Infinite streams

Jörg Endrullis[*], Clemens Grabmayer[†], Dimitri Hendriks[‡], and Jan Willem Klop[§]

February 11, 2009

Abstract

Infinite streams of data are interesting from various points of view: Theoretically, because they are a paradigm example for the application of coalgebraic techniques as well as infinitary rewriting techniques. Practically, because infinite streams arise naturally in several applications concerning data transmissions. And graphically, because they can be visualized by 'drawing turtles' as graphic trajectories that display curious patterns, from aesthetically beautiful to intractably chaotic. In this note we discuss some of these aspects.

1 Background and context

The background and context of this note is the work done during the last three years in the framework of NWO BRICKS-project *Infinity*, a cooperation project of VU Amsterdam, CWI Amsterdam, and Utrecht University. The main objective of that work is to study infinite objects, typically given by recursion equations, from various points of view, to wit, proof theory (find complete proof systems and their relations), rewrite theory (in particular infinitary rewriting), and coalgebraic techniques. The present authors at the VU and the UU were especially concerned with the specialized area of infinite streams over the natural numbers or booleans $0, 1$. In particular we zoomed in on the problem of recognizing the well-definedness of recursive stream specifications, better known as 'productivity' after a proposal of Dijkstra [7], using familiar operators such as tail, zip, even, odd, etc. In fact we outlined a rather expressive format of such stream operators or functions, called the pure stream format, and proved decidability of productivity for recursive stream definitions staying within in the pure format.

[*]Dept. of Computer Science, Vrije Universiteit Amsterdam, e-mail: joerg@few.vu.nl .
[†]Dept. of Philosophy, Universiteit Utrecht, e-mail: clemens@phil.uu.nl .
[‡]Dept. of Computer Science, Vrije Universiteit Amsterdam, e-mail: diem@cs.vu.nl .
[§]Dept. of Computer Science, Vrije Universiteit Amsterdam, e-mail: jwk@cs.vu.nl .

2 History: streams in various disciplines

Infinite streams, also called infinite sequences, infinite words, or ω-words, are the subject of study in several disciplines. A landmark was the work of Axel Thue, who devised in 1906 infinite sequences of symbols avoiding certain simple patterns such as squares *ww* or cubes *www* where *w* is a finite word. He introduced the famous sequence 0110100110010110 ... that is obtained from the morphism $0 \to 01$, $1 \to 10$ with initial word 0. This sequence is cube-free and turned out to be ubiquitous indeed (see [1]), and was rediscovered by Marston Morse in 1921 in the mathematical context of dynamical systems and ergodic theory. The Thue–Morse sequence is also known to be an 'automatic sequence' (see [2]), and in particular it is a morphic sequence or D0L sequence. In the terminology of [2], the sequence is obtained by a 'substitution', another word for morphism.

So infinite streams (we will often just call them streams) arise in theoretical computer science, in particular in the areas of formal languages and combinatorics, and also in mathematics, with applications in dynamical systems and number theory. They also appear on the more practical side of computer science where functional programming languages reside [17].

3 Productivity

The notion of productivity (sometimes also referred to as liveness) was first mentioned by Dijkstra [7]. Since then several papers [18, 15, 5, 11, 17, 4] have been devoted to criteria ensuring productivity. Technically, the common essence of these approaches is a quantitative analysis, in terms of a quantitative input/output behaviour of a stream function f by a 'modulus of production' $\nu_f : \mathbb{N}^k \to \mathbb{N}$ with the property that the first $\nu_f(n_1, \ldots, n_k)$ elements of $f(t_1, \ldots, t_k)$ can be computed whenever the first n_i elements of t_i are defined. We have adopted and elaborated this approach; in [9, 10, 8] we describe a calculus by means of which we can compute composition, infimum and fixed points of periodically increasing production functions.

A general observation is that productivity is in some sense the infinitary analogon of finitary termination, a notion that is widely studied in term rewriting, and for which there nowadays exists an extensive tool technology.

Let us now consider two easy examples, that illustrate the essence of the productivity problem. A specification of the Thue–Morse stream is given in Figure 1. For this specification it is fairly easy to see, or even to prove, that it is productive in the sense that unfolding (by infinitary rewriting [12], to be precise) the definitions will never stagnate, but will 'always eventually' produce more elements of the stream. In short, the specification is productive.

Now let us consider a second example:

$$\mathsf{J} = 0 : 1 : \mathsf{even}(\mathsf{J}) \tag{1}$$

$$\mathsf{M} = 0 : \mathsf{zip}(\mathsf{inv}(\mathsf{M}), \mathsf{tail}(\mathsf{M}))$$

$$\begin{aligned}
\mathsf{tail}(x : \sigma) &= \sigma & \mathsf{zip}(x : \sigma, \tau) &= x : \mathsf{zip}(\tau, \sigma)\\
\mathsf{even}(x : \sigma) &= x : \mathsf{odd}(\sigma) & \mathsf{inv}(0 : \sigma) &= 1 : \mathsf{inv}(\sigma)\\
\mathsf{odd}(x : \sigma) &= \mathsf{even}(\sigma) & \mathsf{inv}(1 : \sigma) &= 0 : \mathsf{inv}(\sigma)
\end{aligned}$$

Figure 1: *A pure specification for the Thue–Morse stream.*

where even is defined as before. Although at first glance this specification looks perfectly OK, it nevertheless is not productive: it only produces 4 elements and then continues with infinitely long unproductive calculations. It will produce an infinitary normal form, namely $0:1:0:0:\mathsf{even}^\omega$, but this is not good enough: we want infinite normal forms built from constructors only, not including unevaluated function calls.

These two examples are setting the scene. In more complicated specifications it is not easy to see whether they are productive or not. In fact, the notion is in general undecidable. So we have to find a restricted format for which productivity is still decidable, but such that it is still sufficiently expressive.

In [9, 8, 10] we have defined and analyzed such a restrictive yet expressive format. We will refrain here from stating the full technical definition, but a good first impression is given by the examples in this section. Basically, the specification is a layered one: in the top-layer, there is a recursive definition of the intended stream constant, like M; the recursive definition may actually be a system of recursive equations. In the second layer, a declaration of stream functions is given, like zip. In the third and last layer, we declare the data functions involved. The whole specification consists of orthogonal rewrite rules, thereby guaranteeing confluence, and other useful properties.

Now for this restricted format we do have decidability. And not only theoretical decidability, but practical decidability. We have developed a tool, available at `http://infinity.few.vu.nl/productivity/` that accepts such specifications, and yields the verdict "productive", or "not productive, only producing n elements". The tool produces a pdf file with the productivity analysis in full detail. An example for Thue–Morse is in Figure 2; an example for the unproductive stream specification J is in Figure 3.

Let us consider a variation on the specification for M given in Figure 1. There the use of the 'fine' definition of zip is crucial. If we replace the definition of zip by the 'coarser' one: $\mathsf{zip}^*(x : \sigma, y : \tau) \to x : y : \mathsf{zip}^*(\sigma, \tau)$, then the specification produces only one element. The reason is that in rewrite sequences starting from M, the second argument of zip^* will never match with a stream constructor. The constant M rewrites, in the limit, to an infinite term with no reducible expressions in it, and hence to an infinite normal form. However, as mentioned before, due to the stacking of unevaluated function calls, this is not a *constructor normal form* as required for productivity. The altered specification therefore is not productive. This shows that one has to be careful when replacing stream functions by variants that are 'extensionally equivalent'; the

$$
\begin{aligned}
[\mathsf{M}] &= \mu\mathrm{M}.\bullet([\mathsf{zip}]([\mathsf{inv}](\mathrm{M}), [\mathsf{tail}](\mathrm{M}))) \\
&= \mu\mathrm{M}.\bullet(\triangle(\mathsf{box}(\overline{-++}, \mathsf{box}(\overline{-+}, \mathrm{M})), \mathsf{box}(\overline{+-+}, \mathsf{box}(-\overline{-+}, \mathrm{M})))) \\
&\twoheadrightarrow_{\mathsf{R}} \mu\mathrm{M}.\bullet(\triangle(\mathsf{box}(\overline{-++}, \mathrm{M}), \mathsf{box}(+-\overline{-++}, \mathrm{M}))) \\
&\twoheadrightarrow_{\mathsf{R}} \mu\mathrm{M}.\mathsf{box}(+\overline{-+}, \triangle(\mathsf{box}(\overline{-++}, \mathrm{M}), \mathsf{box}(+-\overline{-++}, \mathrm{M}))) \\
&\twoheadrightarrow_{\mathsf{R}} \mu\mathrm{M}.\triangle(\mathsf{box}(+\overline{-+}, \mathsf{box}(\overline{-++}, \mathrm{M})), \mathsf{box}(+\overline{-+}, \mathsf{box}(+-\overline{-++}, \mathrm{M}))) \\
&\twoheadrightarrow_{\mathsf{R}} \mu\mathrm{M}.\triangle(\mathsf{box}(\overline{+-+}, \mathrm{M}), \mathsf{box}(++-\overline{-++}, \mathrm{M})) \\
&\twoheadrightarrow_{\mathsf{R}} \triangle(\mu\mathrm{M}.\mathsf{box}(\overline{+-+}, \mathrm{M}), \mu\mathrm{M}.\mathsf{box}(++-\overline{-++}, \mathrm{M})) \\
&\twoheadrightarrow_{\mathsf{R}} \triangle(\mathsf{src}(\infty), \mathsf{src}(\infty)) \\
&\twoheadrightarrow_{\mathsf{R}} \mathsf{src}(\infty)
\end{aligned}
$$

Figure 2: *Output of our productivity decision tool: a computation yielding that evaluation of the stream constant* M *in the specification of Figure 1 can generate infinitely many data-elements, establishing the specification's productivity.*

$$
\begin{aligned}
[\mathsf{J}] &= \mu\mathrm{J}.\bullet(\bullet(\mathsf{box}(\overline{-+-}, \mathrm{J}))) \twoheadrightarrow_{\mathsf{R}} \mu\mathrm{J}.\mathsf{box}(+\overline{-+}, \mathsf{box}(+\overline{-+}, \mathsf{box}(\overline{-+-}, \mathrm{J}))) \\
&\twoheadrightarrow_{\mathsf{R}} \mu\mathrm{J}.\mathsf{box}(++\overline{-+}, \mathsf{box}(\overline{-+-}, \mathrm{J})) \twoheadrightarrow_{\mathsf{R}} \mu\mathrm{J}.\mathsf{box}(++\overline{-+-}, \mathrm{J}) \twoheadrightarrow_{\mathsf{R}} \mathsf{src}(4)
\end{aligned}
$$

Figure 3: *For the specification* (1) *we obtain that* J *is not productive (only 4 elements can be evaluated).*

property of productivity is sensitive to such replacements, due to the 'intensional' aspect of such stream specifications.

4 Complexity

Another challenging cluster of questions concerns the logical complexity, in terms of the classical arithmetic and analytical hierarchy, of various notions involved in stream specifications. The arithmetical hierarchy classifies sets by the complexity of first-order formulas describing them, which in turn is defined as the number of quantifiers of the prenex normal form. The analytical hierarchy continues the classification using second-order formulas.

We present two of the main facts:

(i) *Productivity is* Π^0_2. The problem of deciding whether a given orthogonal term rewriting system is productive, is Π^0_2-complete — a level of the arithmetical hierarchy —, and thereby equivalent to the well-known uniform halting problem for Turing machines.

(ii) *Infinitary normalization is* Π^1_1. A term rewriting system is called *infinitary normalizing* if all (possibly transfinitely long) rewrite sequences end in a (possibly infinite) normal form; the counterpart of normalization when

considering infinite terms. The complexity of this property exceeds the arithmetical hierarchy and thereby classical first-order theory. Its precise complexity is Π^1_1, a level of the analytical hierarchy.

Both results are taken from work in progress. The result in item (i) has been obtained by Endrullis, Grabmayer and Hendriks, item (ii) by Endrullis, Geuvers and Zantema.

5 Comparing Streams

How can we compare streams? Not with respect to recursion-theoretic complexity, because the streams we are interested in are all computable. Some of the tools that come to mind are Kolmogorov complexity, and the technical notion known as 'subword complexity'. With respect to this measure morphic streams (such as Thue–Morse, Toeplitz) have complexity at most quadratic, whereas the subfamily of sturmian streams, to which the Fibonacci stream belongs, have the lowest possible complexity, namely $n + 1$.

We will consider another approach. As we have seen, the streams Thue–Morse and Toeplitz are strongly related, they are twin brothers, the one can be easily converted into the other. From Thue–Morse to Toeplitz this is just taking the differences of consecutive elements modulo 2, call this operation diff, and the other way around undiff is equally simple. (Actually, there are two undiff's, undiff_0 and undiff_1, depending on choosing the first element.) Both transformations can be easily defined in our framework.

Another way of defining these transformation operations such as diff is by means of a finite state transducer (FST), in which we can read in, starting at the unique root, an infinite word σ, on the way recording how each symbol of σ, depending on the current state that we reached in the FST, is transformed into a finite (possibly empty) word.

An example is M and $\mathsf{M}/3$, where M is the Morse stream and $\mathsf{M}/3$ is the stream obtained by taking every third element. Then it is not hard to find FST's to transform M into $\mathsf{M}/3$ and $\mathsf{M}/3$ back into M. We therefore define that the 'degree' of M and $\mathsf{M}/3$ is the same. An interesting result obtained by Sebastian Stern in his recent master thesis [16] at the VU, is that every arithmetical subsequence (in fact some more) of Morse either is eventually periodic, or is still equivalent to Morse, in that it can be transformed back to Morse.

To make a connection with the turtle trajectories: the turtles can be seen as FST's where stream symbols are uniformly translated into machine instructions such as turn, etc.

To wind up this story, we get a partial order of degrees of streams, where a degree is an equivalence class of streams modulo the equivalence obtained by streams being interconvertible via FST's, see Figure 4. The trivial degree is the degree 0 of eventually periodic streams. We get an ordering between degrees in a straightforward way. Of special interest to us are minimal non-trivial degrees (call them 'prime' degrees), that have the property that there is

no non-trivial degree less than that degree. Our favourite conjecture is that the degree of Morse is such a prime degree. Prime degrees are not hard to find, e.g. one is the degree generated by the stream 1101001000100001000001 . . ..

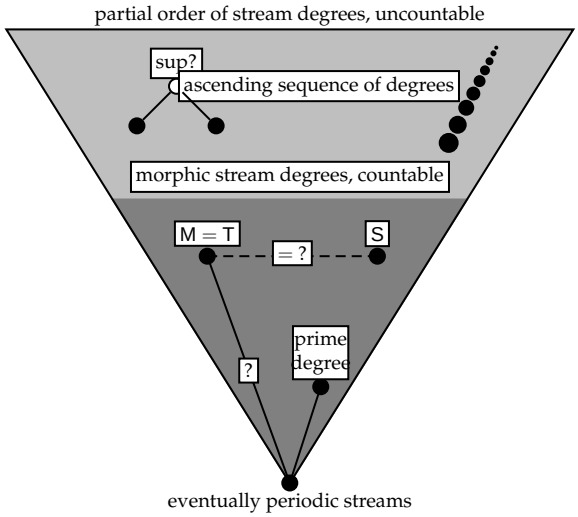

Figure 4: *The partial order of degrees of streams.*

6 Graphics

Recently, a surprising connection was discovered by Holdener and Ma [14], who showed that there is a strong connection beween the Thue–Morse stream and the famous snowflake of Helge von Koch [13], also from 1906. The discovery was done by using 'turtle graphics', an endeavour that was used in the 1980's for didactical purposes: let a turtle with a writing head and a tiny memory draw a trajectory in the plane according to a simple program describing the elementary drawing steps. In the present case, Holdener and Ma gave the turtle as program the Thue–Morse sequence, where a 0 was interpreted as: draw a line segment of one unit length in the direction in which the write head is positioned, 0 turn the write head over $\frac{\pi}{3}$. The first 50 or so steps a trajectory is drawn that crosses itself or overlaps itself, but does not evoke many associations. However, when the drawing is continued, sometimes scaling back the figure when it spills over the edges of the screen, a most remarkable phenomenon appears: the trajectory starts to resemble the Koch snowflake. And indeed, in the limit, one obtains precisely the snowflake. The limiting process is interesting, in that it uses the Hausdorff metric.

Subsequently, it was pointed out by Allouche [3] that such a connection between Thue–Morse and Koch was already implicit present in the work of

F.M. Dekking [6], in the terminology of exponential sums. An even simpler rendering was noticed by the present authors: the Toeplitz stream T, that is the stream of 'first differences' of the Thue–Morse stream, considered as a turtle program, gives the Koch snowflake right away; see Figure 10. The drawing instructions are clear from the figure. Note that the Koch snowflake nicely connects the Thue–Morse stream and the Toeplitz stream: the first can be read off above the snowflake, the second below the snowflake.

Another nice exercise is to consider the Sierpinski curve, see Figure 5, with

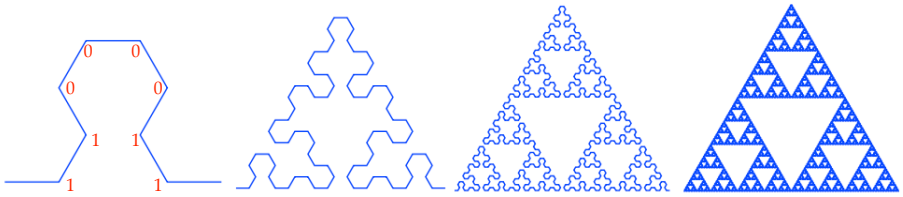

Figure 5: Construction of the Sierpinski triangle.

its associated 0-1-stream, and find a neat definition of that stream. It turns out that the Sierpinski stream, as we will call it, belongs to a familiar family of streams, namely the Toeplitz streams, that are in a simple sense 'self-generating'. Triggered by these turtle drawings, we generated several thousands of such drawings, for various streams and various drawing instructions.

Several streams yield chaotic pictures, depending on the turtle instructions. For the Fibonacci stream defined by the morphism $0 \to 01$, $1 \to 0$ starting on 0, we often find regular, aesthetically pleasing patterns, see Figure 9.

An example of a stream whose turtle trajectories are chaotic is the Kolakoski stream. This stream K is identical to the sequence of its 'run-lengths', that is, $\mathsf{K} = 22\ 11\ 2\ 1\ 22\ 1\ 22\ 11\ldots$, where the run-lengths of the alternating blocks of similar symbols are $2\ 2\ 1\ 1\ 2\ 1\ 2\ 2\ldots$. The Kolakoski stream can also be specified in the format of [8], as follows:

$$\mathsf{K} = \mathsf{f}_2(2 : \mathsf{tail}(\mathsf{K}))$$
$$\mathsf{f}_1(1:\sigma) = 1 : \mathsf{f}_2(\sigma) \qquad \mathsf{f}_1(2:\sigma) = 1 : 1 : \mathsf{f}_2(\sigma)$$
$$\mathsf{f}_2(1:\sigma) = 2 : \mathsf{f}_1(\sigma) \qquad \mathsf{f}_2(2:\sigma) = 2 : 2 : \mathsf{f}_1(\sigma)$$

For the turtle trajectory for K we find very chaotic patterns, see Figure 6. This stream is the subject of many open problems.

7 Problems

We conclude by mentioning a number of areas for further developments:

(i) *Integration into functional languages environments*: In collaboration with people from the functional programming community, we want to examine whether the results on automated recognition of stream productivity [9, 10] can be used to improve compilers.

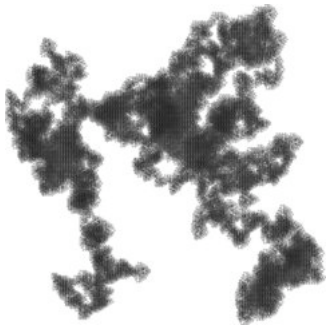

Figure 6: *A turtle trajectory for the Kolakoski sequence* K *for a prefix of* $2 \cdot 10^6$ *entries.*

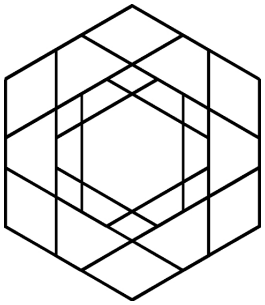

Figure 7: *A turtle trajectory for the Toeplitz stream, which can be obtained by the morphism* $0 \to 11$, $1 \to 10$ *on the initial word* 1.

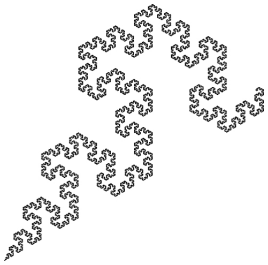

Figure 8: *A turtle trajectory for the Mephisto Waltz, the stream which can be obtained from the morphism* $0 \to 001$, $1 \to 110$ *on the initial word* 0.

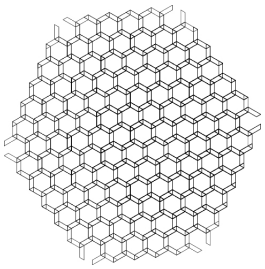

Figure 9: *A turtle trajectory for the Fibonacci stream.*

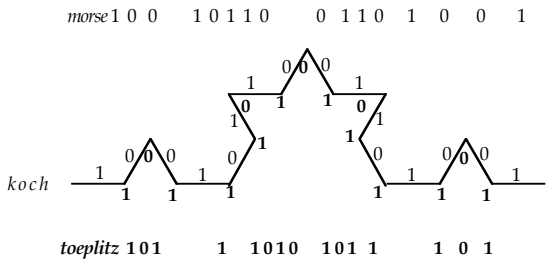

Figure 10: *Construction of the Koch snowflake from the Toeplitz stream.*

(ii) *Relation between productivity and unique definedness*: There is a close connection between productivity of a stream specification $\mathcal{S}$ and the property of $\mathcal{S}$ to have a unique solution. It is easy to see that productivity implies unique solvability, but the converse direction fails in general. Here we mention a recent interesting result by Zantema giving a criterion for unique solvability in terms of finitary termination.

(iii) *Connection with finitary termination*: There is a large arsenal of termination methods in the finitary setting, which can be applied in various ways (for example by the result mentioned in (ii)) to stream specifications.

(iv) *Connection with with lambda calculus theory of Ω-free Böhm Trees*, i.e. totally defined Böhm Trees.

(v) *Graphical aspects*: We aim to study the connections between the graphical aspects of turtle renderings of streams, and the complexity properties of the streams. At present this is 'terra incognita' for us.

References

[1] J.-P. Allouche and J. Shallit. The Ubiquitous Prouhet–Thue–Morse Sequence. In *Sequences and Their Applications: Proceedings of SETA '98*, pages 1–16. Springer–Verlag, 1999.

[2] J.-P. Allouche and J. Shallit. *Automatic Sequences: Theory, Applications, Generalizations*. Cambridge University Press, New York, 2003.

[3] J.-P. Allouche and G. Skordev. Von Koch and Thue–Morse revisited, 2006.

[4] W. Buchholz. A Term Calculus for (Co-)Recursive Definitions on Stream-like Data Structures. *Annals of Pure and Applied Logic*, 136(1-2):75–90, 2005.

[5] Th. Coquand. Infinite Objects in Type Theory. In H. Barendregt and T. Nipkow, editors, *TYPES*, volume 806, pages 62–78. Springer-Verlag, Berlin, 1994.

[6] F.M. Dekking. On the distribution of digits in arithmetic sequences. In *Séminaire de Théorie des Nombres de Bordeaux*, volume 12, pages 3201–3212, 1983.

[7] E.W. Dijkstra. On the Productivity of Recursive Definitions, 1980. EWD749, available at `http://www.cs.utexas.edu/users/EWD/`.

[8] J. Endrullis, C. Grabmayer, and D. Hendriks. Data-Oblivious Stream Productivity. In *Logic for Programming, Artificial intelligence and Reasoning 2008*, number 5330 in LNCS, pages 79–96. Springer, 2008.

[9] J. Endrullis, C. Grabmayer, D. Hendriks, A. Isihara, and J.W. Klop. Productivity of Stream Definitions. In *Proceedings of FCT 2007*, number 4639 in LNCS, pages 274–287. Springer, 2007.

[10] J. Endrullis, C. Grabmayer, D. Hendriks, A. Isihara, and J.W. Klop. Productivity of Stream Definitions. Technical Report Preprint 268, Logic Group Preprint Series, Department of Philosophy, Utrecht University, 2008. Accepted for publication in a forthcoming special issue of TCS. Available at `http://www.phil.uu.nl/preprints/lgps/`.

[11] J. Hughes, L. Pareto, and A. Sabry. Proving the Correctness of Reactive Systems Using Sized Types. In *POPL '96*, pages 410–423, 1996.

[12] J.W. Klop and R.C. de Vrijer. Infinitary Normalization. In S. Artemov, H. Barringer, A.S. d'Avila Garcez, L.C. Lamb, and J. Woods, editors, *We Will Show Them: Essays in Honour of Dov Gabbay (2)*, pages 169–192. College Publications, 2005.

[13] H. von Koch. Une méthode géométrique élementaire pour l'étude de certaines questions de la theorie des courbes planes. In *Acta Math.*, volume 30, pages 145–174, 1906.

[14] J. Ma and J.A. Holdener. When Thue–Morse Meets Koch. In *Fractals: Complex Geometry, Patterns, and Scaling in Nature and Society*, volume 13, pages 191–206, 2005.

[15] B.A. Sijtsma. On the Productivity of Recursive List Definitions. *ACM Transactions on Programming Languages and Systems*, 11(4):633–649, 1989.

[16] S. Stern. The Thue–Morse Sequence. Master's thesis, Vrije Universiteit Amsterdam, 2008.

[17] A. Telford and D. Turner. Ensuring Streams Flow. In *AMAST*, pages 509–523, 1997.

[18] W.W. Wadge. An Extensional Treatment of Dataflow Deadlock. *TCS*, 13:3–15, 1981.