2012 27th Annual ACM/IEEE Symposium on Logic in Computer Science

Automatic Sequences and Zip-Specifications

Clemens Grabmayer
Department of Philosophy
Utrecht University
Utrecht, The Netherlands
Email: clemens@phil.uu.nl

Jörg Endrullis
Department of Computer Science
VU University Amsterdam
Amsterdam, The Netherlands
Email: j.endrullis@vu.nl

Dimitri Hendriks
Department of Computer Science
VU University Amsterdam
Amsterdam, The Netherlands
Email: r.d.a.hendriks@vu.nl

Jan Willem Klop
Department of Computer Science
VU University Amsterdam
Amsterdam, The Netherlands
Email: j.w.klop@vu.nl

Lawrence S. Moss
Department of Mathematics
Indiana University
Bloomington, USA
Email: lsm@cs.indiana.edu

***Abstract*—We consider infinite sequences of symbols, also known as streams, and the decidability question for equality of streams defined in a restricted format. (Some formats lead to undecidable equivalence problems.) This restricted format consists of prefixing a symbol at the head of a stream, of the stream function 'zip', and recursion variables. Here 'zip' interleaves the elements of two streams alternatingly. The celebrated Thue–Morse sequence is obtained by the succinct 'zip-specification'**

$$\mathsf{M} = 0 : \mathsf{X} \qquad \mathsf{X} = 1 : \mathsf{zip}(\mathsf{X}, \mathsf{Y}) \qquad \mathsf{Y} = 0 : \mathsf{zip}(\mathsf{Y}, \mathsf{X})$$

The main results are as follows. We establish decidability of equivalence of zip-specifications, by employing bisimilarity of observation graphs based on a suitably chosen cobasis. Furthermore, our analysis, based on term rewriting and coalgebraic techniques, reveals an intimate connection between zip-specifications and automatic sequences. This leads to a new and simple characterization of automatic sequences. The study of zip-specifications is placed in a wider perspective by employing observation graphs in a dynamic logic setting, yielding yet another alternative characterization of automatic sequences.

By the first characterization result, zip-specifications can be perceived as a term rewriting syntax for automatic sequences. For streams σ the following are equivalent: (a) σ can be specified using zip; (b) σ is 2-automatic; and (c) σ has a finite observation graph using the cobasis $\langle \mathsf{hd}, \mathsf{even}, \mathsf{odd} \rangle$. Here even and odd are defined by $\mathsf{even}(a : s) = a : \mathsf{odd}(s)$, and $\mathsf{odd}(a : s) = \mathsf{even}(s)$. The generalization to zip-k specifications (with zip-k interleaving k streams) and to k-automaticity is straightforward.

As a natural extension of the class of automatic sequences, we also consider 'zip-mix' specifications that use zips of different arities in one specification. The corresponding notion of automaton employs a state-dependent input-alphabet, with a number representation $(n)_A = d_m \dots d_0$ where the base of digit d_i is determined by the automaton A on input $d_{i-1} \dots d_0$.

Finally we show that equivalence is undecidable for a simple extension of the zip-mix format with projections analogous to even and odd.

***Index Terms*—Automatic sequences, term rewriting, coalgebra, dynamic logic, streams, equational specifications.**

This research has been partially funded by the Netherlands Organisation for Scientific Research (NWO) under grant numbers 612.000.935, 639.021.020, and 612.000.934.

I. Introduction

Infinite sequences of symbols, also called 'streams', are a playground of common interest for logic, computer science (functional programming, formal languages, combinatorics on infinite words), mathematics (numerations and number theory, fractals) and physics (signal processing). For logic and theoretical computer science this interest focuses in particular on unique solvability of systems of recursion equations defining streams, expressivity of specification formats, and productivity (does a stream specification indeed unfold to its intended infinite result without stagnation). In addition, there is the 'infinitary word problem': when do two stream specifications over a first-order signature define the same stream? And, is that question decidable? If not, what is the logical complexity?

Against this general background, we can now situate the actual content of this paper. In the landscape of streams there are some well-known families, with automatic sequences [2] as a prominent family, including members such as the Thue–Morse sequence [1]. Such sequences are defined in first-order signature that includes some basic stream functions such as hd (head), tl (tail), ':' (prefixing a symbol to an infinite stream), even, odd; all these are familiar from any functional programming language.

One stream function in particular is frequently used in stream specifications. This is the zip function, that 'zips' the elements of two streams in alternating order, starting with the first stream. Now there is an elegant definition of the Thue–Morse sequence M using only this function zip, next to prefixing an element, and of course recursion variables:

$$\mathsf{M} = 0 : \mathsf{X} \qquad \mathsf{X} = 1 : \mathsf{zip}(\mathsf{X}, \mathsf{Y}) \qquad \mathsf{Y} = 0 : \mathsf{zip}(\mathsf{Y}, \mathsf{X}) \tag{1}$$

For general term rewrite systems, stream equality is easily seen to be undecidable [18], just as most interesting properties of streams. But by adopting some restrictions in the definitional format, decidability may hold.

Thus we consider the problem whether definitions like the one of M, using only zip next to prefixing and recursion, are

1043-6871/12 $26.00 © 2012 IEEE
DOI 10.1109/LICS.2012.44

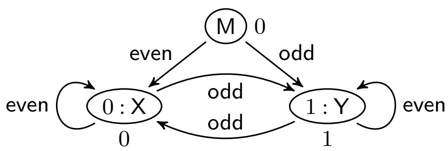


Fig. 1: Observation graph for the specification (1) *of the Thue–Morse sequence* M.

still within the realm of decidability. Answering this question positively turned out to be rewarding. In addition to solving the technical problem, the analysis leading to the solution had a useful surprise in petto: it entailed a new and simple characterization of the important notion of k-automaticity of streams. (The same 'aha-insight' was independently obtained by Kupke and Rutten, preliminary reported in [15].)

The remainder of the paper is devoted to an elaboration of several aspects concerning zip-specifications and automaticity. First, we treat a representation of automatic sequences in a framework of propositional dynamic logic, employing cobases and the ensuing observation graphs (used before for the decidability of equivalence) as the underlying semantics for a dynamic logic formula characterizing the automaticity of a stream. Second, we are led to a natural generalization of automatic sequences, corresponding to mixed zip-specifications that contain zip operators of different arities. The corresponding type of automaton employs a state-dependent alphabet. Third, we show that stream equality for a slight extension of zip-specifications is Π^0_1; the latter via a reduction from the halting problem of Fractran programs [7].

Let us now describe somewhat informally the key method that we employ to solve the equivalence problem for zip-specifications. To that end, consider the specification (1) above with root variable M. This specification is productive [20], [8], [10] and defines the Thue–Morse sequence:

$$\mathsf{M} \to^{\omega} 0:1:1:0:1:0:0:1:1:0:0:1:0:1:1:0:\ldots,$$

that is, by repeatedly applying rewrite rules that arise by orienting the equations for M, X and Y from left to right, M rewrites in the limit to the Thue–Morse sequence [1].

We will construct so-called 'observation graphs' based on the stream cobasis $\langle \mathsf{hd}, \mathsf{even}, \mathsf{odd} \rangle$ where all nodes have a double label: inside, a term corresponding to a stream (such as M and $0 : \mathsf{X}$ in Figure 1) and outside, the head of that stream. The nodes have outgoing edges to their even- and odd-derivatives. An example is shown in Figure 1.

So, the problem of equivalence of zip-specifications reduces to the problem of bisimilarity of their observation graphs, which we prove to be finite. This does not hold for observation graphs of zip-specifications with respect to the cobasis $\langle \mathsf{hd}, \mathsf{tl} \rangle$: for this cobasis, the above specification would yield an infinite observation graph. (The same would hold for any stream which is not eventually periodic.)

The observation graph in Figure 1 evokes the 'aha-insight' mentioned above: it can be recognized as a DFAO[1] (deterministic finite automaton with output) that witnesses the fact that M is a 2-automatic sequence [2].

For a full version of this extended abstract, including proofs that are omitted here, we refer to the technical report [11].

II. Zip-Specifications

For term rewriting notions see further [22]. For $k \in \mathbb{N}$ we define $\mathbb{N}_{<k} = \{0, 1, \ldots, k-1\}$. Let Δ be a finite alphabet of at least two symbols, and $\mathcal{X}$ a finite set of recursion variables.

Definition 1. The set Δ^ω of *streams over* Δ is defined by $\Delta^\omega = \{\sigma \mid \sigma : \mathbb{N} \to \Delta\}$.

We write $a : \sigma$ for the stream τ defined by $\tau(0) = a$ and $\tau(n+1) = \sigma(n)$ for all $n \in \mathbb{N}$. We define $\mathsf{hd} : \Delta^\omega \to \Delta$ and $\mathsf{tl} : \Delta^\omega \to \Delta^\omega$ by $\mathsf{hd}(x : \sigma) = x$ and $\mathsf{tl}(x : \sigma) = \sigma$.

We mix notations for syntax (term rewriting) and semantics ('real' functions), but sometimes use fonts *fun*, and fun to distinguish between functions, and term rewrite symbols.

Definition 2. For $k \in \mathbb{N}_{>0}$, the function $\mathit{zip}_k : (\Delta^\omega)^k \to \Delta^\omega$ is defined by the following rewrite rule:

$$\mathsf{zip}_k(x : \sigma_0, \sigma_1, \ldots, \sigma_{k-1}) \to x : \mathsf{zip}_k(\sigma_1, \ldots, \sigma_{k-1}, \sigma_0)$$

Thus zip_k interleaves its argument streams:

$$\mathit{zip}_k(\sigma_0, \ldots, \sigma_{k-1})(kn + i) = \sigma_i(n) \qquad (0 \le i < k)$$

Definition 3. The set $\mathcal{Z}(\Delta, \mathcal{X})$ of *zip-terms* over $\langle \Delta, \mathcal{X} \rangle$ is defined by the grammar:

$$Z ::= \mathsf{X} \mid a : Z \mid \mathsf{zip}_k(\underbrace{Z, \ldots, Z}_{k \text{ times}}) \quad (\mathsf{X} \in \mathcal{X}, a \in \Delta, k \in \mathbb{N})$$

A *zip-specification* $\mathcal{S}$ over $\langle \Delta, \mathcal{X} \rangle$ consists of a distinguished variable $\mathsf{X}_0 \in \mathcal{X}$ called the *root of* $\mathcal{S}$, and for every $\mathsf{X} \in \mathcal{X}$ a pair $\langle \mathsf{X}, t \rangle$ with $t \in \mathcal{Z}(\Delta, \mathcal{X})$ a zip-term. We treat these pairs are term rewrite rules, and write them as equations $\mathsf{X} = t$.

Definition 4. For $k \in \mathbb{N}$, the set $\mathcal{Z}_k(\Delta, \mathcal{X})$ of *zip-k terms* is the restriction of $\mathcal{Z}(\Delta, \mathcal{X})$ to terms where for every occurrence of a symbol zip_ℓ ($\ell \in \mathbb{N}$) it holds that $\ell = k$.

A *zip-k specification* is a zip-specification such that for all equations $\mathsf{X} = t$ it holds that $t \in \mathcal{Z}_k(\Delta, \mathcal{X})$.

We always assume for zip-specifications $\mathcal{S}$ that every recursion variable is *reachable from the root* X_0.

A. Unique Solvability, Productivity and Leftmost Cycles

Definition 5. A *valuation* is a mapping $\alpha : \mathcal{X} \to \Delta^\omega$. Such a valuation α extends to $[\![\cdot]\!]_\alpha : \mathcal{Z}(\Delta, \mathcal{X}) \to \Delta^\omega$ as follows:

$$[\![\mathsf{X}]\!]_\alpha = \alpha(\mathsf{X})$$
$$[\![a : t]\!]_\alpha = a : [\![t]\!]_\alpha$$
$$[\![\mathsf{zip}_k(t_1, \ldots, t_k)]\!]_\alpha = \mathit{zip}_k([\![t_1]\!]_\alpha, \ldots, [\![t_k]\!]_\alpha)$$

A *solution* for a zip-specification $\mathcal{S}$ is a valuation $\alpha : \mathcal{X} \to \Delta^\omega$, denoted $\alpha \models \mathcal{S}$, such that $[\![\mathsf{X}]\!]_\alpha = [\![t]\!]_\alpha$ for all $\mathsf{X} = t \in \mathcal{S}$.

A zip-specification $\mathcal{S}$ is *uniquely solvable* if there is a unique solution α for $\mathcal{S}$; then we let $[\![\cdot]\!]^{\mathcal{S}} = \alpha$ denote this solution.

[1]The bisimulation collapse of the graph in Fig. 1 identifies the states labeled M and $0 : \mathsf{X}$, giving rise to the familiar (minimal) DFAO for M.

Definition 6. Let $\mathcal{S}$ and $\mathcal{S}'$ be zip-specifications with roots X_0 and X_0', respectively. Then $\mathcal{S}$ is called *equivalent* to $\mathcal{S}'$ if they have the same set of solutions for their roots:

$$\{ [\![\mathsf{X}_0]\!]_\alpha \mid \alpha \models \mathcal{S} \} \; = \; \{ [\![\mathsf{X}_0']\!]_{\alpha'} \mid \alpha' \models \mathcal{S}' \}$$

Definition 7. A zip-specification $\mathcal{S}$ with root X_0 is *productive* if there exists a reduction of the form $\mathsf{X}_0 \to^* a_1 : \ldots : a_n : t$ for all $n \in \mathbb{N}$. If a zip-specification $\mathcal{S}$ is productive, then $\mathcal{S}$ is said to *define the stream* $[\![\mathsf{X}_0]\!]^{\mathcal{S}}$ where X_0 is the root of $\mathcal{S}$.

Note that if a specification is productive, then by confluence of orthogonal term rewrite systems [22], there exists a rewrite sequence of length ω that converges towards an infinite stream term $a_1 : a_2 : a_3 : \ldots$ in the limit.

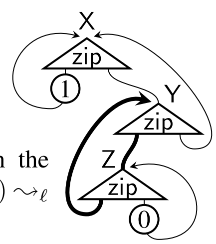
While productivity is undecidable [9], [21] for term rewrite systems in general, zip-specifications fall into the class of 'pure stream specifications' [8], [10] for which (automated) decision procedures exist. However, the latter would be taking a sledgehammer to crack a nut. For zip-specifications, productivity boils down to a simple syntactic criterion.

Definition 8. Let $\mathcal{S}$ be a zip-specification. A *step in* $\mathcal{S}$ is pair of terms $\langle s, t \rangle$, denoted by $s \rightsquigarrow t$, such that (a) $s \to t \in \mathcal{S}$, (b) $s = a : t$, or (c) $s = \mathsf{zip}_k(\ldots, t, \ldots)$. A *guard* is a step of form (b). A *left-step* $s \rightsquigarrow_\ell t$ *in* $\mathcal{S}$ is a step $s \rightsquigarrow t$ of the form (a), (b) or (c') $s = \mathsf{zip}_k(t, \ldots)$.

A *cycle in* $\mathcal{S}$ is a sequence $t_1, t_2, \ldots, t_n$ such that $t_1 = t_n \in \mathcal{X}$ and $t_i \rightsquigarrow t_{i+1}$ for $1 \le i < n$. A *leftmost cycle in* $\mathcal{S}$ is a cycle $t_1, t_2, \ldots, t_n$ such that $t_i \rightsquigarrow_\ell t_{i+1}$ for $1 \le i < n$.

Example 9. Consider the following specification

$$\mathsf{X} = \mathsf{zip}(1 : \mathsf{X}, \mathsf{Y})$$
$$\mathsf{Y} = \mathsf{zip}(\mathsf{Z}, \mathsf{X})$$
$$\mathsf{Z} = \mathsf{zip}(\mathsf{Y}, 0 : \mathsf{Z})$$



visualized as the cyclic term graph on the right. The leftmost cycle $\mathsf{Y} \rightsquigarrow_\ell \mathsf{zip}(\mathsf{Z}, \mathsf{X}) \rightsquigarrow_\ell \mathsf{Z} \rightsquigarrow_\ell \mathsf{zip}(\mathsf{Y}, 0 : \mathsf{Z}) \rightsquigarrow_\ell \mathsf{Y}$ is not guarded.

For orthogonal term rewriting systems, productivity implies the uniqueness of solutions, but unique solvability is not sufficient for productivity. For zip-specifications it turns out that both concepts coincide. Here we need that Δ is not a singleton — otherwise every specification has a unique solution.

Theorem 10. *For zip-specifications $\mathcal{S}$ these are equivalent:*
(i) $\mathcal{S}$ is uniquely solvable.
(ii) $\mathcal{S}$ is productive.
(iii) $\mathcal{S}$ has a guard on every leftmost cycle.

B. Evolving and Solving Zip-Specifications

The key to the proof of Theorem 10 consists of a transformation of zip-specifications by (i) simple equational logic steps, and (ii) internal rewrite steps.

Definition 11. For zip-specifications $\mathcal{S}, \mathcal{S}'$ we say $\mathcal{S}$ *evolves to* $\mathcal{S}'$, denoted by $\mathcal{S} \circlearrowright \mathcal{S}'$, if one of the conditions holds:

(i) $\mathcal{S}$ contains an equation $\mathsf{X} = a : t$ with $\mathsf{X} \neq \mathsf{X}_0$ and $\mathcal{S}'$ is obtained from $\mathcal{S}$ by: let X' be fresh and
  (a) exchange the equation $\mathsf{X} = a : t$ for $\mathsf{X}' = t$, then
  (b) replace all X in all right-hand sides by $a : \mathsf{X}'$, and
  (c) finally rename X' to X (X is no longer used).

(ii) $\mathcal{S}$ contains an equation $\mathsf{X} = t$ such that t rewrites to t' via a zip-rule (Definition 2), and $\mathcal{S}'$ is obtained from $\mathcal{S}$ by replacing the equation $\mathsf{X} = t$ with $\mathsf{X} = t'$.

The condition $\mathsf{X} \neq \mathsf{X}_0$ in clause (i) guarantees that the meaning (its solution) is preserved under evolving. It prevents transforming a specification like $\mathsf{X}_0 = 0 : 1 : \mathsf{X}_0$ into $\mathsf{X}_0 = 1 : 0 : \mathsf{X}_0$ which clearly has a different solution.

Lemma 12. *Let $\mathcal{S} \circlearrowright \mathcal{S}'$. Then for every $\alpha : \mathcal{X} \to \Delta^\omega$ it holds that α is a solution of $\mathcal{S}$ if and only if α is a solution of $\mathcal{S}'$. Moreover, if $\mathcal{S}$ is productive then so is $\mathcal{S}'$.*

Definition 13. A zip-specification $\mathcal{S}$ is said to have a *free root* if the root X_0 of $\mathcal{S}$ does not occur in any right-hand side of $\mathcal{S}$.

Lemma 14. *Every zip-specification can be transformed into an equivalent one with free root.*

The following lemma relates rewriting to evolving:

Lemma 15. *Let $\mathcal{S}$ be a zip-specification with free root X_0. There exists a reduction $\mathsf{X}_0 \to^* a_1 : \ldots : a_n : t$ in $\mathcal{S}$ if and only if there exists a zip-specification $\mathcal{S}'$ such that $\mathcal{S} \circlearrowright^* \mathcal{S}'$ and $\mathcal{S}'$ contains an equation of the form $\mathsf{X}_0 = a_1 : \ldots : a_n : t'$.*

Example 16. We evolve the following specification:

$$\mathsf{X} = \mathsf{zip}(1 : \mathsf{X}, \mathsf{Y}) \quad \mathsf{Y} = \underline{0 :}\,\mathsf{tl}(\mathsf{zip}(\mathsf{Z}, \mathsf{X})) \quad \mathsf{Z} = \mathsf{zip}(\mathsf{Y}, 0 : \mathsf{Z})$$
$$\mathsf{X} = \mathsf{zip}(1 : \mathsf{X}, \overline{0 :}\,\mathsf{Y}) \quad \mathsf{Y} = \mathsf{tl}(\mathsf{zip}(\mathsf{Z}, \mathsf{X})) \quad \mathsf{Z} = \mathsf{zip}(\underline{\overline{0 :}\,\mathsf{Y}, 0 : \mathsf{Z})}$$
$$\ldots \quad \mathsf{Y} = \mathsf{tl}(\mathsf{zip}(\mathsf{Z}, \mathsf{X})) \quad \mathsf{Z} = \underline{\overline{0 :}}\,\mathsf{zip}(0 : \mathsf{Z}, \mathsf{Y})$$
$$\ldots \quad \mathsf{Y} = \mathsf{tl}(\mathsf{zip}(\underline{\overline{0 :}}\,\mathsf{Z}, \mathsf{X})) \quad \mathsf{Z} = \mathsf{zip}(0 : \overline{0 :}\,\mathsf{Z}, \mathsf{Y})$$
$$\ldots \quad \mathsf{Y} = \underline{\mathsf{tl}(\overline{0 :}}\,\mathsf{zip}(\mathsf{X}, \mathsf{Z})) \quad \mathsf{Z} = \mathsf{zip}(0 : 0 : \mathsf{Z}, \mathsf{Y})$$
$$\mathsf{X} = \mathsf{zip}(1 : \mathsf{X}, 0 : \mathsf{Y}) \quad \mathsf{Y} = \mathsf{zip}(\mathsf{X}, \mathsf{Z}) \quad \mathsf{Z} = \mathsf{zip}(0 : 0 : \mathsf{Z}, \mathsf{Y})$$

Note that the contracted redexes are underlined and the created symbols are overlined. Also note that invoking a free root is not needed for the evolution above.

Strictly speaking, the last step in the above example is not covered by Definition 11 since the rule for 'tl' is not included. We have chosen this example to demonstrate another principle. The specification we started from is obtained from Example 9 by inserting $0 : \mathsf{tl}(\ldots)$ on an unguarded leftmost cycle. Evolving has resulted in a productive zip-specification (now every leftmost cycle is guarded) that represents a solution of the original specification. Similarly, by inserting $1 : \mathsf{tl}(\ldots)$, we obtain the solution:

$$\mathsf{X} = \mathsf{zip}(1 : \mathsf{X}, 1 : \mathsf{Y}) \quad \mathsf{Y} = \mathsf{zip}(\mathsf{X}, \mathsf{Z}) \quad \mathsf{Z} = \mathsf{zip}(0 : 1 : \mathsf{Z}, \mathsf{Y})$$

The insertion of $0 : \mathsf{tl}(\ldots)$ and $1 : \mathsf{tl}(\ldots)$ corresponds to choosing whether we are interested in a solution for Y starting with head 0 or 1. To see that the result of the insertions are valid solutions it is crucial to observe that the symbol 'tl' in the inserted $a : \mathsf{tl}(\dots)$ disappears by consuming a 'descendant' of the element $a \in \Delta$. In general we have:

Lemma 17. *Let $\mathcal{S}$ be a zip-specification. Define the set $\{\mathsf{Y}_1, \dots, \mathsf{Y}_m\}$ to contain precisely one recursion variable from every unguarded leftmost cycle from $\mathcal{S}$.*

Let $\vec{a} = \langle a_1, \dots, a_m \rangle \in \Delta^m$ and define $\mathcal{S}_{\vec{a}}$ to be obtained from $\mathcal{S}$ by replacing each equation $\mathsf{Y}_i = t_i$ by $\mathsf{Y}_i = a_i : \mathsf{tl}(t_i)$. Subsequently, we can by the evolving procedure eliminate the occurrences of the symbol tl as in Example 16. Then $\mathcal{S}_{\vec{a}}$ is productive, and the unique solution $[\![\cdot]\!]^{\mathcal{S}_{\vec{a}}} : \mathcal{X} \to \Delta^\omega$ is a solution of $\mathcal{S}$. Hence, $\{\mathcal{S}_{\vec{a}} \mid \vec{a} \in \Delta^m\}$ is the set of all solutions of $\mathcal{S}$, in particular, $\mathcal{S}$ has $|\Delta|^m$ different solutions.

C. Formats of Zip-Specifications

Definition 18. A zip-specification $\mathcal{S}$ is called *flat* if each of its equations is of the form:

$$\mathsf{X}_i = c_{i,1} : \dots : c_{i,m_i} : \mathsf{zip}_{k_i}(\mathsf{X}_{i,1}, \dots, \mathsf{X}_{i,k_i}) \quad (0 \le i < n)$$

for $m_i, k_i \in \mathbb{N}$, $k_i \ge 2$, recursion variables $\mathsf{X}_i, \mathsf{X}_{i,1}, \dots, \mathsf{X}_{i,k_i}$ and data constants $c_{i,1}, \dots, c_{i,m_i}$.

Zip-free cycles correspond to periodic sequences, and these can be specified by flat zip-k specifications. Together with unfolding and introduction of fresh variables we then obtain:

Lemma 19. *Every productive zip-k specification can be transformed into an equivalent productive, flat zip-k specification.*

III. Zip-Specifications and Observation Graphs

For the decidability result and the connection with automaticity we need to observe streams and compare them. This is done with observations in terms of a cobasis and bisimulations to compare the resulting graphs.

A. Cobases, Observation Graphs, and Bisimulation

For general introductions to coalgebra we refer to [4], [19]. We first introduce the notion of 'cobasis' [17], [14]. For the sake of simplicity, we restrict to the single observation hd.

Definition 20. A *stream cobasis* $\mathcal{B} = \langle \mathsf{hd}, \langle \gamma_1, \dots, \gamma_k \rangle \rangle$ is a tuple consisting of *operations* $\gamma_i : \Delta^\omega \to \Delta^\omega$ $(1 \le i \le k)$ such that for all $\sigma, \tau \in \Delta^\omega$ it holds that $\sigma = \tau$ whenever

$$\mathsf{hd}(\gamma_{i_1}(\dots(\gamma_{i_n}(\sigma))\dots)) = \mathsf{hd}(\gamma_{i_1}(\dots(\gamma_{i_n}(\tau))\dots))$$

for all $n \in \mathbb{N}$ and $1 \le i_1, \dots, i_n \le k$.

As hd is integral part of every stream cobasis, we suppress hd and write $\langle \gamma_1, \dots, \gamma_k \rangle$ as shorthand for $\langle \mathsf{hd}, \langle \gamma_1, \dots, \gamma_k \rangle \rangle$.

Definition 21. For $i \in \mathbb{N}$, $k \in \mathbb{N}_{>0}$ define $\pi_{i,k} : \Delta^\omega \to \Delta^\omega$:

$$\pi_{0,k}(x : \sigma) \to x : \pi_{k-1,k}(\sigma) \qquad \pi_{i+1,k}(x : \sigma) \to \pi_{i,k}(\sigma)$$

For every $k \ge 2$ we define two stream cobases:

$$\mathcal{N}_k = \langle \pi_{0,k}, \dots, \pi_{k-1,k} \rangle \qquad \mathcal{O}_k = \langle \pi_{1,k}, \dots, \pi_{k,k} \rangle$$

Note that $\pi_{i,k}(\sigma)$ selects an arithmetic subsequence of σ; it picks every k-th element beginning from index i: $\pi_{i,k}(\sigma)(n) = \sigma(kn + i)$. The $\pi_{i,k}$ are generalized even and odd functions, in particular we have: $\mathsf{tl} = \pi_{1,1}$, $\mathsf{even} = \pi_{0,2}$ and $\mathsf{odd} = \pi_{1,2}$.

Observe that $\mathcal{N}_k$ and $\mathcal{O}_k$ are cobases, that is, every element of a stream can be observed. The main difference between $\mathcal{N}_k$ and $\mathcal{O}_k$ is that $\mathcal{N}_k$ has an ambiguity in naming stream entries: $\mathsf{hd}(\sigma) = \mathsf{hd}(\mathsf{even}(\sigma))$. On the other hand, $\mathcal{O}_k$ is an orthogonal basis, names of stream entries are unambiguous.

We employ the following simple coinduction principle.

Definition 22. Let $\mathcal{B} = \langle \gamma_1, \dots, \gamma_k \rangle$ be a cobasis. A *$\mathcal{B}$-bisimulation* is a relation $R \subseteq \Delta^\omega \times \Delta^\omega$ s.t. $\langle \sigma, \tau \rangle \in R$ implies $\mathsf{hd}(\sigma) = \mathsf{hd}(\tau)$ and $\langle \gamma_i(\sigma), \gamma_i(\tau) \rangle \in R$ for $1 \le i \le k$.

Lemma 23. *For all $\sigma, \tau \in \Delta^\omega$ it holds that $\sigma = \tau$ if and only if there exists a $\mathcal{B}$-bisimulation R such that $\langle \sigma, \tau \rangle \in R$.* $\square$

We now further elaborate the coalgebraic perspective. The following definition formalizes '$\mathcal{B}$-observation graphs' where $\mathcal{B} = \langle \gamma_1, \dots, \gamma_k \rangle$ is a cobasis. Every node n will represent the stream $[\![n]\!] \in \Delta^\omega$, and if the i-th outgoing edge of n points to node m then $\gamma_i([\![n]\!]) = [\![m]\!]$.

Definition 24. Let $\mathcal{B} = \langle \mathsf{hd}, \langle \gamma_1, \dots, \gamma_k \rangle \rangle$ be a stream cobasis, and let F be the functor $F(X) = \Delta \times X^k$.

A *$\mathcal{B}$-observation graph* is an F-coalgebra $\mathcal{G} = \langle S, \langle o, n \rangle \rangle$ with a distinguished *root* element $r \in S$, such that there exists an F-homomorphism $[\![\cdot]\!] : S \to \Delta^\omega$ from $\mathcal{G}$ to the F-coalgebra $\langle \Delta^\omega, \mathcal{B} \rangle$ of all streams with respect to $\mathcal{B}$:

$$\begin{tikzcd} S \arrow[r, "{[\![\cdot]\!]}"] \arrow[d, "{\langle o, n \rangle}"'] & \Delta^\omega \arrow[d, "\mathcal{B}"] \\ \Delta \times S^k \arrow[r, "{\mathrm{id} \times [\![\cdot]\!]^k}"] & \Delta \times (\Delta^\omega)^k \end{tikzcd}$$

The observation graph $\mathcal{G}$ is said to *define* the stream $[\![r]\!] \in \Delta^\omega$. (We note that $[\![\cdot]\!]$ is unique by Lemma 25, below.)

Let $\sigma \in \Delta^\omega$. The *canonical $\mathcal{B}$-observation graph of σ* is defined as the sub-coalgebra of the F-coalgebra $\langle \Delta^\omega, \mathcal{B} \rangle$ generated by σ, that is, the observation graph $\langle T, \mathcal{B} \rangle$ with root σ where $T \subseteq \Delta^\omega$ is the least set containing σ that is closed under $\gamma_1, \dots, \gamma_k$. The set $\partial_{\mathcal{B}}(\sigma)$ of *$\mathcal{B}$-derivatives of σ* is the set of elements of the canonical observation graph of σ.

Lemma 25. *For every $\mathcal{B}$-observation graph the mapping $[\![\cdot]\!]$ is unique whenever it exists.*

For the cobasis $\mathcal{O}_k$, the existence of $[\![\cdot]\!]$ is guaranteed; this result is mentioned without proof in [16, Ex. 6.2(1)]. See our extended version [11] for a proof.

Proposition 26. *The stream coalgebra $\langle \Delta^\omega, \mathcal{O}_k \rangle$ is final for the functor $F(X) = \Delta \times X^k$. As a consequence, we have that every F-coalgebra is an $\mathcal{O}_k$-observation graph.*

In contrast, the existence of $[\![\cdot]\!]$ is *not* guaranteed for $\mathcal{N}_k$. The coalgebra $\langle \Delta^\omega, \mathcal{N}_k \rangle$ is final for a subset of F-coalgebras, called *zero-consistent*, see further [15].

Definition 27. Let $\mathcal{B} = \langle \mathsf{hd}, \langle \gamma_1, \dots, \gamma_k \rangle \rangle$ be a stream cobasis. A *bisimulation* between $\mathcal{B}$-observation graphs $\mathcal{G} = \langle S, \langle o, n \rangle \rangle$

and $\mathcal{G}' = \langle S', \langle o', n' \rangle \rangle$ is a relation $R \subseteq S \times S'$ such that for all $\langle s, s' \rangle \in R$ we have that $o(s) = o'(s')$ and $\langle n_i(s), n'_i(s') \rangle \in R$ for all $1 \leq i \leq k$, where n_i denotes the i-th projection on n. Two observation graphs are *bisimilar* if there is a bisimulation relating their roots.

For deterministic transition systems, such as observation graphs, bisimilarity coincides with trace equivalence. As a consequence, the algorithm of Hopcroft–Karp [12] is applicable.

Proposition 28. *Bisimilarity of finite $\mathcal{B}$-observation graphs is decidable (in linear time with respect to the sum of the number of vertices).*

Proposition 29. *Let $\mathcal{B}$ be a stream cobasis. Two $\mathcal{B}$-observation graphs define the same stream if and only if they are bisimilar.*

B. From Zip-k Specifications To Observation Graphs

We construct observation graphs for zip-k specifications.

Definition 30. Let $\mathcal{X} = \{\mathsf{X}_0, \dots, \mathsf{X}_{n-1}\}$ be a set of recursion variables and Δ a finite alphabet (here regarded as a set of data-constants). Let $k \in \mathbb{N}$, and $\mathcal{S}$ be a zip-k specification over $\langle \Delta, \mathcal{X} \rangle$. We define the orthogonal term rewrite system $\mathcal{R}_k(\mathcal{S})$ to consist of the following rules:

$$\begin{aligned}
\mathsf{hd}(a : \sigma) &\to a \\
\pi_{0,k}(a : \sigma) &\to a : \pi_{k-1,k}(\sigma) \\
\pi_{i+1,k}(a : \sigma) &\to \pi_{i,k}(\sigma) && (0 \leq i < k+1) \\
\mathsf{hd}(\mathsf{zip}_k(\sigma_0, \dots, \sigma_{k-1})) &\to \mathsf{hd}(\sigma_0) \\
\pi_{i,k}(\mathsf{zip}_k(\sigma_0, \dots, \sigma_{k-1})) &\to \sigma_i && (0 \leq i < k)
\end{aligned}$$

and additionally for every equation $\mathsf{X}_j = t$ of $\mathcal{S}$ the rules

$$\mathsf{hd}(\mathsf{X}_j) \to \mathsf{hd}(t) \qquad \pi_{i,k}(\mathsf{X}_j) \to \pi_{i,k}(t) \quad (0 \leq i \leq k+1)$$

where the X_j are treated as constant symbols.

Whenever $\mathcal{S}$ is clear from the context, then by $t{\downarrow}$ we denote the unique normal form of term t with respect to $\mathcal{R}_k(\mathcal{S})$.

Definition 31. Let $\mathcal{S}$ be a productive, flat zip-k specification with root X_0. The set $\delta_k(\mathcal{S})$ is the least set containing X_0 that is closed under $\lambda t.(\pi_{i,k}(t){\downarrow})$ for every $0 \leq i < k$.

Definition 32. Let $\mathcal{S}$ be a productive, flat zip-k specification with root X_0. The *$\mathcal{N}_k$-observation graph* $\mathcal{G}(\mathcal{S})$ is defined as:

$$\mathcal{G}(\mathcal{S}) = \langle \delta_k(\mathcal{S}), \langle o, n \rangle \rangle \quad o(t) = \mathsf{hd}(t){\downarrow}$$
$$n(t) = \langle \pi_{0,k}(t){\downarrow}, \dots, \pi_{k-1,k}(t){\downarrow} \rangle$$

with root X_0. In words: every node t has

(i) the observation $\mathsf{hd}(t){\downarrow}$ (the label), and
(ii) outgoing edges to $\pi_{0,k}(t){\downarrow}, \dots, \pi_{k-1,k}(t){\downarrow}$ (in this order).

Lemma 33. *Let $\mathcal{S}$ be a productive, flat zip-k specification with root X_0. There exists $m \in \mathbb{N}$ such that every term in $\delta_k(\mathcal{S})$ is of the form $d_0 : \ldots : d_{\ell-1} : \mathsf{X}_j$ with $\ell \leq m$, $d_0, \dots, d_{\ell-1} \in \Delta$ and $X_j \in \mathcal{X}$. As a consequence $\delta_k(\mathcal{S})$ and $\mathcal{G}(\mathcal{S})$ are finite.*

Proof Sketch: The equations of $\mathcal{S}$ are of the form:

$$\mathsf{X}_j = c_{j,0} : \ldots : c_{j,m_j-1} : \mathsf{zip}_k(\mathsf{X}_{j,0}, \dots, \mathsf{X}_{j,k-1}) \;\; (0 \leq j < n)$$

Let $m := \max\{ m_i \mid 0 \leq i < n \}$. It suffices that the claimed shape is closed under $\lambda s.\pi_{i,k}(s){\downarrow}$ for $0 \leq i < k$. This follows by a straightforward application of Definition 30 together with a precise counting of the 'produced' elements. ■

We need to ensure that the rewrite system from Definition 30 implements (is sound for) the intended semantics; recall that $\mathcal{S}$ has a unique solution $[\![\cdot]\!]^{\mathcal{S}} : \mathcal{X} \to \Delta^\omega$ due to productivity:

Lemma 34. *Let $\mathcal{S}$ be a productive, flat zip-k specification with root X_0. For every $t \in \delta_k(\mathcal{S})$ and $0 \leq i < k$ we have that $\mathsf{hd}(t) \to^* \mathsf{hd}([\![t]\!])$ and $[\![\pi_{i,k}(t){\downarrow}]\!] = \pi_{i,k}([\![t]\!])$. Hence, the graph $\mathcal{G}(\mathcal{S})$ is an $\mathcal{N}_k$-observation graph defining $[\![\mathsf{X}_0]\!]^{\mathcal{S}}$.*

Proof: The extension of $[\![\cdot]\!]_\alpha$ from Definition 5, interpreting the symbols $\pi_{i,k}$ by the stream function $\pi_{i,k} : \Delta^\omega \to \Delta^\omega$ for every $0 \leq i < k$, is a model of $\mathcal{R}_k(\mathcal{S})$. ■

As an application of Lemmas 19 and 34 we get

Lemma 35. *For every productive zip-k specification with root X_0 we can construct an $\mathcal{N}_k$-observation graph defining the stream $[\![\mathsf{X}_0]\!]^{\mathcal{S}}$.*

We arrive at our first main result:

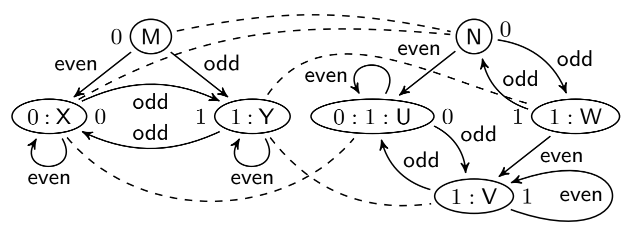
Theorem 36. *Equivalence of zip-k specifications is decidable.*

Proof: Lemma 17 allows to reduce the equivalence problem for unproductive zip-k specifications to a finite number of equivalence problems for productive zip-k specifications. Propositions 29, 28 and Lemma 35 imply decidability of equivalence for productive zip-k specifications. ■

Proposition 37. *Equivalence of productive, flat zip-specifications is decidable in quadratic time.*

Example 38. Consider the zip-2 specification with root N:

$$\mathsf{N} = 0 : \mathsf{zip}(1 : \mathsf{W}, 1 : \mathsf{U}) \qquad \mathsf{U} = 1 : \mathsf{zip}(\mathsf{V}, \mathsf{U})$$
$$\mathsf{V} = 0 : \mathsf{zip}(\mathsf{V}, 1 : \mathsf{U}) \qquad \mathsf{W} = \mathsf{zip}(\mathsf{N}, \mathsf{V})$$



Its $\mathcal{N}_2$-observation graph is depicted on the right above. The dashed lines indicate a bisimulation with the observation graph from Fig. 1 here depicted on the left.

C. From Observation Graphs To Zip-k Specifications

Lemma 39. *The canonical $\mathcal{O}_k$-observation graph of a stream $\sigma \in \Delta^\omega$ is finite if and only if σ can be defined by a zip-k specification consisting of equations of the form:*

$$\mathsf{X}_i = a_i : \mathsf{zip}_k(\mathsf{X}_{i,1}, \mathsf{X}_{i,2}, \dots, \mathsf{X}_{i,k})$$

Proof: For the translation forth and back, it suffices to observe the correspondence between an equation $\mathsf{Y} = a :$

$\mathsf{zip}_k(\mathsf{Y}_1,\ldots,\mathsf{Y}_k)$ and its semantics $\mathsf{hd}([\![\mathsf{Y}]\!]) = a$, $\pi_{1,k}([\![\mathsf{Y}]\!]) = [\![\mathsf{Y}_1]\!]$, …, $\pi_{k,k}([\![\mathsf{Y}]\!]) = [\![\mathsf{Y}_k]\!]$. ■

Lemma 40. *The canonical $\mathcal{N}_k$-observation graph of a stream $\sigma \in \Delta^\omega$ is finite if and only if σ can be defined by a zip-k specification consisting of pairs of equations of the form:*

$$\mathsf{X}_i = a_i : \mathsf{X}'_i \qquad \mathsf{X}'_i = \mathsf{zip}_k(\mathsf{X}_{f(i,1)},\ldots,\mathsf{X}_{f(i,k-1)},\mathsf{X}'_{f(i,0)})$$

over recursion variables $\mathcal{X} \cup \mathcal{X}'$ where $\mathcal{X} = \{\mathsf{X}_0,\ldots,\mathsf{X}_{n-1}\}$ and $\mathcal{X}' = \{\mathsf{X}'_i \mid \mathsf{X}_i \in \mathcal{X}\}$, and $f : \mathbb{N}_{<n} \times \mathbb{N}_{<k} \to \mathbb{N}_{<n}$ such that $a_{f(i,0)} = a_i$ for all $i \in \mathbb{N}_{<n}$.

Proof: If $\mathsf{Y} = a : \mathsf{Y}'$ and $\mathsf{Y}' = \mathsf{zip}_k(\mathsf{Y}_1,\ldots,\mathsf{Y}_{k-1},\mathsf{Y}'_0)$ then $\mathsf{hd}([\![\mathsf{Y}]\!]) = a$, $\pi_{0,k}([\![\mathsf{Y}]\!]) = a : [\![\mathsf{Y}'_0]\!]$, and $\pi_{i,k}([\![\mathsf{Y}]\!]) = [\![\mathsf{Y}_i]\!]$ $(1 \le i < k)$. Since there also is an equation $\mathsf{Y}_0 = a : \mathsf{Y}'_0$, it holds that $[\![\mathsf{Y}'_0]\!] = \mathsf{tl}([\![\mathsf{Y}_0]\!])$ and hence $\pi_{0,k}([\![\mathsf{Y}]\!]) = [\![\mathsf{Y}_0]\!]$. ■

IV. Automaticity and Observation Graphs

After our first main result (Theorem 36) we proceed with connecting zip-k specifications to k-automatic sequences.

A. Automatic Sequences

Definition 41 ([2])**.** A *deterministic finite automaton with output (DFAO)* is a tuple $\langle Q, \Sigma, \delta, q_0, \Delta, \lambda\rangle$ where

– Q is a finite set of states with $q_0 \in Q$ the initial state,
– Σ a finite input alphabet, Δ an output alphabet,
– $\delta : Q \times \Sigma \to Q$ a transition function, and
– $\lambda : Q \to \Delta$ an output function.

We extend δ to words over Σ as follows:

$$\begin{aligned} \delta(q,\varepsilon) &= q && \text{for } q \in Q \\ \delta(q,wa) &= \delta(\delta(q,a),w) && \text{for } q \in Q, a \in \Sigma, w \in \Sigma^* \end{aligned}$$

and we write $\delta(w)$ as shorthand for $\delta(q_0,w)$.

For $n,k \in \mathbb{N}$, $k \ge 2$, we use $(n)_k$ to denote the representation of n with respect to the base k (without leading zeros). More precisely, for $n > 0$ we have $(n)_k = n_m n_{m-1} \ldots n_0$ where $0 \le n_m,\ldots,n_0 < k$, $n_m > 0$ and $n = \sum_{i=0}^{m} n_i k^i$; for $n = 0$ we fix $(n)_k = \varepsilon$.

Definition 42. A k-DFAO A is a DFAO $\langle Q, \Sigma, \delta, q_0, \Delta, \lambda\rangle$ with input alphabet $\Sigma = \mathbb{N}_{<k}$. For $q \in Q$, we define a stream $\zeta(A,q)$ by: $\zeta(A,q)(n) = \lambda(\delta(q,(n)_k))$ for every $n \in \mathbb{N}$.

We write $\zeta(A)$ as shorthand for $\zeta(A,q_0)$. Moreover, we say that the automaton A *generates* the stream $\zeta(A)$.

Definition 43. A stream $\sigma : \Delta^\omega$ is called *k-automatic* if there exists a k-DFAO that generates σ. A stream is called *automatic* if it is k-automatic for some $k \ge 2$.

The exclusion of leading zeros in the number representation $(n)_k$ is not crucial for the definition of automatic sequences. Every k-DFAO can be transformed into an equivalent k-DFAO that ignores leading zeros:

Definition 44. A k-DFAO $\langle Q, \Sigma, \delta, q_0, \Delta, \lambda\rangle$ is called *invariant under leading zeros* if for all $q \in Q$: $\lambda(q) = \lambda(\delta(q,0))$.

Lemma 45 ([2, Theorem 5.2.1 with Corollary 4.3.4])**.** *For every k-DFAO A there is a k-DFAO A' that is invariant under leading zeros and generates the same stream ($\zeta(A) = \zeta(A')$).*

Automatic sequences can be characterized in terms of their ‘kernels’ being finite. Kernels of a stream σ are sets of arithmetic subsequences of σ, defined as follows.

Definition 46. The *k-kernel* of a stream $\sigma \in \Delta^\omega$ is the set of subsequences $\{\pi_{i,k^p}(\sigma) \mid p \in \mathbb{N}, i < k^p\}$.

Lemma 47 ([2, Theorem 6.6.2])**.** *A stream σ is k-automatic if and only if the k-kernel of σ is finite.*

B. Observation Graphs and Automatic Sequences

There is a close correspondence between observation graphs with respect to the cobasis $\mathcal{N}_k$ and k-DFAOs. For k-DFAOs A that are invariant under leading zeros an edge $q \to p$ labeled i implies that the stream generated by p is the $\pi_{i,k}$-projection of the stream generated by q, that is, $\zeta(A,p) = \pi_{i,k}(\zeta(A,q))$. The following lemma treats the case of general k-DFAOs.

Lemma 48. *Let $A = \langle Q, \Sigma, \delta, q_0, \Delta, \lambda\rangle$ be a k-DFAO. Then for every $q \in Q$ we have:* $\mathsf{tl}(\zeta(A,\delta(q,0))) = \mathsf{tl}(\pi_{0,k}(\zeta(A,q)))$ *and for all $1 \le i < k$:*

$$\zeta(A,\delta(q,i)) = \pi_{i,k}(\zeta(A,q)) \tag{2}$$

Hence, if A is invariant under leading zeros, then property (2) *holds for all $0 \le i < k$.*

Proof: Follows immediately from $(kn+i)_k = (n)_k i$ for all $n \in \mathbb{N}$ and $0 \le i < k$ such that $n \ne 0$ or $i \ne 0$. ■

As a consequence of Lemma 48 we have that k-DFAOs, that are invariant under leading zeros, are $\mathcal{N}_k$-observation graphs for the streams they define, and vice versa. Formally, this is just a simple change of notation[2]:

Definition 49. Let $A = \langle Q, \Sigma, \delta, q_0, \Delta, \lambda\rangle$ be a k-DFAO that is invariant under leading zeros. We define the $\mathcal{N}_k$-observation graph $\mathcal{G}(A) = \langle Q, \langle o, n\rangle\rangle$ with root q_0 where for every $q \in Q$: $o(q) = \lambda(q)$, $n_i(q) = \delta(q,i)$ for $i < k$, and $[\![q]\!] = \zeta(A,q)$.

Let $\mathcal{G} = \langle S, \langle o, n\rangle\rangle$ be an $\mathcal{N}_k$-observation graph over Δ with root $r \in S$. Then we define a k-DFAO $A(\mathcal{G})$ as follows: $A(\mathcal{G}) = \langle Q, \mathbb{N}_{<k}, \delta, q_0, \Delta, \lambda\rangle$ where $Q = S$, $q_0 = r$, and for every $s \in S$: $\lambda(s) = o(s)$, and $\delta(s,i) = n_i(s)$ for $i < k$.

Proposition 50. *For every k-DFAO A that is invariant under leading zeros, the $\mathcal{N}_k$-observation graph $\mathcal{G}(A)$ defines the stream that is generated by A.*

Conversely, we have for every $\mathcal{N}_k$-observation graph $\mathcal{G}$, that the k-DFAO $A(\mathcal{G})$ is invariant under zeros and generates the stream defined by $\mathcal{G}$.

Another way to see the correspondence between automatic sequences and their finite, canonical $\mathcal{N}_k$-observation graphs is the following. The elements of the canonical observation graph of a stream σ, that is, the set of $\{\pi_{0,k},\ldots,\pi_{k-1,k}\}$-derivatives

[2]Note that even this small change of notation can be avoided by introducing k-DFAOs as coalgebras over the functor $F(X) = \Delta \times X^k$ as well.

of σ, coincide with the elements of the k-kernel of σ. This is used in the proof of the following theorem.

Proposition 51. *For streams $\sigma \in \Delta^\omega$ the following properties are equivalent:*

(i) The stream σ is k-automatic.
(ii) The canonical $\mathcal{N}_k$-observation graph of σ is finite.

Proof: The equivalence of (i) and (ii) is a consequence of Lemma 47 in combination with the observation that the set of functions $\{\pi_{i,k^p} \mid p \in \mathbb{N}, i < k^p\}$ coincides with the set of functions obtained from arbitrary iterations of functions $\pi_{0,k}$, $\ldots$, $\pi_{k-1,k}$ (that is, function compositions $\gamma_1 \cdot \ldots \cdot \gamma_n$ with $n \in \mathbb{N}$ and $\gamma_i \in \{\pi_{0,k}, \ldots, \pi_{k-1,k}\}$). ∎

Proposition 51 gives a coalgebraic perspective on automatic sequences. Moreover, it frequently allows for simpler proofs or disproofs of automaticity than existing characterizations. For example, in the following sections we will derive observation graphs for streams that are specified by zip-specifications. Then it is easier to stepwise iterate the finite set of functions $\{\pi_{0,k}, \ldots, \pi_{k-1,k}\}$ than to reason about infinitely many subsequences in the kernel $\{\pi_{i,k^p}(\sigma) \mid p \in \mathbb{N}, i < k^p\}$.

Proposition 51 was independently found by Kupke and Rutten, see Theorem 8 in their recent report [15].

We arrive at our second main result:

Theorem 52. *For streams $\sigma \in \Delta^\omega$ the following properties are equivalent:*

(i) The stream σ is k-automatic.
(ii) The stream σ can be defined by a zip-k specification.
(iii) The canonical $\mathcal{N}_k$-observation graph of σ is finite.
(iv) The canonical $\mathcal{O}_k$-observation graph of σ is finite.

Proof: We have that $(i) \Leftrightarrow (iii)$ by Theorem 51, $(iii) \Rightarrow (ii)$ by Lemma 40, and $(ii) \Rightarrow (iii)$ by Lemma 35. Moreover, it holds that $(iv) \Rightarrow (ii)$ by Lemma 39.

Finally, we show $(iii) \Rightarrow (iv)$. Assume $\mathcal{G} = \langle S, \langle o, n \rangle\rangle$ is a finite $\mathcal{N}_k$-observation graph with root r defining σ and let $[\![\cdot]\!]_\mathcal{G} : S \to \Delta^\omega$ be the unique F-homomorphism into $\langle \Delta^\omega, \mathcal{N}_k \rangle$. Let $n = \langle n_1, \ldots, n_k \rangle$. Then $o(s) = \mathsf{hd}([\![s]\!]_\mathcal{G})$ and $[\![n_i(s)]\!]_\mathcal{G} = \pi_{i-1,k}([\![s]\!]_\mathcal{G})$ for all $1 \le i \le k$ and $s \in S$. We define $\mathcal{G}' = \langle S', \langle o', n' \rangle\rangle$ where $S' = S \cup \{\underline{\mathsf{tl}}(s) \mid s \in S\}$, $o'(s) = o(s)$, $o'(\underline{\mathsf{tl}}(s)) = o(n_2(s))$ $n'_i(s) = n_{i+1}(s)$ for $1 \le i < k$, $n'_k(s) = \underline{\mathsf{tl}}(n_1(s))$, $n'_i(\underline{\mathsf{tl}}(s)) = n_{i+2}(s)$ for $1 \le i \le k-2$, $n'_{k-1}(\underline{\mathsf{tl}}(s)) = \underline{\mathsf{tl}}(n_1(s))$ and $n'_k(\underline{\mathsf{tl}}(s)) = \underline{\mathsf{tl}}(n_2(s))$ with root $r \in S'$. Let $[\![\cdot]\!]_{\mathcal{G}'} : S' \to \Delta^\omega$ be defined by $[\![s]\!]_{\mathcal{G}'} = [\![s]\!]_\mathcal{G}$ and $[\![\underline{\mathsf{tl}}(s)]\!]_{\mathcal{G}'} = \mathsf{tl}([\![s]\!]_\mathcal{G})$. It can be checked that $[\![\cdot]\!]_{\mathcal{G}'}$ is an F-homomorphism into $\langle \Delta^\omega, \mathcal{O}_k \rangle$ with $\sigma = [\![r]\!]_{\mathcal{G}'}$. Hence $\mathcal{G}'$ is an $\mathcal{O}_k$-observation graph defining σ. ∎

V. A Dynamic Logic Representation of Automatic Sequences

This section connects automatic sequences with expressivity in a *propositional dynamic logic* (PDL) derived from the cobases $\mathcal{N}_k$ and $\mathcal{O}_k$. For simplicity, we shall restrict attention to the case of $\Delta = \{0, 1\}$ and $\mathcal{N}_2 = \langle \mathsf{hd}, \mathsf{even}, \mathsf{odd} \rangle$.

The set of *sentences* φ and *programs* π of our version of PDL is given by the following BNF grammar:

$$\varphi ::= 0 \mid 1 \mid \neg\varphi \mid \varphi \wedge \varphi \mid [\pi]\varphi$$
$$\pi ::= \mathsf{even} \mid \mathsf{odd} \mid \pi;\pi \mid \pi \sqcup \pi \mid \pi^*$$

We interpret PDL in an arbitrary F-coalgebra $\mathcal{G} = \langle S, \langle o, n \rangle\rangle$. Actually, we can be more liberal and interpret PDL in *models* of the form $\mathcal{G} = \langle S, 0, 1, \mathsf{even}, \mathsf{odd} \rangle$ where $0 \subseteq S$, $1 \subseteq S$, $\mathsf{even} \subseteq S^2$, and $\mathsf{odd} \subseteq S^2$. These are more general than F-coalgebras because we do not insist that $0 \cap 1 = \varnothing$, or that even and odd be interpreted as functions. Nevertheless, these extra properties do hold in the intended model $\langle \Delta^\omega, 0, 1, \mathsf{even}, \mathsf{odd} \rangle$ where 0 is the set of streams whose head is the number 0, and similarly for 1; $(\sigma, \tau) \in \mathsf{even}$ iff $\tau = (\sigma_0, \sigma_2, \sigma_4, \ldots)$, and similarly for odd.

The interpretation of each sentence φ is a subset of S; the interpretation of each program π is a relation on S, that is, a subset of $S \times S$. The definition is as usual for PDL:

$$[\![0]\!] = \{x \in S : x \in 0\} \qquad [\![\mathsf{even}]\!] = \mathsf{even}$$
$$[\![1]\!] = \{x \in S : x \in 1\} \qquad [\![\mathsf{odd}]\!] = \mathsf{odd}$$
$$[\![\varphi \wedge \psi]\!] = [\![\varphi]\!] \cap [\![\psi]\!] \qquad [\![\pi_1;\pi_2]\!] = [\![\pi_1]\!];[\![\pi_2]\!]$$
$$[\![\neg\varphi]\!] = S \setminus [\![\varphi]\!] \qquad [\![\pi_1 \sqcup \pi_2]\!] = [\![\pi_1]\!] \cup [\![\pi_2]\!]$$
$$[\![\pi^*]\!] = [\![\pi]\!]^*$$
$$[\![[\pi]\varphi]\!] = \{x : (\forall y)(\langle x, y \rangle \in [\![\pi]\!] \to y \in [\![\varphi]\!])\}$$

In words, we interpret even and odd by themselves that correspond in the given model. We interpret ; by relational composition, $\sqcup$ by union of relations, * by Kleene star (= reflexive-transitive closure) of relations, and we use the usual boolean operations and dynamic modality $[\pi]\varphi$.

We use the standard boolean abbreviations for $\varphi \to \psi$ and $\varphi \leftrightarrow \psi$, and of course we use the standard semantics. We also write $\langle\pi\rangle\varphi$ for $\neg[\pi]\neg\varphi$; again this is standard.

For example, let χ be the sentence $[(\mathsf{even} \sqcup \mathsf{odd})^*](0 \leftrightarrow \neg 1)$. Then in any model $\mathcal{G}$, a point x has $x \models \chi$ iff for all points y reachable from x in zero or more steps in the relation $\mathsf{even} \cup \mathsf{odd}$, y satisfies exactly one of 0 or 1.

Proposition 53. *If $f : M \to N$ is a morphism of models and $x \models \varphi$ in M, then $f(x) \models \varphi$ in N.*

Proposition 54. *For every finite pointed model $\langle \mathcal{G}, x \rangle$ there is a sentence φ_x of PDL so that for all (finite or infinite) F-coalgebras $\langle \mathcal{H}, y \rangle$, the following are equivalent:*

(i) $y \models \varphi_x$ in $\mathcal{H}$.
(ii) There is a bisimulation between $\mathcal{G}$ and $\mathcal{H}$ relating x to y.

We call φ_x the characterizing sentence *of x.*

For infinitary modal logic, this result was shown in [4], and the result here for PDL is a refinement of it.

For example, we construct a characterizing sentence for the Thue–Morse sequence M, see Fig. 1. Let φ and ψ be given by

$$\varphi = 0 \wedge \neg 1 \wedge \langle\mathsf{even}\rangle 0 \wedge [\mathsf{even}]0 \wedge \langle\mathsf{odd}\rangle 1 \wedge [\mathsf{odd}]1$$
$$\psi = \neg 0 \wedge 1 \wedge \langle\mathsf{even}\rangle 1 \wedge [\mathsf{even}]1 \wedge \langle\mathsf{odd}\rangle 0 \wedge [\mathsf{odd}]0$$

Then $\varphi_{\mathsf{M}} = \varphi \wedge [(\mathsf{even} \sqcup \mathsf{odd})^*](\varphi \vee \psi)$ is a characteristic sentence of the top node in Fig. 1; φ_{M} also characterizes M in the following sense: the only stream σ such that $\sigma \models \varphi_{\mathsf{M}}$ is M.

Proposition 55. *The following finite model properties hold:*

(i) If a sentence φ has a model, it has a finite model [13].

(ii) If φ has a model in which even *and* odd *are total functions, then it has a finite model with these properties [5].*

Remark 56. Our statement of the second result is a slight variation of what appears in [5].

We arrive at our third main result:

Theorem 57. *The following are equivalent for $\sigma \in \Delta^\omega$:*

(i) σ is 2-automatic.

(ii) There is a sentence φ such that for all $\tau \in \Delta^\omega$, $\tau \models \varphi$ in $\langle \Delta^\omega, \langle \mathsf{hd}, \mathsf{even}, \mathsf{odd} \rangle \rangle$ iff $\tau = \sigma$.

Proof: $(i) \Rightarrow (ii)$: Let σ be automatic, and let M be a finite F-coalgebra and $x \in M$ be such that the unique coalgebra morphism $f : M \to \Delta^\omega$ has $f(x) = \sigma$. Let φ_x be the characterizing sentence of x in M, using Proposition 54. By Proposition 53, $\sigma \models \varphi_x$ in Δ^ω. Now suppose that $\tau \models \varphi_x$ in Δ^ω. Since φ_x is a characterizing sentence, there is a bisimulation on Δ^ω relating σ to τ. By Lemma 23, $\sigma = \tau$.

$(ii) \Rightarrow (i)$: Let φ be a sentence with the property that σ is the only stream which satisfies φ. Since σ has a model, it has a finite model, by Proposition 55. Moreover, this model M may be taken to be a finite F-coalgebra with a distinguished point x. By [15, Theorem 5] let $\varphi : M \to \Delta^\omega$ be the unique coalgebra morphism. Let $\tau = \varphi(x)$. Since M is finite, τ is automatic. By Proposition 54, $\tau \models \varphi$ in Δ^ω. But by the uniqueness assertion in part (2) of our theorem, we must have $\tau = \sigma$. Therefore σ is automatic. ■

VI. Mix-Automaticity

The zip-specifications considered so far were uniform, all zip-operations in a zip-k specification have the same arity k. Now we admit different arities of zip in one zip-specification (Definition 3). To emphasize the difference with zip-k specifications we will here speak of *zip-mix* specifications. This extension leads to a proper extension of automatic sequences and some delicate decidability problems.

Definition 58. A *state-dependent-alphabet DFAO* is a tuple $\langle Q, \Sigma, \delta, q_0, \Delta, \lambda \rangle$, where

– Q is a finite set of states with $q_0 \in Q$ the initial state,
– $\Sigma = \{\Sigma_q\}_{q \in Q}$ a family of input alphabets,
– $\delta = \{\delta_q : \Sigma_q \to Q\}_{q \in Q}$ a family of transition functions,
– Δ an output alphabet, and
– $\lambda : Q \to \Delta$ an output function.

We write $\delta(q, i)$ for $\delta_q(i)$ iff $i \in \Sigma_q$, and extend δ to words as follows: Let $q \in Q$ and $w = a_{n-1} \dots a_0$ where $a_i \in \Sigma_{r_i}$ $(0 \leq i < n)$ with $r_i \in Q$ defined by: $r_0 = q$ and $r_{i+1} = \delta(r_i, a_i)$. Then we let $\delta(q, w) = r_n$.

A state-dependent-alphabet DFAO can be seen as a DFAO whose transition function is a *partial* map $\delta : Q \times \bigcup \Sigma \rightharpoonup Q$ such that $\delta(q, a)$ is defined iff $a \in \Sigma_q$.

We use this concept to generalize k-DFAOs where the input format are numbers in base k by the following two-tiered construction. We define P-DFAOs where P is a DFAO determining the base of each digit depending on the digits read before. Thus P can be seen as fixing a variadic numeration system. For example, for ordinary base k numbers, we define P to consist of a single state q with output k and edges $0, \dots, k-1$ looping to itself.

Definition 59. A *base determiner* P is a state-dependent-alphabet DFAO of the form $P = \langle Q, \{\mathbb{N}_{<\beta(q)}\}_q, \delta, q_0, \mathbb{N}, \beta \rangle$. The *base-$P$ representation of* $n \in \mathbb{N}$ is defined by

$$(n)_P = (n)_{P,q_0} \qquad \text{where} \qquad (n)_{P,q} = (n')_{P,\delta(q,d)} \cdot d$$

with $n' = \lfloor \frac{n}{\beta(q)} \rfloor$ and $d = [n]_{\beta(q)}$, the quotient and the remainder of division of n by $\beta(q)$, respectively.

A *P-DFAO* A is a state-dependent-alphabet DFAO

$$A = \langle Q', \{\mathbb{N}_{<\beta'(q')}\}_{q' \in Q'}, \delta', q_0', \Delta, \lambda \rangle$$

compatible with P, i.e. $\langle Q', \{\mathbb{N}_{<\beta'(q')}\}_{q' \in Q'}, \delta', q_0', \mathbb{N}, \beta' \rangle$ and P are bisimilar.

A *mix-DFAO* is a P-DFAO for some base determiner P.

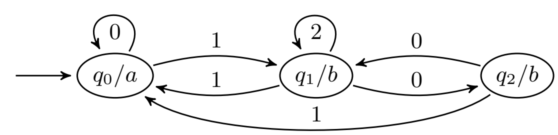
Note that the output alphabet of a base determiner can be taken to be finite as the range of β. The compatibility of A with P entails that A reads the number format defined by P. Moreover, every mix-DFAO $A = \langle Q, \{\mathbb{N}_{<\beta(q)}\}_{q \in Q}, \delta, q_0, \Delta, \lambda \rangle$ is a P_A-DFAO where $P_A = \langle Q, \{\mathbb{N}_{<\beta(q)}\}_{q \in Q}, \delta, q_0, \mathbb{N}, \beta \rangle$.

These DFAOs introduce a new class of sequences, which we call 'mix-automatic' in order to emphasize the connection with zip-mix specifications.

Definition 60. Let P be a base determiner, and $A = \langle Q, \{\mathbb{N}_{<\beta(q)}\}_{q \in Q}, \delta, q_0, \Delta, \lambda \rangle$ a P-DFAO. For states $q \in Q$, we define $\zeta(A, q) \in \Delta^\omega$ by: $\zeta(A, q)(n) = \lambda(\delta(q, (n)_P))$ for all $n \in \mathbb{N}$. We define $\zeta(A) = \zeta(A, q_0)$, and say A *generates* the stream $\zeta(A)$.

A sequence $\sigma \in \Delta^\omega$ is *P-automatic* if there is a P-DFAO A such that $\sigma = \zeta(A)$. A stream is called *mix-automatic* if it is P-automatic for some base determiner P.

Example 61. Consider the following mix-DFAO A:



We note that A is a P-DFAO where P is the base determiner obtained from A by redefining the output for q_0, q_1 and q_2 as the number of their outgoing edges 2, 3 and 2, respectively.

As an example, we compute $(5)_A$, and $(23)_A$ as follows:

$$(5)_{q_0} = (2)_{q_1} 1 = (0)_{q_2} 21 = 21$$
$$(23)_{q_0} = (11)_{q_1} 1 = (3)_{q_1} 21 = (1)_{q_2} 021 = (0)_{q_0} 1021 = 1021$$

where $(n)_q$ denotes $(n)_{A,q}$. The sequence $\zeta(A)$ begins with $a{:}b{:}b{:}a{:}b{:}\underline{b}{:}a{:}a{:}b{:}b{:}b{:}a{:}a{:}a{:}a{:}b{:}b{:}b{:}b{:}b{:}a{:}a{:}\underline{a}{:}a{:}b{:}a{:}b{:}\dots$ with entries 5 and 23 underlined. E.g. $\lambda(\delta(q_0, 1021)) = a$ since starting from q_0 and reading 1021 from right to left brings you back at state q_0 with output a.

Mix-automaticity properly extends automaticity: Let σ and τ be 2- and 3-automatic, but not ultimately periodic sequences. If $\mathsf{zip}(\sigma,\tau)$ were m-automatic, then so would be σ and τ, but then, by Cobham's Theorem [6], $2^a = 3^b$ for some $a, b > 0$. Hence $\mathsf{zip}(\sigma,\tau)$ is mix-automatic, but not automatic.

Proposition 62. *The class of mix-automatic sequences properly extends that of automatic sequences.*

Definition 63. Let $\kappa : \Delta^\omega \to \mathbb{N}_{>1}$, and let G be the functor $G(X) = \sum_{k=2}^{\infty} \Delta \times X^k$. We define the cobasis

$$\mathcal{N}_\kappa = \langle \mathsf{hd}, \lambda\sigma.\langle \pi_{0,\kappa(\sigma)}(\sigma), \ldots, \pi_{\kappa(\sigma)-1,\kappa(\sigma)}(\sigma)\rangle\rangle$$

An $\mathcal{N}_\kappa$*-observation graph* is a G-coalgebra $\mathcal{G} = \langle S, \langle o, n\rangle\rangle$ with a distinguished *root* element $r \in S$, such that there exists a G-homomorphism $[\![\cdot]\!] : S \to \Delta^\omega$ from $\mathcal{G}$ to the G-coalgebra $\langle \Delta^\omega, \mathcal{N}_\kappa\rangle$ of all streams with respect to $\mathcal{N}_\kappa$:

$$\begin{array}{ccc} S & \xrightarrow{\quad [\![\cdot]\!] \quad} & \Delta^\omega \\ {\scriptstyle\langle o,n\rangle}\downarrow & & \downarrow{\scriptstyle\mathcal{N}_\kappa} \\ \sum_{k=2}^{\infty} \Delta \times S^k & \xrightarrow{\sum_{k=2}^{\infty} \mathsf{id} \times [\![\cdot]\!]^k} & \sum_{k=2}^{\infty} \Delta \times (\Delta^\omega)^k \end{array}$$

The observation graph $\mathcal{G}$ *defines* the stream $[\![r]\!] \in \Delta^\omega$. A *mix-observation graph* is an $\mathcal{N}_\kappa$-observation graph for some κ.

The following result is a generalization of Theorem 52. The key idea is to adapt Definition 32 by computing the derivatives $\pi_{0,k}(t)\!\downarrow, \ldots, \pi_{k-1,k}(t)\!\downarrow$ of a zip-term t where now k is the arity of the first zip-symbol in the tree unfolding of t. Moreover, we note that mix-DFAOs yield mix-observation graphs by collapsing states that generate the same stream (for each of the equivalence classes one representative and its outgoing edges is chosen). This collapse caters for mix-DFAOs which employ different bases for states that generate the same stream.

Theorem 64. *For streams $\sigma \in \Delta^\omega$ the following properties are equivalent:*
(i) The stream σ is mix-automatic.
(ii) The stream σ can be defined by a zip-mix specification.
(iii) There exists a finite mix-observation graph defining σ.

Example 65. The zip-mix specification corresponding to the mix-automaton from Example 61 is:

$$\begin{array}{ll} \mathsf{X}_0 = a : \mathsf{X}_0' & \mathsf{X}_0' = \mathsf{zip}_2(\mathsf{X}_1, \mathsf{X}_0') \\ \mathsf{X}_1 = b : \mathsf{X}_1' & \mathsf{X}_1' = \mathsf{zip}_3(\mathsf{X}_0, \mathsf{X}_1, \mathsf{X}_2') \\ \mathsf{X}_2 = b : \mathsf{X}_2' & \mathsf{X}_2' = \mathsf{zip}_2(\mathsf{X}_0, \mathsf{X}_1') \end{array}$$

We have seen that equivalence for zip-k specifications is decidable (Theorem 36), and it can be shown that comparing zip-k with zip-mix is decidable as well. In the next section we show that equivalence becomes undecidable when zip-mix specifications are extended with projections $\pi_{i,k}$. But what about zip-mix specifications?

Question 66. Is equivalence decidable for zip-mix specifications?

VII. Stream Equality is Π^0_1-complete

In this section, we show that the decidability results for the equality of zip-k specifications are on the verge of undecidability. To this end we consider an extension of the format of zip-specifications with the projections $\pi_{i,k}$.

Definition 67. The set $\mathcal{Z}^\pi(\Delta, \mathcal{X})$ of *zip*$^\pi$*-terms* over $\langle \Delta, \mathcal{X}\rangle$ is defined by the grammar:

$$Z ::= \mathsf{X} \mid a : Z \mid \mathsf{zip}_k(\overbrace{Z, \ldots, Z}^{k \text{ times}}) \mid \pi_{i,k}(Z)$$

where $\mathsf{X} \in \mathcal{X}$, $a \in \Delta$, $i, k \in \mathbb{N}$. A *zip*$^\pi$*-specification* consists for every $\mathsf{X} \in \mathcal{X}$ of an equation $\mathsf{X} = t$ where $t \in \mathcal{Z}^\pi(\Delta, \mathcal{X})$.

The class of zip$^\pi$-specifications forms a subclass of pure specifications [10], and hence their productivity is decidable. In contrast, the equivalence of zip$^\pi$-specifications turns out to be undecidable (even for productive specifications).

Theorem 68. *The problem of deciding the equality of streams defined by productive zip*$^\pi$*-specifications is Π^0_1-complete.*

For the proof of the theorem, we devise a reduction from the halting problem of Fractran programs (on the input 2) to an equivalence problem of zip$^\pi$-specifications. Fractran [7] is a Turing-complete programming language. As intermediate step of the reduction we employ an extension of Fractran programs with output (and immediate termination):

Definition 69. A *Fractran program with output* consists of:
– a list of fractions $\frac{p_1}{q_1}, \ldots, \frac{p_k}{q_k}$ $(k, p_1, q_1, \ldots, p_k, q_k \in \mathbb{N}_{>0})$,
– a partial *step output* function $\lambda : \{1, \ldots, k\} \rightharpoonup \Gamma$

where Γ is a finite output alphabet. A *Fractran program* is a Fractran program with output for which $\lambda(1)\!\uparrow, \ldots, \lambda(k)\!\uparrow$.

Let F be a Fractran program with output as above. Then we define the partial function $\langle\cdot\rangle : \mathbb{N} \rightharpoonup \{1, \ldots, k\}$ that for every $n \in \mathbb{N}$ selects the index $\langle n\rangle$ of the first applicable fraction by:

$$\langle n\rangle = \min\left\{\, i \mid 1 \le i \le k,\ n \cdot \frac{p_i}{q_i} \in \mathbb{N} \,\right\}$$

where we fix $(\min \varnothing)\!\uparrow$. We define $f_F : \mathbb{N} \to \mathbb{N} \cup \Gamma \cup \{\bot\}$ by:

$$f_F(n) = \begin{cases} n \cdot \frac{p_{\langle n\rangle}}{q_{\langle n\rangle}} & \text{if } \langle n\rangle\!\downarrow \text{ and } \lambda(\langle n\rangle)\!\uparrow \\ \lambda(\langle n\rangle) & \text{if } \langle n\rangle\!\downarrow \text{ and } \lambda(\langle n\rangle)\!\downarrow \\ \bot & \text{if } \langle n\rangle\!\uparrow \end{cases}$$

for all $n \in \mathbb{N}$. The first case is a *computation step*, the latter two are *termination with* and *without output*, respectively.

We define the *output function* $\lambda^*_F : \mathbb{N} \rightharpoonup \Gamma \cup \{\bot\}$ *of* F by

$$\lambda^*_F(n) = \begin{cases} \gamma & \text{if } \gamma = f_F^{\,i}(n) \in \Gamma \cup \{\bot\} \text{ for some } i \in \mathbb{N} \\ \uparrow & \text{if no such } i \text{ exists} \end{cases}$$

If $\lambda^*_F(n)\!\downarrow$ then F is said to *halt on n with output $\lambda^*_F(n)$*. Then F is called *universally halting* if F halts on every $n \in \mathbb{N}_{>0}$, and F is *decreasing* if $p_i < q_i$ for every $1 \le i \le k$ with $\lambda(i)\!\uparrow$.

For convenience, we denote Fractran programs with output by lists of annotated fractions where $\lambda(i)\!\uparrow$ is represented by the empty word (no annotation): $\frac{p_1}{q_1}\lambda(1), \ldots, \frac{p_k}{q_k}\lambda(k)$.

In [7] it is shown that Fractran programs can simulate register machines. The next lemma is an easy consequence.

Lemma 70. *The problem of deciding on the input of a Fractran program whether it halts on 2 is Σ^0_1-complete.*

We transform Fractran programs F into two decreasing (and therefore universally halting) Fractran programs F_0 and F_1 with output such that F halts on input 2 if and only if there exists $n \in \mathbb{N}$ such that the outputs of F_0 and F_1 differ on n.

Definition 71. Let $F = \frac{p_1}{q_1}, \ldots, \frac{p_k}{q_k}$ be a Fractran program. Let $a_1 < \ldots < a_m$ be the primes occurring in the factorizations of $p_1, \ldots, p_k, q_1, \ldots, q_k$. Let z_1, z_2, c be primes such that $z_1, z_2, c > \prod_{0 \le i \le k} p_i \cdot q_i$, and $z_1 > z_2$ and $z_1 > 2 \cdot c$.

We define the Fractran program F^0 with output as:

$$\overbrace{\frac{p_1}{q_1 \cdot z_2}, \ \ldots, \ \frac{p_k}{q_k \cdot z_2}}^{\text{simulate } F}, \ \overbrace{\frac{1}{a_1}, \ \ldots, \ \frac{1}{a_m}}^{\text{cleanup}},$$
$$\underbrace{\frac{1}{c \cdot z_2}\chi_a}_{F \text{ halted}}, \ \frac{1}{c}, \ \underbrace{\frac{z_2}{z_1 \cdot z_1}, \ \frac{2 \cdot c}{z_1}}_{\text{initialization}}, \ \underbrace{\frac{1}{1}\chi_b}_{F \text{ did not halt}}$$

Let F^1 be obtained from F^0 by dropping $\frac{z_2}{z_1 \cdot z_1}$ and $\frac{2 \cdot c}{z_1}$.

Lemma 72. *The programs F^0, F^1 are decreasing and universally halting, and $\lambda^*_{F^i}(n) \in \{\chi_a, \chi_b\}$ for all $n \in \mathbb{N}$, $i \in \{0, 1\}$.*

Lemma 73. *The following statements are equivalent:*

(i) $\lambda^*_{F^0}(n) = \lambda^*_{F^1}(n)$ *for all* $n \in \mathbb{N}_{>0}$.
(ii) $\lambda^*_{F^0}(z_1^{e_1} \cdot z_2^{e_2}) = \lambda^*_{F^1}(z_1^{e_1} \cdot z_2^{e_2})$ *for all* $e_1, e_2 \in \mathbb{N}$.
(iii) The Fractran program F does not halt on 2.

Next, we translate Fractran programs to zip$^\pi$-specifications.

Definition 74. Let $F = \frac{p_1}{q_1}\lambda(1), \ldots, \frac{p_k}{q_k}\lambda(k)$ be a decreasing Fractran program with output.

Let $d := lcm(q_1, \ldots, q_k)$, and define $p'_n = d \cdot p_{\langle n \rangle}/q_{\langle n \rangle}$ and $b_n = n \cdot p_{\langle n \rangle}/q_{\langle n \rangle}$ for $1 \le n \le d$; if $\langle n \rangle\uparrow$, let $p'_n\uparrow$ and $b_n\uparrow$. We define the *zip$^\pi$-specification* $\mathcal{S}(F)$ for $1 \le n \le d$ by:

$$\begin{aligned}
&\mathsf{X}_0 = \mathsf{zip}_d(\mathsf{X}_1, \ldots, \mathsf{X}_d) && \\
&\mathsf{X}_n = \pi_{b_n - 1, p'_n}(\mathsf{X}_0) && \text{if } \langle n \rangle\downarrow \text{ and } \lambda(\langle n \rangle)\uparrow \\
&\mathsf{X}_n = \lambda(\langle n \rangle) : \mathsf{X}_n && \text{if } \langle n \rangle\downarrow \text{ and } \lambda(\langle n \rangle)\downarrow \\
&\mathsf{X}_n = \bot : \mathsf{X}_n && \text{if } \langle n \rangle\uparrow
\end{aligned}$$

Lemma 75. *Let F be a decreasing Fractran program with output. The zip$^\pi$-specification $\mathcal{S}(F)$ is productive and it holds that $[\![\mathsf{X}_0]\!]^{\mathcal{S}(F)}(n) = \lambda^*_F(n+1)$ for every $n \in \mathbb{N}$.*

Proof of Theorem 68: We reduce the *complement* of the halting problem of Fractran programs on input 2 (which is Π^0_1-complete by Lemma 70) to equivalence of zip$^\pi$-specifications.

Let F be a Fractran program. Define F^0, F^1 as in Definition 71. By Lemma 72 both are decreasing. By Lemma 75 $\mathcal{S}(F^i)$ is productive, and $[\![\mathsf{X}_0]\!]^{\mathcal{S}(F^i)}(n) = \lambda^*_{F^i}(n+1)$ for every $n \in \mathbb{N}$ and $i \in \{0, 1\}$. Finally, by Lemma 73 it follows that $\mathcal{S}(F^0)$ and $\mathcal{S}(F^1)$ are equivalent iff F does *not* halt on 2.

The equivalence problem of productive specifications is obviously in Π^0_1 since every element can be evaluated. ∎

The complexity of deciding the equality of streams defined by systems of equations has been considered in [18] and [3]. In [18], Roşu shows Π^0_2-completeness of the problem for (unrestricted) stream equations. In [3], Balestrieri strengthens the result to polymorphic stream equations. However, both results depend on the use of ill-defined (non-productive) specifications that do not uniquely define a stream. The Π^0_2-hardness proofs employ stream specifications for which productivity coincides with unique solvability. As a consequence, both results depend crucially on the notion of equivalence for specifications without unique solutions.

In contrast to [18] and [3], we are concerned with productive specifications, that is, every element of which can be evaluated constructively. Then equality is obviously in Π^0_1. We show that equality is Π^0_1-hard even for a restricted class of polymorphic, productive stream specifications.

References

[1] J.-P. Allouche and J. Shallit. The Ubiquitous Prouhet–Thue–Morse Sequence. In *SETA '98*, pages 1–16. Springer, 1999.

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

[3] F. Balestrieri. The Undecidability of Pure Stream Equations. See http://www.cs.nott.ac.uk/~fyb/.

[4] J. Barwise and L.S. Moss. *Vicious Circles: On the Mathematics of Non-Wellfounded Phenomena*. Number 60 in CSLI Lecture Notes. CSLI Publications, Stanford, CA, 1996.

[5] M. Ben-Ari, J.Y. Halpern, and A. Pnueli. Deterministic Propositional Dynamic Logic: Finite Models, Complexity, and Completeness. *J. Comput. Syst. Sci.*, 25(3):402–417, 1982.

[6] A. Cobham. On the Base-Dependence of Sets of Numbers Recognizable by Finite Automata. *Mathematical Systems Theory*, 3(2):186–192, 1969.

[7] J.H. Conway. Fractran: A Simple Universal Programming Language for Arithmetic. In *Open Problems in Communication and Computation*, pages 4–26. Springer, 1987.

[8] J. Endrullis, C. Grabmayer, and D. Hendriks. Data-Oblivious Stream Productivity. In *LPAR 2008*, volume 5330 of *LNCS*, pages 79–96. Springer, 2008.

[9] J. Endrullis, C. Grabmayer, and D. Hendriks. Complexity of Fractran and Productivity. In *CADE-22*, volume 5663 of *LNAI*, pages 371–387. Springer, 2009.

[10] J. Endrullis, C. Grabmayer, D. Hendriks, A. Isihara, and J.W. Klop. Productivity of Stream Definitions. *Theor. Comput. Sci.*, 411, 2010.

[11] C. Grabmayer, J. Endrullis, D. Hendriks, J. W. Klop, and L. S. Moss. Automatic Sequences and Zip-Specifications. Technical Report arXiv:1201.3251v1, arXiv, 2012.

[12] J.E. Hopcroft and R.E. Karp. A Linear Algorithm for Testing Equivalence of Finite Automata. Technical report, Cornell University, 1971.

[13] D. Kozen and R. Parikh. An Elementary Proof of the Completness of PDL. *Theor. Comput. Sci.*, 14:113–118, 1981.

[14] C. Kupke and J.J.M.M. Rutten. Complete Sets of Cooperations. *Information and Computation*, 208(12):1398–1420, 2010.

[15] C. Kupke and J.J.M.M. Rutten. On the Final Coalgebra of Automatic Sequences. CWI Technical Report SEN-1112, CWI, 2011.

[16] C. Kupke, J.J.M.M. Rutten, and M. Niqui. Stream Differential Equations: Concrete Formats for Coinductive Definitions. Technical Report RR-11-10, Oxford University, 2011.

[17] G. Roşu. *Hidden Logic*. PhD thesis, University of California, 2000.

[18] G. Roşu. Equality of Streams is a Π^0_2-Complete Problem. In *ICFP 2006*, pages 184–191. ACM, 2006.

[19] D. Sangiorgi and J.J.M.M. Rutten. *Advanced Topics in Bisimulation and Coinduction*, volume 52 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2012.

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

[21] J.G. Simonsen. The Π^0_2-Completeness of Most of the Properties of Rewriting Systems You Care About (and Productivity). In *RTA 2009*, volume 5595 of *LNCS*, pages 335–349. Springer, 2009.

[22] Terese. *Term Rewriting Systems*, volume 55 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2003.