations $o(x) = 1$ and $x' = kx$.

We have built a simple package, QStream, providing the necessary definitions required for such coinductive reasoning, as well as a usable and simple interface for stream exploration, including a number of pretty-printing functions for systems of streams, as well as an interface to the Online Encyclopedia of Integer Sequences[4].

Introductions to generating functions can be found, for example, in [Wil06] and [GKP94]. The idea of using coinductive techniques and weighted automata to describe combinatorial problems can be traced back to [Rut02]. For background material on the theory of rational and algebraic streams, we refer to [Rut08] and [BRW12], respectively.

⋆ Supported by the NWO project CoRE.

[3] The letter Q is intended to highlight the connection to automata theory.

[4] `http://oeis.org`

The QStream package has been developed and tested using version 7.4.1 of The Glorious Glasgow Haskell Compilation System[5], and can be downloaded from `http://homepages.cwi.nl/~winter/qstream`.

Related work: Existing tools aimed at stream calculus tend to fall into two distinct categories. The first category consists of tools, generally more 'heavy-weight' and with an emphasis on proving equality of streams: this group of tools includes CIRC [LGCR09] and Streambox[6].

The second category – in which computation of streams, rather than proving equality is the main aim – consists of more 'lightweight' implementations in Haskell. Earlier implementations in this category include those by McIlroy[7] and Hinze[8], which both have been a source of inspiration for QStream. Compared to these existing implementations, the present implementation attempts to provide a closer link with the underlying framework of systems of behavioural differential equations, as well as adding some useful interactivity by providing integration with the Online Encyclopedia of Integer Sequences.

Acknowledgements: The author would like to thank Marcello Bonsangue and Jan Rutten for their comments and constructive criticism, as well as to the anonymous reviewers for their comments and suggestions.

2 A Suite of Streams

Elementary coinductive definitions: Haskell's built-in capabilities allow for easy coinductive specifications of streams. As a very elementary example, consider the specification `x = 1:2:3:x`. With this specification, it is directly possible to obtain initial segments of the stream thus defined as follows:

```
Prelude> take 10 x
[1,2,3,1,2,3,1,2,3,1]
```

Eventually periodic streams can, in general, be specified using specifications of this type. Although elementary, some important streams can already be defined now, such as the stream $0, 0, 0, \ldots$ defined by `zero = 0:zero`, and the stream $1, 1, 1, \ldots$, defined by `ones = 1:ones`.

In order to be able to produce more interesting classes of streams than the eventually periodic streams, we coinductively define a few basic operations on streams. Moreover, following McIlroy's example, we let streams be a `Num` type, enabling us to reap the fruits of type coercion, use the standard operators `+` and `*`, and furthermore directly inherit functions such as `sum`, `^`, etc. We also include a separate scalar product operator `*!`, which brings extra conceptual clarity as well as a tremendous performance boost.

The behavioural differential equations for sum, scalar product, and convolution product can be now represented directly in Haskell:

[5] `http://www.haskell.org/ghc`
[6] `http://infinity.few.vu.nl/streambox/`
[7] `http://www.cs.dartmouth.edu/~doug/powser.html`
[8] `http://hackage.haskell.org/packages/archive/hinze-streams`

```
s  + t = o s + o t : d s  + d t       -- Or: (+) = zipWith (+)
k *! t =   k * o t :   k *! d t       -- Or: (*!) k = map ((*) k)
s  * t = o s * o t : d s  *   t + o s *! d t
```

Here `o` and `d`, standing for *output* and *derivative* respectively, are simply defined as synonyms for `head` and `tail`.

Rational and algebraic streams: As a first example of a rational stream, consider the definition:

```
fibs = 0 : 1 : fibs + d fibs
```

This definition corresponds to the following system of behavioural differential equations

$$o(x) = 0 \qquad o(y) = 1 \qquad x' = y \qquad y' = x + y$$

and yields the Fibonacci sequence $0, 1, 1, 2, 3, 5, 8, \ldots$

Making use of the existing `oeis` package on Hackage, our module `QStream.IO` provides a function `info`, which takes a stream of integers as argument, looks up an initial part of this stream in the Online Encyclopedia of Integer Sequences (`http://oeis.org`), and then displays its description and identifier.

```
*QStream> info fibs
Fibonacci numbers: F(n) = F(n-1) + F(n-2) with F(0) = 0 and F(1) =
1. (A000045)
```

For two more rational streams, consider:

```
dups = 1 : 2 *! dups
hypercube = 1 : 2 *! (hypercube + dups)
```

Here `dups` consists of the powers of 2: (1,2,4,8,...). The nth element of `hypercube`, on the other hand, is equal to the number of edges in a n-dimensional hypercube.

A celebrated example of an algebraic stream is the specification

```
cats = 1 : cats ^ 2
```

corresponding to the system of behavioural differential equations

$$o(x) = 1 \qquad x' = x^2$$

and yielding the stream of Catalan numbers $1, 1, 2, 5, 14, 42, 132, 429, 1430, \ldots$

Systems of streams: It is also possible to define complete systems of streams at once. As an example, the following stream systems yield, respectively, diagonal rows from Pascal's triangle, and the Stirling numbers of the 2nd kind:

```
pascal n = 1 : sum [ pascal i | i <- [1..n] ] -- A007318
stirling2 n = 1 : sum [ i *! stirling2 i | i <- [1..n] ] -- A008277
```

These Haskell specifications are in direct correspondence to the systems of behavioural differential equations

$$o(p_n) = 1 \qquad p'_n = \sum_{i=1}^{n} p_i \quad (n \in \mathbb{N}) \qquad \text{giving: } [\![p_n]\!](k) = \binom{n+k}{k}$$

$$o(s_n) = 1 \qquad s'_n = \sum_{i=1}^{n} i s_i \quad (n \in \mathbb{N}) \qquad \text{giving: } [\![s_n]\!](k) = \frac{1}{k!} \sum_{j=0}^{k} (-1)^{k-j} \binom{k}{j} j^{n+k}$$

which can easily be derived from the familiar recurrence relations for these sequences.

For these types of stream systems, again a few helper functions are provided: `rangeinfo` gives the description and OEIS numbers of a stream system for a provided range of values:

```
*QStream> rangeinfo stirling2 [2..4]
2: 2^n - 1. (Sometimes called Mersenne numbers, although that
name is usually reserved for A001348.) (A000225)
3: Stirling numbers of second kind S(n,3). (A000392)
4: Stirling numbers of the second kind, S(n,4). (A000453)
```

Another pair of helper functions, `maketable` and `oeistable`, generates a table providing an initial part of a function from natural numbers to streams, with `oeistable` presenting the OEIS ids in addition:

```
*QStream> oeistable pascal 12 5
1:  1 1  1  1  1   1   1   1   1   1    1    1   (A000012)
2:  1 2  3  4  5   6   7   8   9  10   11   12   (A000027)
3:  1 3  6 10 15  21  28  36  45  55   66   78   (A000217)
4:  1 4 10 20 35  56  84 120 165 220  286  364   (A000292)
5:  1 5 15 35 70 126 210 330 495 715 1001 1365   (A000332)
```

Sometimes, systems of behavioural differential equations, and the corresponding specifications in Haskell, are much simpler in form than the sometimes more familiar explicit formulas for these sequences. For example, the number of m-ary search trees on n keys is equal to the nth element of the stream `searchtrees m`, specified by:

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

This equation can easily be derived from the generating function specification

$$A(X) = \sum_{j=0}^{m-2} X^j + X^{m-1} A^m(X),$$

found in e.g. [FD97], where a corresponding explicit formula (omitted here due to space constraints) is also provided.

Building a catalog of streams: In [Plo92], generating functions for 1031 different integer sequences have been identified using gfun, a Maple package. With QStream, we have so far found behavioural differential equations for over 100 of the generating functions presented there [Plo92]: this small catalog can be found in the module `QStream.Plouffe`.

3 Conclusions and Future Work

So far, QStream has been, at least for its author, a useful tool in exploration of classes of streams and systems of behavioural differential equations. The underlying theoretical framework links up beautifully with Haskell, and typical Haskell features such as lazy evaluation. Even merely experimenting around a bit with coinductive specifications often yields interesting sequences; as well as elegant specifications for these sequences.

However, when parameterized systems such as `pascal` are involved, computation of streams turns out to be awkwardly slow. Possible tactics to address this issue include memoization, and direct modelling of weighted automata using linear combinations of weighted states. As a first step in the second direction, the module `QStream.Fast` hard-codes weighted automata for a relatively wide class of streams (including the `pascal` and `stirling2` stream systems). This approach, albeit ad hoc, already yields a huge speed up, resulting in much more reasonable computation times. Further work here should include a more modular approach, in which data types representing weighted automata are introduced.

Looking up streams on OEIS can be a rather slow process: somehow, especially looking up basic sequences (such as `ones` or the natural numbers) often is inexplicably slow. Although this issue is mostly out of our control, to remedy this, we might think for example of building a local database of OEIS entries.

As a final remark, we note that there should be a number of easy generalizations of this work: for example by moving from integers to rationals, or from streams to formal power series over noncommuting variables (or, equivalently, weighted languages). However, in neither of these cases would we be able to make any good use of the OEIS, which focuses on integer sequences.

References

[BRW12] Marcello M. Bonsangue, Jan J. M. M. Rutten, and Joost Winter. Defining context-free power series coalgebraically. *LNCS* 7399, 20–39. Springer, 2012.

[FD97] James Allen Fill and Robert P. Dobrow. The number of m-ary search trees on n keys. *Combinatorics, Probability & Computing*, 6(4):435–453, 1997.

[GKP94] Ronald L. Graham, Donald E. Knuth, and Oren Patashnik. *Concrete Mathematics: A Foundation for Computer Science*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edition, 1994.

[Hin11] Ralf Hinze. Concrete stream calculus—an extended study. *JFP*, 20(5-6):463–535, 2011.

[McI01] M. Douglas McIlroy. The music of streams. *Inf. Process. Lett.*, 77(2-4):189–195, 2001.

[LGCR09] Dorel Lucanu, Eugen-Ioan Goriac, Georgiana Caltais and Grigore Rosu. CIRC: A Behavioral Verification Tool Based on Circular Coinduction *LNCS* 5728, 433–442. Springer, 2009.

[Plo92] Simon Plouffe. Approximations de séries génératrices et quelques conjectures. Master's thesis, Université du Québec à Montréal, 1992.

[Rut02] Jan J. M. M. Rutten. Coinductive counting: bisimulation in enumerative combinatorics. *Electr. Notes Theor. Comput. Sci.*, 65(1):286–304, 2002.

[Rut05] Jan J. M. M. Rutten. A coinductive calculus of streams. *Mathematical Structures in Computer Science*, 15(1):93–147, 2005.

[Rut08] Jan J. M. M. Rutten. Rational streams coalgebraically. *Logical Methods in Computer Science*, 4(3), 2008.

[Wil06] Herbert S. Wilf. *Generatingfunctionology.* A. K. Peters, Ltd., Natick, MA, USA, 2006.