Automatic Sequences and Decidable Properties:

Implementation and Applications

by

Daniel Goč

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2013

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

In 1912 Axel Thue sparked the study of combinatorics on words when he showed that the Thue-Morse sequence contains no overlaps, that is, factors of the form $ayaya$. Since then many interesting properties of sequences began to be discovered and studied. In this thesis, we consider a class of infinite sequences generated by automata, called the k-automatic sequences. In particular, we present a logical theory in which many properties of k-automatic sequences can be expressed as predicates and we show that such predicates are decidable.

Our main contribution is the implementation of a theorem prover capable of practically characterizing many commonly sought-after properties of k-automatic sequences. We showcase a panoply of results achieved using our method. We give new explicit descriptions of the recurrence and appearance functions of a list of well-known k-automatic sequences. We define a related function, called the condensation function, and give explicit descriptions for it as well. We re-affirm known results on the critical exponent of some sequences and determine it for others where it was previously unknown. On the more theoretical side, we show that the subword complexity $\rho(n)$ of k-automatic sequences is k-synchronized, i.e., the language of pairs $(n, \rho(n))$ (expressed in base k) is accepted by an automaton. Furthermore, we prove that the Lyndon factorization of k-automatic sequences is also k-automatic and explicitly compute the factorization for several sequences. Finally, we show that while the number of unbordered factors of length n is not k-synchronized, it is k-regular.

Acknowledgements

I would like to thank Jeffrey Shallit for supervising the research presented in this thesis. I am also very grateful to Luke Schaeffer, Hamoon Mousavi and Kalle Saari with whom I have collaborated on this research. Last but not least, I would like to thank Kevin Hare and Timothy Chan for serving as readers for this thesis.

Dedication

This thesis is dedicated to the one I love.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Remark 1. Parts of this chapter are taken verbatim and re-purposed from [55].

This thesis is concerned with certain natural questions about *automatic sequences*: that is, sequences over a finite alphabet where the n'th term of the sequence is expressible as a finite-state function of the base-k representation of n. In particular, we consider answering these questions purely mechanically, in an *automated* fashion.

The prototypical example of a k-automatic sequence is the Thue-Morse sequence generated by the automaton in Figure 1.1.

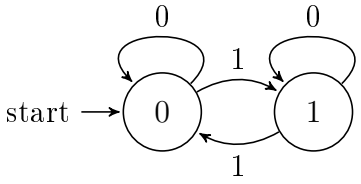


Figure 1.1: A finite automaton generating the Thue-Morse sequence

Here, the n'th term in the Thue-Morse sequence is determined by 'feeding in' n in base 2 into the automaton above and noting the label of the last state reached. For example, the fifth term is 0 since upon reading `101` we end in the state labelled 0.

1.1 History and Background

For at least 25 years, researchers have been interested in the algorithmic decidability of assertions about automatic sequences. For example, in one of the earliest results, Honkala [63] showed that, given an automaton M, it is decidable if the sequence generated by M is ultimately periodic.

Recently, Allouche et al. [9] found a different proof of Honkala's result using a more general technique. Using this technique, they were able to give algorithmic solutions to many classical problems from combinatorics on words such as

Given an automaton, is the generated sequence squarefree? Or overlapfree?

The technique of Allouche et al. is at its core, very similar to work of Büchi, Bruyère, Michaux, Villemaire, and others, involving formal logic; see, e.g., [19]. The basic idea is as follows: given the automaton M, and some predicate $P(n)$ we want to check, we alter M by a series of transformations to obtain a new automaton M' that accepts the base-k representations of those integers n for which $P(n)$ is true. Then we can check the assertion "$\exists n\ P(n)$" simply by checking if M' accepts anything (which can be done by a standard depth-first search on the underlying directed graph of the automaton). We can check the assertion "$\forall n\ P(n)$" by checking if M' accepts everything. And we can check assertions like "$P(n)$ holds for infinitely many n" by checking if M' has a reachable cycle from which a final state is reachable.

Using this idea, Allouche et al. were able to reprove, purely mechanically using a computer program, Thue's classic result on the overlapfreeness of the Thue-Morse sequence.

More recently, the technique has been applied to give decision procedures for other properties of automatic sequences. For example, Charlier et al. [28] showed that it can be used to decide if a given k-automatic sequence

- contains powers of arbitrarily large exponent;
- is recurrent;
- is uniformly recurrent.

More recently, variations of the technique have been used to

- compute the critical exponent;

- compute the initial critical exponent;
- decide if a sequence is linearly recurrent;
- compute the Diophantine exponent.

1.2 Overview

One of the main contributions of the author of this thesis was the implementation and optimization of a theorem solver able to decide precisely the sort of predicates described above. Unlike previous applications which tended to be more *ad hoc*, this theorem solver rather versatile in that it takes the predicate in question as program input. The author is also responsible for analysing and cataloguing the results presented in this thesis. Furthermore, the author contributed many important non-trivial insights necessary to formulate the predicates describing the desired properties in question.

Chapters 2 and 3 of this thesis lay the groundwork for many of the concepts and ideas used throughout the thesis. Chapter 2 gives preliminary definitions as well as a list of various k-automatic sequences frequently referred to in the later chapters. Chapter 3 presents the main result of this thesis — mainly the implementation details of the theorem solver.

In Chapters 4, 5, and 6 present respectively a catalogue of the recurrence, appearance and condensation functions of a list of well-known k-automatic sequences. Chapter 7 stays the course by giving a list of the critical exponents for various sequences. The critical exponent of a sequence is the largest power that occurs as a factor in the sequence. Some of the results in Chapter 7 were previously known and are here re-confirmed, while others are novel.

Chapter 8 restates a result regarding the least periods of k-automatic sequences previously published in [55] and also presents novel results regarding quasi-periods. In Chapter 9 we consider bordered and unbordered factors of k-automatic sequences and we give a description of the number of unbordered factors of length n as a k-regular sequence. Chapter 10 concerns itself with Lyndon factors and factorizations of k-automatic sequences.

A function f is called k-synchronized if the pairs $(n, f(n))$ are accepted by an automaton. For example, in Chapters 4, 5, and 6 we show that the recurrence, appearance and condensation functions of k-automatic sequences are k-synchronized. Chapter 11 demonstrates that the subword complexity of k-automatic sequences is also k-synchronized. The

subword complexity of a sequence is a function $\rho(n)$ that counts the number of unique factors of length n that occur in the sequence. Furthermore, as we shall see in this chapter, the number of factors that are primitive (or powers) is k-synchronized as well. In Chapter 11 we also demonstrate that the number of unbordered factors, previously discussed in Chapter 9, is not k-synchronized in general.

Chapters 9, 10, and 11, present material previously published in [56], [57], and [58], respectively; only the contributions of the author of this thesis are included here.

As is customary, Chapter 12 summarizes the results and presents a list of open problems as well as potential further research.

Chapter 2

Preliminaries

2.1 Words

An *alphabet* is a (typically) finite set of symbols. For example, the English lower-case alphabet $\Delta = \{a, b, c, \ldots, z\}$ consists of 26 symbols. The Greek alphabet $\Gamma = \{\alpha, \beta, \ldots, \omega\}$ only consists of 24 symbols. For our purposes, alphabets need not be based on real-world examples. It is often useful to consider the binary alphabet $\Sigma_2 = \{0, 1\}$. In the rest of this paper we will refer to a generalized version of the binary alphabet, the k-ary alphabet $\Sigma_k = \{0, 1, \ldots, k-1\}$. We let $(n)_k$ denote the k-ary representation of n starting with the most significant digit and omitting any leading zeroes.

A *word* over Σ is a member of Σ^* where $\Sigma^* = \bigcup_{i \geq 0} \Sigma^i$. For example, the English word `hi` is a member of Δ^2, and the word `four` is a member of Δ^4. In this way, all the lower-case English words belong to Δ^*. We extend the notion of a word to *right-infinite words*. A *right-infinite word* $\mathbf{x}$ over Σ is defined as a sequence $\mathbf{x} = (x_i)_{i \geq 0}$ where each $x_i \in \Sigma$. One can, of course, also define *left-infinite words* or *bi-infinite words* in a similar matter, but they are not necessary for this thesis. In the rest of this thesis, an *infinite word* will be used to mean a right-infinite word, and we will use a **bold typeface** to differentiate them from finite words.

A *factor* of a finite word w is a contiguous subsequence of the word. For example, `duct` is a factor of the word `introduction`. To refer to the character at position i of w we write $w[i]$. Note that the first character is at position 0, rather than 1. We let $w[i..j]$ denote the factor of w starting at position i and ending at j. In this example, if we let $v =$ `duct`, and $w =$ `introduction`, we have that $v = w[5..8]$. A factor of w that starts at the first

position is called a *prefix* of w. To illustrate, the word `intro` is a prefix of `introduction`. We say that v is a *proper prefix* of w if $v \neq w$. Similarly, a *suffix* of a word w is a factor that ends on the last position of w. The word `ion` is a *proper suffix* of `introduction`.

The notion of a factor can be extended to factors of infinite words. We only need to extend the indexes to $\mathbb{N} \cup \{\infty\}$. For example, we let $\mathbf{x}[i..\infty]$ denote the suffix of an infinite word $\mathbf{x}$ starting at position i.

2.2 Languages

A *language* is simply a set of finite words. We say that L is a language *over the alphabet* Σ if each word in L is also a member of Σ^*. In fact, Σ^* is itself a language.

2.3 Morphisms and Codings

A *morphism* is a map ϕ from Σ^* to Δ^* such that $\phi(xy) = \phi(x)\phi(y)$ for all $x, y \in \Sigma^*$. We call a morphism ϕ a *k-uniform morphism* if for all $a \in \Sigma$ we have $|\phi(a)| = k$. For example, the morphism

$$0 \to 01$$
$$1 \to 10$$

is 2-uniform. We call a word $\mathbf{x}$ a *fixed point of* ϕ if $\mathbf{x} = \phi(\mathbf{x})$.

A *coding* is simply a map μ from an alphabet Σ to alphabet Δ (possibly the same).

2.4 Automata

2.4.1 Nondeterministic Finite Automata

Formally, a *nondeterministic finite automaton (NFA)* is defined as a five-tuple $(Q, \Sigma, \delta, I, F)$ where Q is a finite set of states, Σ is an alphabet,

$$\delta : Q \times \Sigma \to 2^Q$$

is the transition function, I is the set of initial states, and $F \subseteq Q$ is the set of accepting states. Essentially, an automaton performs 'work' on an input word and decides whether or not to 'accept' the word. Formally an automaton M *accepts* a word w iff there exists a path from an initial state to a final state labelled w. We let $L(M)$ denote the set of words accepted by M, which is called the *language accepted* by M. A language is called *regular* if there exists a NFA that accepts it.

It is well-known that the English word `buffalo` is an noun adjunct, noun and a verb. As such, the word `buffalo` has the interesting property that any number of repetitions of the followed by a period forms a grammatically correct English sentence (if one ignores upper-case letters.) Below in Figure 2.1 we depict an automaton that will accept precisely such sentences.

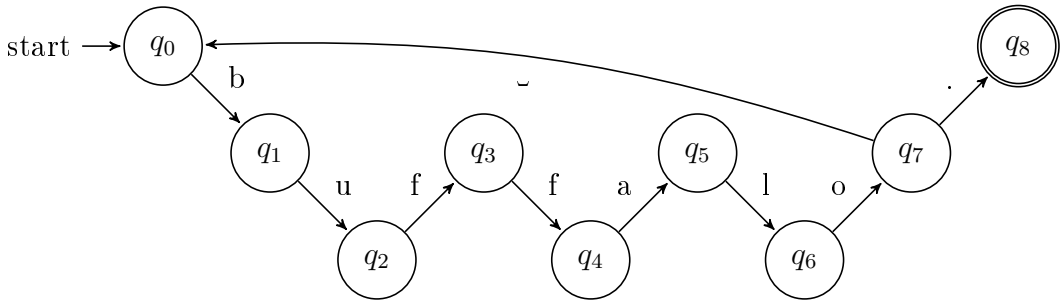


Figure 2.1: A finite automaton accepting English sentences consisting of the word `buffalo` (The symbol ␣ denotes the space character.)

2.4.2 Deterministic Finite Automata

In fact, the example in Figure 2.1 is a *deterministic finite automaton (DFA)*. A DFA is a restriction on an NFA where there is only one start state $I = \{q_0\}$ and the transition function δ only maps to one destination. In fact, any NFA can be turned into an *equivalent DFA* — a DFA that accepts the same set of words. However, the process of conversion can increase the number of states exponentially. In other words, the equivalent minimal DFA for an NFA with n states might have as many as 2^n states. We say that the number of states of the minimal DFA accepting a regular language L is the *state complexity* of L.

2.4.3 Deterministic Finite Automata with Output

Traditionally, automata are restricted to two possible outputs: `accept` and `reject`. For the purposes of our discussion in this thesis, we find it useful to extend a DFA to have a finite number of outputs. We shall refer to such a machine as a *deterministic finite automaton with output (DFAO)*. Formally a DFAO is a six-tuple $(Q, \Sigma, \delta, q_0, \Delta, \tau)$ where where Q is a finite set of states, Σ is an alphabet as before, $\delta : Q \times \Sigma \to Q$ is the transition function restricted to be deterministic, q_0 is the initial state and Δ is output alphabet, and $\tau : Q \to \Delta$ is the output function. A DFAO is particularly useful when representing disjoint classes of properties of the output, for example to output n (mod 3), as in Figure 2.2.

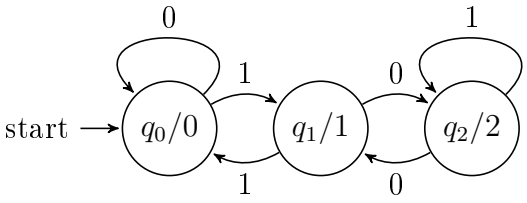


Figure 2.2: A DFAO reading in $(n)_2$ and outputting n (mod 3)

Note that the output associated with each state of a DFAO is typically printed after a forward slash (/) in the state label.

2.4.4 Transducers

Alternatively, instead of one output at the end of the computation, an automaton can produce output at each step in the computation. One such class of automata is the class of a *transducers*. A transducer is formally defined as six-tuple $(Q, \Sigma, \delta, q_0, \Delta, \lambda)$ where Q, Σ, δ, q_0, and Δ are as in a DFAO and

$$\lambda : Q \times \Sigma \to \Delta^*$$

is the output function. We think of the transducer as a function from Σ^* into Δ^*; a transducer output a word in Δ^* upon reading an input from Σ^*. For example, in Figure 2.3 the transducer reads binary encodings of natural numbers and 'divides' the input number x by 2.

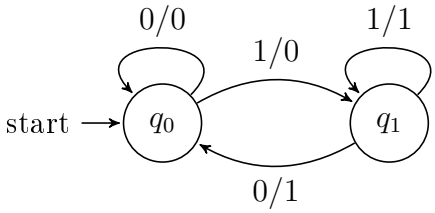


Figure 2.3: A finite-state transducer that outputs $\left(\lfloor \frac{x}{2} \rfloor\right)_2$ on input $(x)_2$

2.5 k-Automatic Sequences

Let $\mathbf{x} = (a(n))_{n \geq 0}$ be an infinite sequence over a finite alphabet Δ, and let $k \geq 2$ be an integer. Then $\mathbf{x}$ is said to be *k-automatic* if there is a deterministic finite automaton M taking as input the base-k representation of n, and having $a(n)$ as the output associated with the last state encountered [10]. In this case, we say that M *g*enerates the sequence $\mathbf{x}$.

Alternatively, an a sequence is k-automatic if and only if it is the the fixed point of a k-uniform morphism, or the image, under a coding, of such a fixed point. We will see some examples of that below.

2.5.1 The Thue-Morse Sequence

The *Thue-Morse sequence* $\mathbf{t} = t_0 t_1 t_2 \cdots = 0110100110010110100101100110100 1 \cdots$ also sometimes called *Prouhet-Thue-Morse sequence* was first studied by Eugène Prouhet in 1851, although it was Axel Thue who formally described it in 1912 [104]. In fact, the field of combinatorics on words, the proper subject of this thesis, was founded when Thue studied this famous sequence.

There are many equivalent descriptions of the Thue-Morse sequence, one such is to let $\mathbf{t}$ be the fixed point of the morphism

$$0 \to 01$$
$$1 \to 10$$

starting with with the string '0'.

Another way to define Thue-Morse is to start with 0 and repeatedly append the complement of the string thus far [7].

The Thue-Morse sequence is 2-automatic and in Figure 2.4 we give, as an example, an automaton generating the sequence. The input is n, expressed in base 2, and the output is the number contained in the state last reached. Thus $t(n)$ is the sum, modulo 2, of the binary digits of n.

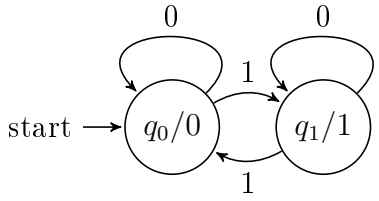


Figure 2.4: A DFAO generating $\mathbf{t}$: the Thue-Morse sequence

2.5.2 The Rudin-Shapiro Sequence

The *Rudin-Shapiro sequence* [89, 98] $\mathbf{r} = 00010010000111010001001011100 0\cdots$ is yet another famous 2-automatic sequence. It can be defined as the count, modulo 2, of the number of (possibly overlapping) occurrences of 11 in $(n)_2$.

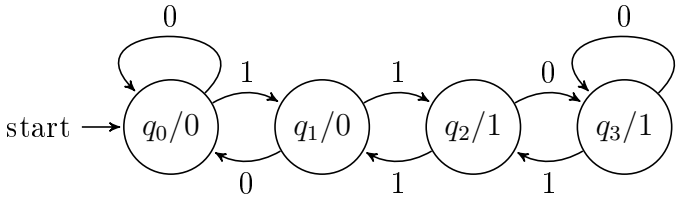


Figure 2.5: A DFAO generating $\mathbf{r}$: the Rudin-Shapiro sequence

2.5.3 The Regular Paperfolding Sequence

The *regular paperfolding sequence* [34] $\mathbf{p} = 11011001110010011101100011001001\cdots$ is a sequence generated by iteratively folding a piece of paper right and then uncurling it and noting down the sequence of turns. For example, we may assign 0 to left turns and 1 to right turns.

Thus each consecutive iteration is defined as the previous iteration followed by 1 and again the previous iteration, but in reverse and complemented, i.e., $\mathbf{p} = \lim_{n\to\infty} p(n)$ where

$p(0) = \epsilon$ and

$$p(n) = p(n-1)\ 1\ \overline{p(n-1)^R}.$$

for all $n \geq 1$ where $\overline{x}$ flips the bits in x and R denotes reversal. To illustrate, the sequence would be generated as follows:

$$\begin{aligned}
&\mathbf{1}\\
&1\ \mathbf{1}\ 0\\
&1\ 1\ 0\ \mathbf{1}\ 1\ 0\ 0\\
&1\ 1\ 0\ 1\ 1\ 0\ 0\ \mathbf{1}\ 1\ 1\ 0\ 0\ 1\ 0\ 0
\end{aligned}$$

and so on.

The paperfolding sequence can also be described by the formula

$$p_n = \begin{cases} 1, & \text{if } m \equiv 1 \pmod 4; \\ 0, & \text{if } m \equiv 3 \pmod 4, \end{cases}$$

where $n+1 = m \cdot 2^r$ and m is odd. (We note that the '$n+1$' is a modification of the usual definition which is due to our convention of indexing sequences starting at 0.)

p is also the fixed point of the map

$$\begin{aligned}
00 &\to 1000\\
01 &\to 1001\\
10 &\to 1100\\
11 &\to 1101
\end{aligned}$$

starting with with the string '11'. The paperfolding sequence is a 2-automatic sequence and is, for example, generated by the automaton in Figure 2.6.

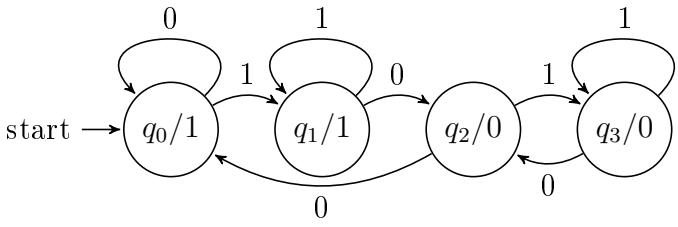


Figure 2.6: A DFAO generating **p**: the paperfolding sequence

2.5.4 The Period-doubling Sequence

The *period-doubling sequence* [33] $\mathbf{d} = 10111010101110111011101010111010\cdots$ is a sequence related to the Thue Morse sequence and defined by

$$d_n = \begin{cases} 1, & \text{if } t_n = t_{n+1}; \\ 0, & \text{otherwise,} \end{cases}$$

where $t_0 t_1 t_2 \cdots$ is the Thue-Morse word $\mathbf{t}$.

Note that $\mathbf{d}$ is the fixed point of the morphism

$$0 \to 01$$
$$1 \to 00$$

starting with with the string '0'. The period doubling sequence is a 2-automatic sequence and is, for example, generated by the automaton in Figure 2.7.

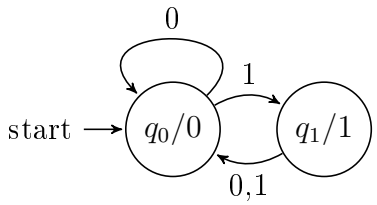


Figure 2.7: A DFAO generating $\mathbf{d}$: the period-doubling sequence

2.5.5 The Baum-Sweet Sequence

The *Baum-Sweet sequence* $\mathbf{b} = 11011001010010011001000001001001\cdots$ takes on the value 1 if the binary representation of n contains no block of consecutive 0's of odd length, and 0 otherwise [10]. Alternatively, this sequence is defined by

$$b_n = \begin{cases} 0, & \text{if } m \text{ is even;} \\ b_{(m-1)/2}, & \text{if } m \text{ is odd,} \end{cases}$$

where $n = m \cdot 4^\ell$ and m is not divisible by 4.

There is a 4-state automaton over Σ_2 generating the Baum-Sweet sequence, as can be seen in Figure 2.8 demonstrating it to be yet another 2-automatic sequence.

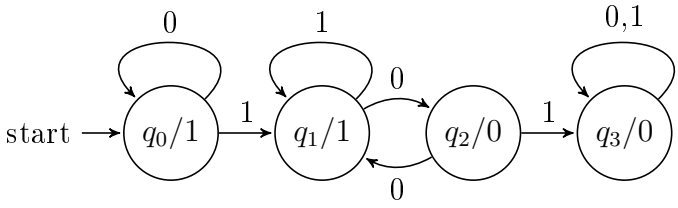


Figure 2.8: A DFAO generating **b**: the Baum-Sweet sequence

2.5.6 The Mephisto Waltz Sequence

The *Mephisto waltz sequence*, $\mathbf{m} = 00100111000100111011011000100100\cdots$ so-called for its alleged resemblance of Franz Liszt's famous Mephisto waltzes, is defined as the fixed point of the morphism

$$0 \to 001$$
$$1 \to 110$$

starting with with the string '0'.

The Mephisto waltz sequence is a 3-automatic sequence and is generated by the automaton in Figure 2.9.

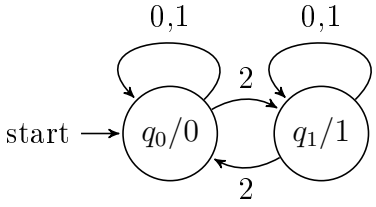


Figure 2.9: A DFAO generating **m**: the Mephisto Waltz sequence

2.5.7 The Stewart Choral Sequence

The *Stewart choral sequence*, $\mathbf{s} = 00100101100100101100101101100100\cdots$ was introduced by Ian Stewart in an article in the *Scientific American* in 1995 [102]. This sequence is

defined as the fixed point of the morphism

$$0 \to 001$$
$$1 \to 011$$

starting with with the string '0'. More recently, it was picked up and studied by J. R. Roche [86] who showed that the sequence can also be defined by

$$s_n = \begin{cases} 1, & \text{if } \exists\ i, j \in \mathbb{Z}^* \text{ such that } n = 3^{i+1}j + \frac{1}{2}(5 \cdot 3^i - 1); \\ 0, & \text{if } \exists\ i, j \in \mathbb{Z}^* \text{ such that } n = 3^{i+1}j + \frac{1}{2}(3^i - 1). \end{cases}$$

The Stewart choral sequence is also 3-automatic sequence and is generated by the automaton in Figure 2.10.

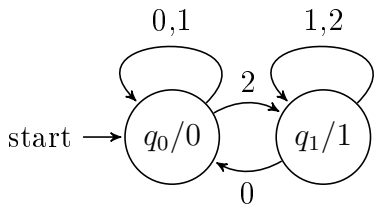


Figure 2.10: A DFAO generating **s**: the Stewart choral sequence

2.5.8 The Leech Sequence

The *Leech sequence*, also known as the *Leech word*, is a 13-automatic sequence described in 1957 by John Leech [69]. It is defined as the fixed point of the following morphism,

$$0 \to 0121021201210$$
$$1 \to 1202102012021$$
$$2 \to 2010210120102$$

starting with the string '0'.

The Leech word is generated by a 3 state automaton over Σ_{13} as can be seen in Figure 2.11.

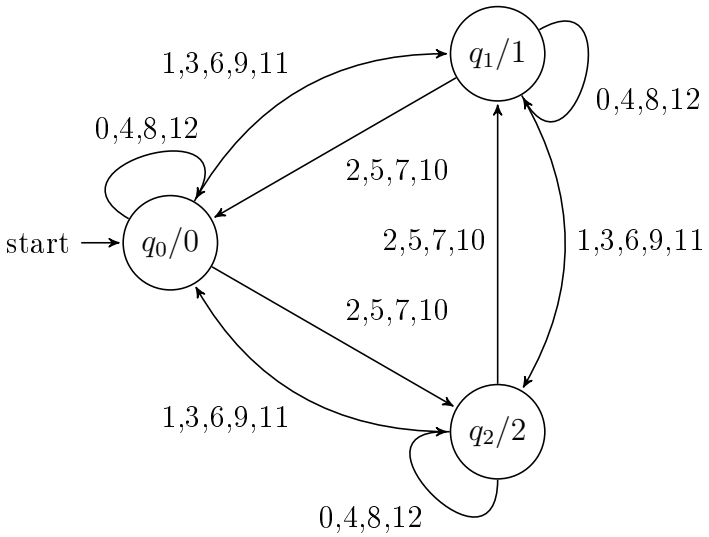


Figure 2.11: A DFAO generating **l**: the Leech sequence

2.5.9 k-Kernel

A subsequence **y** of **x** of the form $\mathbf{y}[i] = \mathbf{x}[ik^a + l]$ where $a \geq 1, l \geq 0$ is called a k-kernel of **x**. A distinguishing aspect of k-automatic sequences is that the *k-kernel* of a k-automatic sequence is finite [10].

2.6 k-Regular Sequences

The k-automatic sequences are defined over finite alphabets. A natural extension is to extend to an infinite alphabet. Formally, we say that a sequence $(a(n))_{n\geq 0}$ is *k-regular* if for every $i \geq 0$ and $0 \leq b \leq k^i$ there exist $c_1, c_2, \ldots, c_s \in \mathbb{Z}$ such that for all integers $n \geq 0$ we have

$$a(k^i n + b) = \sum_{1 \leq j \leq s} c_j a(n)$$

Here we borrow a typical example of a k-regular sequence from [10]. Let $s_2(n)$ be the function counting the sum of the binary digits of n. Whenever $i \geq 0$ and $0 < b < 2^i$ we

have that

$$s_2(2^i n + b) = s_2(n) + s_2(b).$$

It follows that s_2 is 2-regular.

See [11] for more details.

2.7 k-Synchronized Functions

Another way to extend the notion of 'automatic' to an infinite alphabet is via k-synchronized functions. We say a function $f : \mathbb{N} \to \mathbb{N}$ is *k-synchronized* if there exists an automaton accepting

$$\{(n, f(n))_k \text{ for all } n \geq 0\}.$$

This notion is taken from Carpi and more information and examples can be found in [22, 23, 25]. Below in Figure 2.12 we give an example of a 2-synchronized function $f(n) = 2^{\lceil \log_2(n) \rceil}$. The representation used here is most significant digit first and is further explained in Section 3.3.

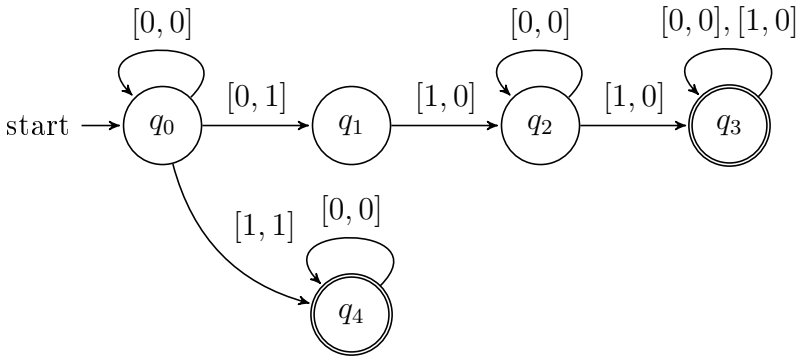


Figure 2.12: A DFA encoding the 2-synchronized function $f(n) = 2^{\lceil \log_2(n) \rceil}$

Chapter 3

Decidability and Implementation

Many interesting properties of k-automatic sequences expressible in our proposed logical theory. Charlier, Rampersad and Shallit [28] showed that this theory is decidable. The linchpin on this thesis rests on the fact that many of the properties of common sequences such as Thue-Morse or Rudin-Shapiro are also practically computable. We will see lots of examples of properties we computed in the later sections of this thesis.

3.1 Logic

Formally, the logical predicates presented in this paper can be hierarchically defined as follows:

- *Variables* are symbols from a set V and they represent natural numbers.
- *Integer expressions* (addition, multiplication by constants) are either
  - c, a constant,
  - x, or
  - $(A + B)$;

  where $c \in \mathbb{N}$, $x \in V$, and A and B are integer expressions. Once we define $(A + B)$ we can also express $c \cdot x$ for any $c \in \mathbb{N}$ and $x \in V$ as

$$c \cdot x = (\underbrace{x + (x + \cdots (x + x) \cdots)}_{c \text{ times}}).$$

We will use this short-hand in the rest of this thesis.

For example: '$((2x + 3) + 4z)$' is an integer expression.

- *Comparison predicates* (CP) are predicates comparing integer expressions and are of the form
  - $A = B$,
  - $A \leq B$, or
  - $A < B$;

  where A and B are integer expressions. For example: '$(x + y) < z$' is a comparison predicate.

- *Indexing predicates* (IP) are predicates indexing into a k-automatic sequence and are of the form
  - $\mathbf{x}[A] = \mathbf{x}[B]$,
  - $\mathbf{x}[A] \leq \mathbf{x}[B]$, or
  - $\mathbf{x}[A] < \mathbf{x}[B]$;

  where A and B are integer expressions and some sort of ordering is imposed on Δ the finite alphabet over which the sequence $\mathbf{x}$ is defined. An example of an indexing predicate would be '$\mathbf{t}[i] = \mathbf{t}[2i]$' where $\mathbf{t}$ is the Thue-Morse sequence.

- *Formulae* are essentially boolean functions on variables. A formula is either
  - P
  - $\neg\phi$,
  - $(\phi \wedge \psi)$,
  - $(\phi \vee \psi)$, or
  - $\exists x, \phi$;

  where P is either IP or CP and ϕ and ψ are formulae. We let $\neg\phi$ denote the *logical negation* of ϕ. We also let $(\phi \wedge \psi)$ denote the *conjunction* and $(\phi \vee \psi)$ the *disjunction* of the formulae ϕ and ψ.

  The *existential quantifier* $\exists x$ requires that x is a variable not already quantified in ϕ.

The identity

$$\neg\forall x, \phi = \exists x, \neg\phi$$

which is a form of De Morgan's law gives us a way of expressing the *universal quantifier* and for the rest of this paper we will use $\forall x, \phi$ as a short form for $\neg\exists x, \neg\phi$.

3.2 Decidability

Many of the logical concepts present in this thesis can be thought of as an extension of *Presburger arithmetic*. Presburger arithmetic is a first-order theory of the natural numbers with addition [82]. In 1929 a 25-year old Jewish-Polish mathematician Mojżesz Presburger showed that Presburger arithmetic is decidable. In fact, in 1974 Fischer and Rabin showed that the computational complexity of this decision problem is doubly exponential [45].

In order to effectively talk about various logical theories we first need to formally define a few things. Let S be a set of structure in the sense of [19]. The *theory of* S is the set of logical sentences true for S and we denote it $\mathrm{Th}(S)$.

Thus, using the notation of Bruyère et al. in [19], Presburger arithmetic can be also identified as $\mathrm{Th}(\langle\mathbb{N}, +, <, 0\rangle)$. This refers to the first order logical theory defined over natural numbers with the predicates $+$ and $<$ and the constant 0.

Furthermore, we say that a decision problem is *decidable* if there exists a program that will halt to return a `true` or `false` value (instead of looping forever) and furthermore, the value returned answers the decision problem correctly. We say a logical theory $\mathrm{Th}(S)$ is *decidable* if the inclusion of a logical sentence over S in $\mathrm{Th}(S)$ is decidable.

For a given k-automatic sequence $\mathbf{x}$, the logical structures described in Section 3.1 can be expressed as

$$\langle\mathbb{N}, +, <, 0, P_a, V_k\rangle$$

where $P_a(i)$ is the predicate '$\mathbf{x}[i] = a$' and $V_k(x) = y$ where y is the largest power of k that divides x.

Notably, the predicates $\mathbf{x}[i] = \mathbf{x}[j]$ and $\mathbf{x}[i] < \mathbf{x}[j]$ mentioned in Section 3.1 are missing from this list. However, it is immediately obvious that they can be expressed using a clever arrangement of $P_a(i)$.

In fact, as Bruyère et al. show in [19], the logical structure $\langle\mathbb{N}, +, <, 0, P_a, V_k\rangle$ is equivalent to $\langle\mathbb{N}, +, V_k\rangle$. Essentially, $<$ and 0 can be expressed using existential quantifiers and $+$ and V_k are powerful enough to simulate any k-automatic sequence.

Much like Presburger arithmetic, the theory $\langle \mathbb{N}, +, V_k \rangle$ is also decidable. In the rest of this chapter we give details on how the validity of a sentence in $\langle \mathbb{N}, +, V_k \rangle$ is computed. The following theorem formally describes the decidability result, which we quote essentially verbatim from [57].

Theorem 2. *Let $P(n)$ be a predicate associated with a k-automatic sequence $\mathbf{x}$, expressible using addition, subtraction, comparisons, logical operations, indexing into $\mathbf{x}$, and existential and universal quantifiers. In other words, let $P(n)$ be over $\langle \mathbb{N}, +, <, 0, P_a, V_k \rangle$.*

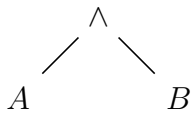
Then there is a computable finite automaton accepting the base-k representations of those n for which $P(n)$ holds. Furthermore, we can decide if $P(n)$ holds for at least one n, or for all n, or for infinitely many n.

Sketch of proof. Let ϕ be a sentence given over $\langle \mathbb{N}, +, >, 0, P_a, V_k \rangle$

First, we create a parse tree for ϕ. Starting at the leaf nodes we replace the logical predicates with equivalent automata.

We proceed to collapse the tree evaluating an expression whenever its child nodes have been evaluated.

For example, if our current step looks as follows:



we perform the direct product construction for intersection on the automata A and B. (Similarly for disjunction, negation or the existential quantifier.)

Eventually, we end up with an automaton 'encoding' ϕ as promised.

□

3.3 Representing Natural Numbers

Any natural number can be readily expressed in base k starting with either the least digit (LSD) or the most significant digit (MSD). For example, the number eleven reads as `1101` in base 2 in MSD representation. To facilitate design decisions mentioned in Section 3.4 we chose the LSD representation throughout the computation steps of our

decision procedure. We can easily convert between LSD and MSD representations via language reversal. We should note an important caveat, the worst case scenario for language reversal is an exponential increase in state complexity. Fortunately, none of the automata in our experiments achieved this pathological behaviour.

Remark 3. In order to best illustrate the underlying processes all the automata in this section assume LSD representation. However, the final automata presented in the following section of this thesis have been 'reversed' into the MSD representation.

In Figure 3.1 we can see an LSD representation automaton generating the Baum-Sweet sequence. Compare with the MSD representation found in Figure 2.8.

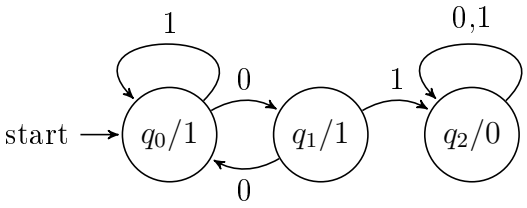


Figure 3.1: An automaton generating the Baum-Sweet sequence in LSD representation

3.3.1 Tuples

We can also represent several natural numbers 'simultaneously' by using symbols in Σ_k^n where n is the number of natural numbers being represented. For example, the pair $(13, 5)$ can be represented in base 2 (in LSD representation) by the word

$$\begin{bmatrix} 1 \\ 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix}.$$

We omit the trailing zeroes. This way an automaton with input alphabet Σ_k^n can be thought of as accepting n integers in base k. Due to the limitations of the graphing software we will represent the vectors as row vectors. Thus, we will more commonly represent the pair $(13, 5)$ as

$$[1, 1], [0, 0], [1, 1], [1, 0].$$

Below, in Figure 3.3.1 is an automaton on two base-2 variables x and y accepting only pairs (x, y) such that x is even and y is odd. The automaton is in LSD representation. We say that the automaton *encodes* the predicate

$$(n) : \exists m \text{ such that } 2m = n.$$

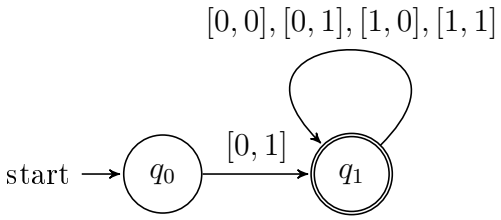


Figure 3.2: An automaton accepting (x, y) where x is even and y is odd (LSD representation)

3.4 Integer Expressions

Integer expressions involving n variables can be expressed by automata. More formally, given an integer expression A (as defined in the Section 3.1) in a predicate involving n variables $x_1, x_2, \ldots, x_n$ we can create a transducer with input $(x_1, x_2, \ldots, x_n)$ and output in Σ_k where the output is understood encode the integer expression A in base k.

For example, the automaton in Figure 3.4 encodes the expression '$(x + y)$'.

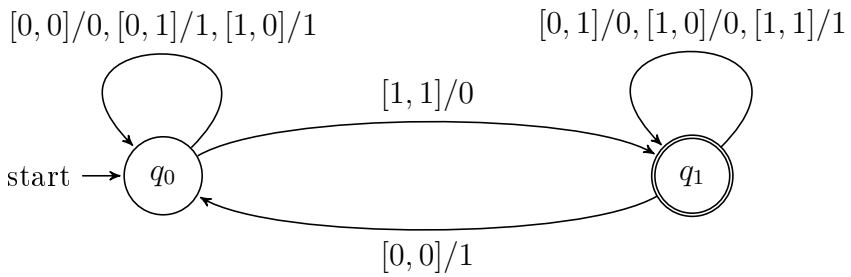


Figure 3.3: A transducer with input (x, y) encoding the expression '$(x + y)$'

The above expression is encoded by a transducer in LSD representation and in fact, there is no MSD transducer that would express it. To see this consider adding $(2^m - 1, 1)$, an MSD transducer would have to 'wait' until it sees the last digit before being able to output anything. A simple argument with the pumping lemma then refutes the existence of such a transducer.

Alternatively, we can avoid transducers altogether and choose to express integer expressions with an automaton on $n+1$ inputs where the last input encodes the result of the expression. This would allow us to work using both MSD and LSD representations.

For example, for the expression '$(x + y)$' we create the language of $(x, y, z)_k$ such that $x + y = z$.

A neat aspect of this alternative encoding is that addition can be expressed in non base-k systems such as the Ostrowski numeration. These other exotic numerations can be used to automatically prove theorems about certain non-automatic sequences such as the Sturmian words [94].

On the other hand, the transducer approach does not require the introduction of another free variable into the predicate, instead we 'pipe' the expression directly where needed. In practice, the number of free variables is a limiting factor on the feasibility of the computation; each extra variable effectively increases the number of transitions k-fold. And thus, for reasons of efficiency and convenience we have chosen to implement the first alternative. Perhaps future research exploring the alternative approach is warranted.

3.5 Expression Comparison

Comparing variables or integer expressions in base-k is likewise readily done with automata. Below in Figure 3.5, we give an example of a base-2 comparer of inputs (x, y) ensuring that $x \leq y$. Similarly, by altering the set of accepting states and automata ensuring $x < y$, $x = y$, $x \geq y$, etc. can be made.

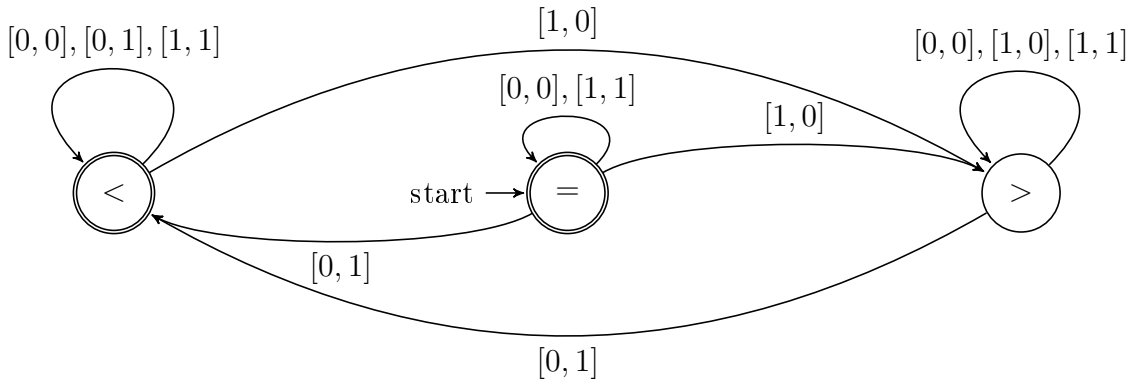


Figure 3.4: An automaton accepting input (x, y) such that '$x \leq y$'

In order to compare two integer expressions A and B, we take the cross product of their corresponding automata resulting in an automaton on inputs $(x_1, x_2, \ldots, x_n)$ and

outputting (A, B). Finally, we compose this automaton with the appropriate base-k comparison automaton.

3.6 Logical Connectives and Logical Negation

Perhaps the most straightforward to implement are the logical connectives. Suppose we have two automata M_P and M_Q for their respective predicates P and Q on n variables $(x_1, x_2, \ldots, x_n)$. The predicate $(P \wedge Q)$ corresponds to the intersection of M_P and M_Q and similarly the predicate $(P \vee Q)$ corresponds to the union of M_P and M_Q.

Similarly, given an automaton M_P for a predicate P we create the complement automaton $\overline{M_P}$ for the predicate $(\neg P)$.

3.7 The Existential Quantifier

Suppose we have an automaton M_P for a predicate P on variables $x_1, x_2, \ldots, x_n$. Let $\Gamma_n : \Sigma^n \to \Sigma^{n-1}$ be the projection dropping the last coordinate, i.e.,

$$\Gamma_n([a_1, a_2, \ldots, a_n]) = [a_1, a_2, \ldots, a_{n-1}].$$

Such a projection naturally implies a map from n-variable automata to $(n-1)$-variable automata. In particular, let $M_P = (Q, \Sigma^n, \delta, q_0, F)$ then the operation implied by Γ_n maps M_P to $M' = (Q, \Sigma^{n-1}, \delta', q_0, F)$ where

$$\delta'(q, a) = \{r \mid \exists b \in \Sigma^n \text{ such that } \Gamma_n(b) = a \text{ and } \delta(q, b) = r\}.$$

This operation corresponds to applying the existential quantifier on the variable x_n. In other words the automaton M' encodes the predicate '$\exists x_n P$'. To see this consider a valuation Θ on the variables $x_1, x_2, \ldots, x_n$ where $\Theta(x_i) = m_i$. The predicate $\exists x_n P$ is satisfied whenever, having fixed the valuation for all $j \neq i$, there exists some assignment m for x_n such that P is satisfied under the new valuation $\Theta[x_n/m]$. However, the predicate P is encoded with the automaton M_P and whenever such an assignment exists then there will exist a path from the start state to a final state labelled $(m_1, m_2, \ldots, m_{n-1}, m)_k$. Thus, in the new automaton M' there exists a path from a start state to a final state labelled $(m_1, m_2, \ldots, m_{n-1})_k$. We conclude that M' accepts precisely those assignments to $x_1, x_2, \ldots, x_{n-1}$ that satisfy $\exists x_n P$ and so M' encodes $\exists x_n P$.

We should note that the operation will generally result in a nondeterministic automaton that subsequently has to be determinized before a logical negation operation may be applied.

The author of this thesis has made the design decision to determinize and minimize the encoding automaton at each step in the predicate parse. This protocol is meant to keep the automata small and easy to work with, although it may introduce an unnecessary workload such as in the case of two consecutive existential quantifiers.

As mentioned in the introductory section, the universal quantifier is implemented as $\neg\exists\neg$ and the the automaton must be determinized immediately after our projection.

3.7.1 Trailing Zeroes

A crucial aspect of representing predicates with automata is to ensure the automaton treats all representation of the same number assignment equally. For example, suppose $P(13)$ evaluates to true and we are working in base 2, then the automaton M_P accepts `1011` but also need accept `10110`, `101100`, or `101100000`, etc.

Formally, we say that an n-variable automaton M is *zero-input stable* if whenever M accepts $w \in \Sigma^n$ if and only if M also accepts $w\mathbf{0}$ where $\mathbf{0}$ is the letter $(\underbrace{0, 0, \ldots, 0}_{n \text{ times}}) \in \Sigma^n$.

The automata described in the previous sections satisfy this property. However, after applying the Γ_n map to M_P the new automaton M' may not be zero-input stable. To see this, consider that Γ_N maps $(0, 0, \ldots, 0, 1)$ to $(0, 0, \ldots, 0)$. In order to ensure that M' is zero-input stable we must modify the automaton slightly after the above described operation. Specifically, if there exists a path in M' labelled by $\mathbf{0}$'s from a state $q \in Q$ to a state $f \in F$ then we add q to the set of final states F. In this way we ensure that our automata always properly encode their respective predicates.

3.8 Extrema: Minimum and Maximum

Often, a property expressible in our logical theory will demand for the minimum or maximum value such that a certain predicate, say P, is satisfied. Suppose we desire to compute $\min\{x_n \mid P(x_1, x_2, \ldots, x_n)\}$. We can rewrite this as the predicate

$$P(x_1, x_2, \ldots, x_n) \text{ and } \forall y > x_n \text{ we have } \neg P(x_1, x_2, \ldots, y).$$

This observation is enough to show that the predicate '$\min\{x_n \mid P(x_1, x_2, \ldots, x_n)\}$' is also decidable by our methods. However, for the sake of efficiency the author has decided to implement extrema as a primitive operation. It is implemented as follows: given an automaton M_P for a predicate P we apply the map

$$\Lambda_n([a_1, a_2, \ldots, a_n]) = \{[a_1, a_2, \ldots, a_n, b] \mid b \in \Sigma\}$$

to M_P in a manner similar to the one described in the previous Section. This new automaton M' accepts $(x_1, x_2, \ldots, x_n, y)$ such that $P(x_1, x_2, \ldots, x_n)$ is satisfied.

3.9 Program Output

Throughout this thesis we will refer to the computation results of a particular predicate in question. As such, it seems expedient to give a few details about what such output might mean, and how it relates to the original predicate.

Our program logs the state complexity and computation time of each step in the parsed logical predicate. To illustrate we will provide an example. Suppose we are interested in computing the following function:

$$\begin{aligned} A(n) &= \min\{m \ : \ \text{such that } \mathbf{x}[0..m-1] \text{ contains all factors of } \mathbf{x} \text{ of length } n \ \} \\ &= \min\{m \ : \ \forall j \geq 0 \ \exists \ell, \ 0 \leq \ell \leq m-n, \text{ such that } \mathbf{x}[j..j+n-1] = \mathbf{x}[\ell..\ell+n-1]\}. \end{aligned}$$

However, functions are not one of the implemented primitives; instead we will encode the function as the predicate P where $P(n, m)$ if and only if $m = A(n)$.

The predicate is then rewritten in custom infix notation that our program can parse. The above predicate would be expressed as follows:

Here is a summary of the computation:

```
(\min, m,
    (\forall, j,
        (\exists, l,
            (\and,
                (\factor, n, l, j),
                (>=, m, n+l)
            )
        )
    )
)
```

Figure 3.5: An input expression for computing $A(n)$

In fact, `'\factor'` and `'\forall'` are not one our building blocks either. We have built in several oft-used predicates which are replaced with a corresponding expression composed of primitives. Thus, the parser immediately replaces the expression in Figure 3.5 with the expression:

Here is a summary of the computation:

```
(\min, m,
    (\forall, j,
        (\exists, l,
            (\and,
                (\forall, i0
                    (\or,
                        (>=, i0, n),
                        (\out=, l+i0, j+i0)
                    )
                ),
                (>=, m, n+l)
            )
        )
    )
)
```

and yet again replaced with the expression: Here is a summary of the computation:

```
(\min, m,
    (\not,
        (\exists, j,
            (\not,
                (\exists, l,
```

```
                    (\and,
                        (\not,
                            (\exists, i0,
                                (\not,
                                    (\or,
                                        (>=, i0, n),
                                        (\out=, l+i0, j+i0)
                                    )
                                )
                            )
                        ),
                        (>=, m, n+l)
                    )
                )
            )
        )
    )
)
```

which is then finally evaluated.

A typical output of the evaluation of the above expression can be seen in Figure 3.6 below. In the program output, the indentation of the line indicates its nesting within the predicate. Each step records the initial state complexity after the operation is performed as well as the minimal state for the resulting automaton. Finally, the total time for the computation is also recorded.

Here is a summary of the computation:

```
          [>= i0 n] (3 => 2 states) in 0.032s
          [\out= l+i0 j+i0] (8 => 8 states) in 0.093s
        [\or ] (16 => 16 states) in 0.099s
      [\not ] (17 => 16 states) in 0.104s
    [\exists i0] (16 => 22 states) in 0.069s
   [\not ] (23 => 21 states) in 0.079s
   [>= m n+l] (6 => 3 states) in 0.031s
  [\and ] (40 => 40 states) in 0.094s
 [\exists l] (40 => 42 states) in 0.050s
[\not ] (43 => 42 states) in 0.087s
[\exists j] (42 => 11 states) in 0.035s
[\not ] (12 => 11 states) in 0.054s
[\min m] (9 => 9 states) in 0.122s
[total] (9 states) in 1.002s
```

Figure 3.6: Typical program output on the expression in Figure 3.5

3.10 Implementation Details and Statistics

Our framework rests on top of an automata toolbox library named FSA6.2xx: Finite State Automata Utilities (FSA). FSA was originally created by Gertjan van Noord of University of Groningen who released it under the GNU General Public License. The library is written mainly in Prolog and was chosen for its flexibility and relative efficiency. For example, the library natively handles the use of arrays as the input alphabet. Furthermore, it can routinely handle automata with thousands or tens of thousands of states. You may find the sources of FSA at http://www.let.rug.nl/~vannoord/Fsa/.

Our framework was written by the author of this thesis with the guidance of Jeffrey Shallit and occasional brilliant insight of Luke Schaeffer. The vast majority of it is written in PHP, a language known for its flexibility allowing for rapid fleshing-out of prototype applications. Of course, this flexibility comes at a cost of performance. Perhaps, a more ideal implementation in a statically typed language such as C++ or Java can be made in the future. Overall the source code (excluding the FSA library) amounts to 2786 lines spread across 5 files.

One of the bottlenecks in program execution turned out to be reading and parsing the automata output by the FSA library after an operation was performed. To address this problem the author implemented a small efficient parser in Java. The parser reads the automata which it then outputs in JSON, a format natively supported by PHP. However, this addition has only been needed for sequences over a large input alphabet such as the Leech word which is 13-automatic.

A neat feature of our implementation is the caching of intermediate results. At each step in the computation the resulting automaton is cached. The results are stored using a hash on the expression tree and the particular k-automatic sequence in question. This allows for efficient retrieval of already computed results and, furthermore, saves computation time even if two predicates with identical sub-expressions are computed.

Chapter 4

Recurrence

Remark 4. Excerpts of the next 3 chapters are taken essentially verbatim from [56].

4.1 Introduction

A sequence is said to be *recurrent* if every factor that occurs, occurs infinitely often. A sequence $\mathbf{x}$ is said to be *uniformly recurrent* if it is recurrent and furthermore for each finite factor w occurring in $\mathbf{x}$, there is a constant $c(w)$ such that two consecutive occurrences of w are separated by at most $c(w)$ positions.

The *recurrence function* $R_{\mathbf{x}}(n)$ of a sequence $\mathbf{x}$ is the smallest integer m such that every factor of $\mathbf{x}$ of length m contains as a factor all the factors of length n. Formally, we define the recurrence function as follows:

$$\begin{aligned} R_{\mathbf{x}}(n) &= \min\{m \ : \ \text{such that } \forall k \ \mathbf{x}[k..k+m-1] \text{ contains all factors of } \mathbf{x} \text{ of length } n\} \\ &= \min\{m \ : \ \forall\, k, j \geq 0 \ \exists \ell, \ k \leq \ell \leq k+m-n, \\ &\qquad\qquad \text{such that } \mathbf{x}[j..j+n-1] = \mathbf{x}[\ell..\ell+n-1]\}. \end{aligned}$$

This predicate is expressible over $\langle \mathbb{N}, +, <, 0, P_a, V_k \rangle$ and so it can be decided by our methods. Thus we obtain

Theorem 5. *If* $\mathbf{x}$ *is* k*-automatic, then the sequence* $(R_{\mathbf{x}}(n))_{n \geq 0}$ *is* k*-synchronized.*

In Figure 4.1 we illustrate the recurrence window of the Thue-Morse sequence $\mathbf{t}$ for $n = 2$. In this example we see that $R_{\mathbf{t}}(2) = 9$.

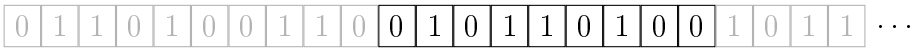


Figure 4.1: The recurrence window of the Thue-Morse sequence containing all factors of length 2

The *recurrence quotient* of a sequence $\mathbf{x}$ is defined as

$$\sup_{n \geq 1} \frac{R_{\mathbf{x}}(n)}{n}.$$

In this chapter we restate the recurrence function and quotient of the Rudin-Shapiro sequence, first presented in [55]. Next, we reaffirm one of the earliest results regarding the recurrence function of the Thue-Morse sequence. The remaining results presented in this chapter are novel contributions of this thesis.

4.2 Rudin-Shapiro

Recall that $\mathbf{r} = (r(n))_{n \geq 0}$ denotes the Rudin-Shapiro sequence. In 1994 Allouche and Bousquet-Mélou gave the estimate for the recurrence quotient of the Rudin-Shapiro sequence as $R_{\mathbf{r}}(n+1) < 172n$ for $n \geq 1$ [3]. (Actually, their result was more general, as it applies to any "generalized" Rudin-Shapiro sequence.) We used our method to compute a new explicit expression for the recurrence function $R_{\mathbf{r}}(n)$ and recurrence quotient:

Theorem 6. *Then*

$$R_{\mathbf{r}}(n) = \begin{cases} 5, & \text{if } n = 1; \\ 19, & \text{if } n = 2; \\ 25, & \text{if } n = 3; \\ 20 \cdot 2^t + n - 1, & \text{if } n \geq 4 \text{ and } t = \lceil \log_2(n-1) \rceil. \end{cases}$$

Furthermore, the recurrence quotient is equal to 41*; it is not attained.*

Proof. First we created a DFA to encoding the recurrence function of the Rudin-Shapiro sequence. The resulting automaton can be seen below in Figure 4.2.

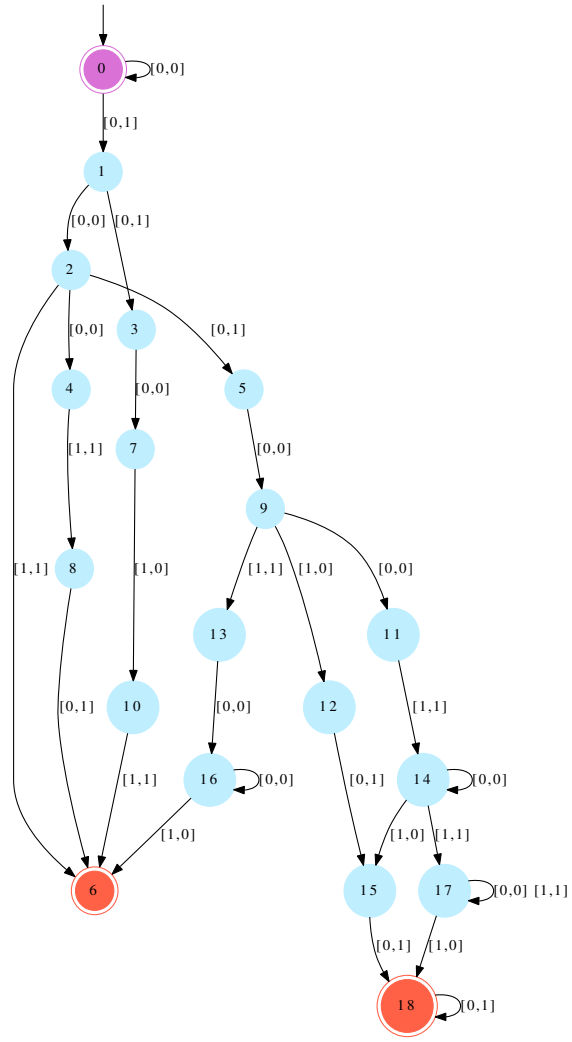


Figure 4.2: An automaton encoding $R_{\mathbf{r}}(n)$

Here is a summary of the computation:

```
   [>= i0 n] (3 => 2 states) in 0.021s
   [\out= l+i0 j+i0] (28 => 28 states) in 0.248s
  [\or ] (56 => 56 states) in 0.284s
 [\not ] (57 => 56 states) in 0.303s
[\exists i0] (56 => 99 states) in 1.269s
```

```
          [\not ] (100 => 98 states) in 0.196s
          [>= m+k n+1] (12 => 4 states) in 0.023s
         [\and ] (273 => 273 states) in 0.336s
         [>= 1 k] (3 => 2 states) in 0.014s
        [\and ] (425 => 425 states) in 0.537s
       [\exists l] (425 => 1818 states) in 22.553s
      [\not ] (1819 => 1818 states) in 2.573s
     [\exists j] (1818 => 323 states) in 2.518s
    [\not ] (324 => 323 states) in 0.193s
   [\not ] (324 => 323 states) in 0.183s
  [\exists k] (323 => 27 states) in 0.043s
 [\not ] (28 => 27 states) in 0.026s
[\min m] (16 => 16 states) in 0.075s
[total] (16 states) in 31.996s
```

We then created a DFA to accept

$$\{(n)_2 \ : \exists\ m \text{ such that } m = 20 \cdot 2^t + n - 1 \text{ and } m = R_{\mathbf{r}}(n) \text{ and } t = \lceil \log_2(n-1) \rceil\}.$$

We then verified that the resulting DFA accepted all $(n)_2$ for $n \geq 4$.

For the recurrence quotient, the local maximum is evidently achieved when $n = 2^r + 2$ for some $r \geq 1$; here it is equal to $(41 \cdot 2^r + 2)/(2^r + 2)$. As $r \to \infty$, this clearly approaches 41 from below.

□

4.3 Thue-Morse

The recurrence function of the Thue-Morse sequence was first studied by M. Morse and G.A. Hedlund in their 1938 article entitled *Symbolic Dynamics* [77]. In section 8 of the article they laboriously derive the recurrence function and give the recurrence quotient. Below, we confirm their results using our automated method.

Theorem 7. *For the Thue-Morse sequence, we have*

$$R_{\mathbf{t}}(n) = \begin{cases} 3, & \text{if } n = 1; \\ 9, & \text{if } n = 2; \\ 9 \cdot 2^{t-1} + n - 1, & \text{if } n \geq 3 \text{ and } t = \lceil \log_2(n-1) \rceil. \end{cases}$$

Furthermore, the recurrence quotient is equal to 10*; it is not attained.*

Proof. The resulting automaton encoding the recurrence function of the Thue-Morse sequence can be seen below in Figure 4.3.

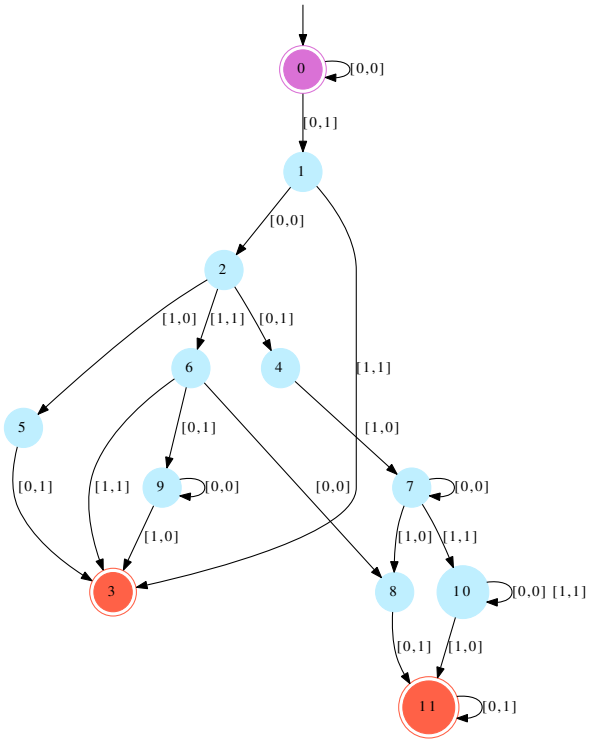


Figure 4.3: An automaton encoding $R_{\mathbf{t}}(n)$

The local maximum of the recurrence quotient is achieved when $n = 2^r + 2$ for some $r \geq 1$; here it is equal to

$$\frac{9 \cdot 2^r + (2^r + 2) - 1}{2^r + 1} = \frac{10 \cdot 2^r + 1}{2^r + 1}.$$

As $r \to \infty$, this clearly approaches 10 from below.

Here is a summary of the computation:

```
[>= i0 n] (3 => 2 states) in 0.041s
[\out= 1+i0 j+i0] (8 => 8 states) in 0.147s
```

```
            [\or ] (16 => 16 states) in 0.153s
           [\not ] (17 => 16 states) in 0.171s
          [\exists i0] (16 => 22 states) in 0.111s
         [\not ] (23 => 21 states) in 0.120s
         [>= m+k n+1] (12 => 4 states) in 0.057s
        [\and ] (61 => 61 states) in 0.162s
        [>= 1 k] (3 => 2 states) in 0.032s
       [\and ] (97 => 97 states) in 0.249s
      [\exists l] (97 => 268 states) in 0.588s
     [\not ] (269 => 268 states) in 0.517s
    [\exists j] (268 => 68 states) in 0.200s
   [\not ] (69 => 68 states) in 0.094s
  [\not ] (69 => 68 states) in 0.103s
 [\exists k] (68 => 18 states) in 0.039s
[\not ] (19 => 18 states) in 0.052s
[\min m] (13 => 13 states) in 0.153s
[total] (13 states) in 3.165s
```

□

4.4 Paperfolding

Theorem 8. *For the regular paperfolding sequence, we have*

$$R_{\mathbf{p}}(n) = \begin{cases} 4, & \text{if } n = 1; \\ 9, & \text{if } n = 2; \\ 8 \cdot 2^t + n - 1, & \text{if } n \geq 3 \text{ and } t = \lceil \log_2(n) \rceil. \end{cases}$$

Furthermore, the recurrence quotient is equal to 17*; it is not attained.*

Proof. The resulting automaton encoding the recurrence function of the paperfolding sequence can be seen below in Figure 4.4.

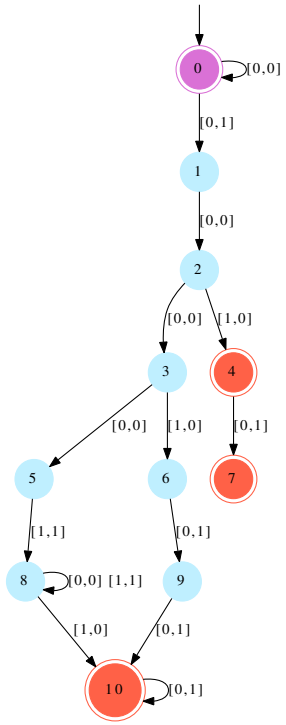


Figure 4.4: An automaton encoding $R_{\mathbf{p}}(n)$

The local maximum of the recurrence quotient is achieved when $n = 2^r + 1$ for some $r \geq 1$; here it is equal to

$$\frac{8 \cdot 2^{(r+1)} + (2^r + 1) - 1}{2^r + 1} = \frac{17 \cdot 2^r}{2^r + 1}.$$

As $r \to \infty$, this ratio approaches 17 from below.

Here is a summary of the computation:

```
    [>= i0 n] (3 => 2 states) in 0.047s
    [\out= l+i0 j+i0] (33 => 15 states) in 0.400s
   [\or ] (31 => 31 states) in 0.268s
  [\not ] (32 => 30 states) in 0.257s
 [\exists i0] (30 => 61 states) in 0.398s
[\not ] (62 => 60 states) in 0.214s
[>= m+k n+1] (12 => 4 states) in 0.042s
```

```
        [\and ] (157 => 157 states) in 0.295s
        [>= 1 k] (3 => 2 states) in 0.033s
       [\and ] (241 => 241 states) in 0.428s
      [\exists l] (241 => 433 states) in 3.170s
     [\not ] (434 => 433 states) in 0.867s
    [\exists j] (433 => 155 states) in 0.678s
   [\not ] (156 => 155 states) in 0.175s
  [\not ] (156 => 155 states) in 0.168s
 [\exists k] (155 => 15 states) in 0.047s
[\not ] (16 => 15 states) in 0.045s
[\min m] (9 => 9 states) in 0.156s
[total] (9 states) in 7.992s
```

□

4.5 Period-Doubling

Theorem 9. *For the period-doubling sequence, we have*

$$R_{\mathbf{d}}(n) = \begin{cases} 4, & \text{if } n = 1; \\ 7 \cdot 2^{t-1} + n - 1, & \text{if } n \geq 2 \text{ and } t = \lceil \log_2(n) \rceil. \end{cases}$$

Furthermore, the recurrence quotient is equal to 8*; it is not attained.*

Proof. The resulting automaton encoding the recurrence function of the period-doubling sequence can be seen below in Figure 4.5.

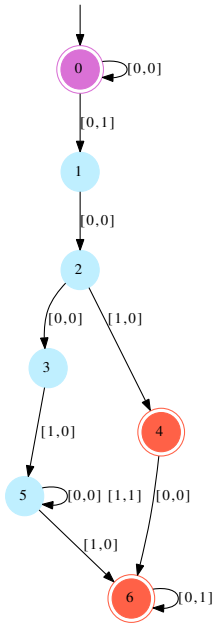


Figure 4.5: An automaton encoding $R_{\mathbf{d}}(n)$

The local maximum of the recurrence quotient is achieved when $n = 2^r + 1$ for some $r \geq 1$; here it is equal to

$$\frac{7 \cdot 2^r + (2^r + 1) - 1}{2^r + 1} = \frac{8 \cdot 2^r}{2^r + 1}.$$

As $r \to \infty$, this ratio approaches 8 from below.

Here is a summary of the computation:

```
          [>= i0 n] (3 => 2 states) in 0.037s
          [\out= l+i0 j+i0] (13 => 6 states) in 0.235s
        [\or ] (13 => 13 states) in 0.142s
       [\not ] (14 => 12 states) in 0.132s
      [\exists i0] (12 => 11 states) in 0.049s
     [\not ] (12 => 10 states) in 0.089s
     [>= m+k n+l] (12 => 4 states) in 0.047s
    [\and ] (30 => 30 states) in 0.107s
    [>= l k] (3 => 2 states) in 0.037s
   [\and ] (50 => 50 states) in 0.141s
  [\exists l] (50 => 73 states) in 0.105s
 [\not ] (74 => 73 states) in 0.176s
[\exists j] (73 => 66 states) in 0.070s
```

```
    [\not ] (67 => 66 states) in 0.096s
   [\not ] (67 => 66 states) in 0.102s
  [\exists k] (66 => 12 states) in 0.041s
 [\not ] (13 => 12 states) in 0.053s
[\min m] (11 => 11 states) in 0.143s
[total] (11 states) in 1.896s
```

□

4.6 Baum-Sweet

Theorem 10. *The Baum-Sweet sequence is not uniformly recurrent. In fact, let* $N_{\mathbf{x}}(i)$ *be the smallest integer* n *such that* $\mathbf{x}[i..i+n-1]$ *does not occur later in* $\mathbf{x}$*. Then*

$$N_{\mathbf{b}}(i) = \begin{cases} 2 \cdot 2^t - n + 2, & \text{if } t = \lceil \log_2(n+1) \rceil \text{ is even;} \\ 2 \cdot 2^t - n + 1, & \text{if } t = \lceil \log_2(n+1) \rceil \text{ is odd.} \end{cases}$$

Proof. We computed the set of pairs (i, n) such that the factor of length n at position i doesn't occur anywhere later in the sequence and furthermore, n is the shortest such length at position i. Here is the function expressed in our logical theory:

$$N_{\mathbf{b}}(i) = \min\{n \ : \text{ such that } \neg \exists\ \ell > i \text{ where } \mathbf{b}[i..i+n-1] = \mathbf{b}[\ell..\ell+n-1]\}$$

The resulting automaton encoding the the function $N_{\mathbf{b}}(i)$ can be seen below in Figure 4.6. Thus, for example, $\mathbf{b}[1..5] = \texttt{1011}$ does not occur anywhere else in this sequence.

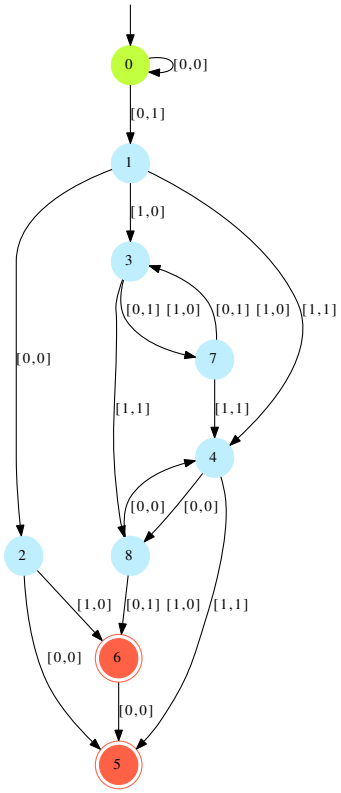


Figure 4.6: An automaton encoding the location and size of factors that do not occur at a later position in the Baum-Sweet sequence

Here is a summary of the computation:

```
      [>= i0 n] (3 => 2 states) in 0.020s
      [\out= l+i0 i+i0] (36 => 25 states) in 0.126s
    [\or ] (49 => 49 states) in 0.119s
   [\not ] (50 => 48 states) in 0.113s
  [\exists i0] (48 => 301 states) in 2.323s
 [\not ] (302 => 301 states) in 0.277s
 [> l i] (3 => 2 states) in 0.025s
[\and ] (357 => 357 states) in 0.331s
[\exists l] (357 => 15 states) in 0.071s
[\not ] (16 => 15 states) in 0.063s
[\min n] (12 => 12 states) in 0.128s
[total] (12 states) in 3.691s
```

To confirm that the function $N_{\mathbf{b}}(i)$ matches the description given in the theorem, we computed the DFA for the predicate

$$\{i \ : \exists\ n \text{ such that } n = N_{\mathbf{b}}(i) \text{ and, either } n = 2 \cdot 2^t - n + 2 \text{ and } t \text{ is even;} \\ \text{or } n = 2 \cdot 2^t - n + 1 \text{ and } t \text{ is odd} \\ \text{where } t = \lceil \log_2(n-1) \rceil\}.$$

We then verified that the resulting DFA accepted all $i \geq 0$.

□

4.7 Mephisto Waltz

Theorem 11. *For the Mephisto waltz sequence, we have*

$$R_{\mathbf{m}}(n) = \begin{cases} 4, & \text{if } n = 1; \\ 11 \cdot 3^t + n - 1, & \text{if } n \geq 2 \text{ and } t = \lceil \log_3(n-1) \rceil. \end{cases}$$

Furthermore, the recurrence quotient is equal to 34*; it is not attained.*

Proof. The resulting automaton encoding the recurrence function of the Mephisto waltz sequence can be seen below in Figure 4.7.

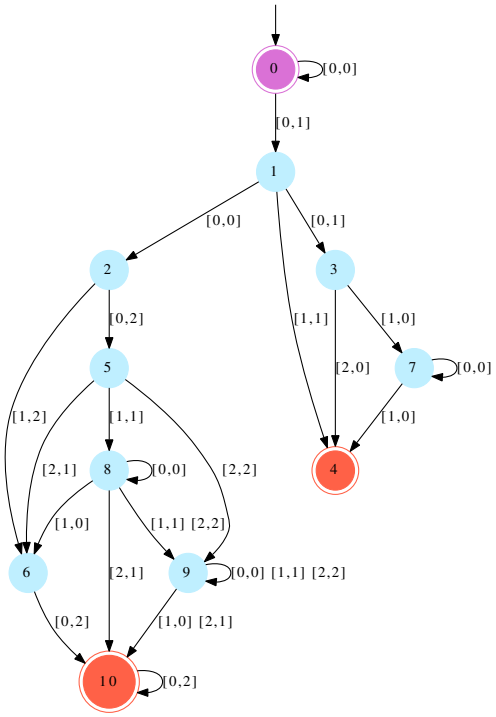


Figure 4.7: An automaton encoding $R_{\mathbf{m}}(n)$

The local maximum of the recurrence quotient is achieved when $n = 3^r + 2$ for some $r \geq 1$; here it is equal to

$$\frac{11 \cdot 3^{(r+1)} + (3^r + 2) - 1}{3^r + 2} = \frac{34 \cdot 3^r + 1}{3^r + 2}.$$

As $r \to \infty$, this ratio approaches 34 from below.

Here is a summary of the computation:

```
    [>= i0 n] (3 => 2 states) in 0.158s
    [\out= l+i0 j+i0] (8 => 8 states) in 1.117s
   [\or ] (16 => 16 states) in 1.577s
  [\not ] (17 => 16 states) in 1.755s
 [\exists i0] (16 => 20 states) in 0.742s
[\not ] (21 => 19 states) in 0.459s
[>= m+k n+1] (12 => 4 states) in 0.180s
```

```
        [\and ] (56 => 56 states) in 0.806s
        [>= 1 k] (3 => 2 states) in 0.077s
       [\and ] (89 => 89 states) in 1.244s
      [\exists l] (89 => 226 states) in 2.066s
     [\not ] (227 => 226 states) in 2.328s
    [\exists j] (226 => 81 states) in 0.607s
   [\not ] (82 => 81 states) in 0.250s
  [\not ] (82 => 81 states) in 0.256s
 [\exists k] (81 => 15 states) in 0.072s
[\not ] (16 => 15 states) in 0.056s
[\min m] (9 => 9 states) in 0.186s
[total] (9 states) in 14.994s
```

□

4.8 Stewart Choral

Theorem 12. *For the Stewart choral sequence, we have*

$$R_{\mathbf{s}}(n) = \begin{cases} 3, & \text{if } n = 1; \\ 3 \cdot 3^t + n - 1, & \text{if } n \geq 2 \text{ and } t = \lceil \log_3(n) \rceil. \end{cases}$$

Furthermore, the recurrence quotient is equal to 10*; it is not attained.*

Proof. The resulting automaton encoding the recurrence function of the Stewart choral sequence can be seen below in Figure 4.8.

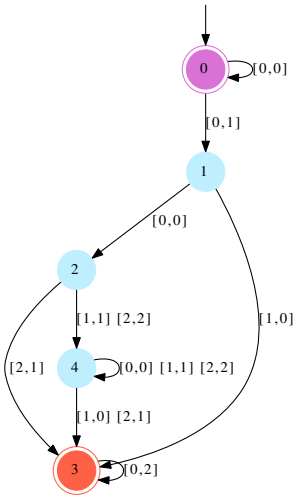


Figure 4.8: An automaton encoding $R_{\mathbf{s}}(n)$

The local maximum of the recurrence quotient is achieved when $n = 3^r + 1$ for some $r \geq 1$; here it is equal to

$$\frac{3 \cdot 3^{(r+1)} + (3^r + 1) - 1}{3^r + 1} = \frac{10 \cdot 3^r}{3^r + 1}.$$

As $r \to \infty$, this ratio approaches 10 from below.

Here is a summary of the computation:

```
          [>= i0 n] (3 => 2 states) in 0.161s
          [\out= l+i0 j+i0] (22 => 11 states) in 2.130s
        [\or ] (23 => 23 states) in 2.261s
       [\not ] (24 => 22 states) in 2.138s
      [\exists i0] (22 => 14 states) in 0.661s
     [\not ] (15 => 13 states) in 0.344s
     [>= m+k n+1] (12 => 4 states) in 0.177s
    [\and ] (46 => 46 states) in 0.634s
    [>= l k] (3 => 2 states) in 0.076s
   [\and ] (72 => 72 states) in 1.012s
  [\exists l] (72 => 110 states) in 1.160s
 [\not ] (111 => 110 states) in 1.075s
[\exists j] (110 => 64 states) in 0.343s
[\not ] (65 => 64 states) in 0.203s
[\not ] (65 => 64 states) in 0.202s
[\exists k] (64 => 8 states) in 0.047s
```

```
 [\not ] (9 => 8 states) in 0.053s
[\min m] (6 => 6 states) in 0.162s
[total] (6 states) in 13.700s
```

□

Chapter 5

Appearance

5.1 Introduction

The *appearance function* $A_{\mathbf{x}}(n)$ of a sequence $\mathbf{x}$ is the smallest integer m such that the *prefix* $\mathbf{x}[0 \ldots m-1]$ contains as a factor all the factors of length n. The appearance function was first introduced and studied by Allouche and Shallit in Section 10.10 of [10]. Formally, we define the appearance function as follows:

$$\begin{aligned} A_{\mathbf{x}}(n) &= \min\{m \ : \ \text{such that } \mathbf{x}[0..m-1] \text{ contains all factors of } \mathbf{x} \text{ of length } n\} \\ &= \min\{m \ : \ \forall j \geq 0 \ \exists \ell, \ 0 \leq \ell \leq m-n, \ \text{such that } \mathbf{x}[j..j+n-1] = \mathbf{x}[\ell..\ell+n-1]\}. \end{aligned}$$

This predicate is also expressible over $\langle \mathbb{N}, +, <, 0, P_a, V_k \rangle$ and so it can be decided by our methods. Thus we obtain

Theorem 13. *If* $\mathbf{x}$ *is* k*-automatic, then the sequence* $(A_{\mathbf{x}}(n))_{n \geq 0}$ *is* k*-synchronized.*

In Figure 5.1 we illustrate the appearance window of the Thue-Morse sequence $\mathbf{t}$ for $n = 2$. In this example we see that $A_{\mathbf{t}}(2) = 7$.

Figure 5.1: The appearance window of the Thue-Morse sequence containing all factors of length 2

In Example 10.10.3 of [10] Allouche and Shallit give a formal description of the appearance function of the Thue-Morse sequence. The remainder of the appearance functions presented in this chapter were not previously known.

5.2 Thue-Morse

Theorem 14. *For the Thue-Morse sequence, we have*

$$A_{\mathbf{t}}(n) = \begin{cases} 2, & \text{if } n = 1; \\ 7, & \text{if } n = 2; \\ 3 \cdot 2^t + n - 1, & \text{if } n \geq 3 \text{ and } t = \lceil \log_2(n-1) \rceil. \end{cases}$$

Proof. First we created a DFA to encode the appearance function of the Thue-Morse sequence. The resulting automaton can be seen below in Figure 5.2.

We then created a DFA to accept

$$\{(n)_2 \ : \ \exists \ m \text{ such that } m = 3 \cdot 2^t + n - 1 \text{ and } m = A_{\mathbf{t}}(n) \text{ and } t = \lceil \log_2(n-1) \rceil\}.$$

We then verified that the resulting DFA accepted all $(n)_2$ for $n \geq 4$.

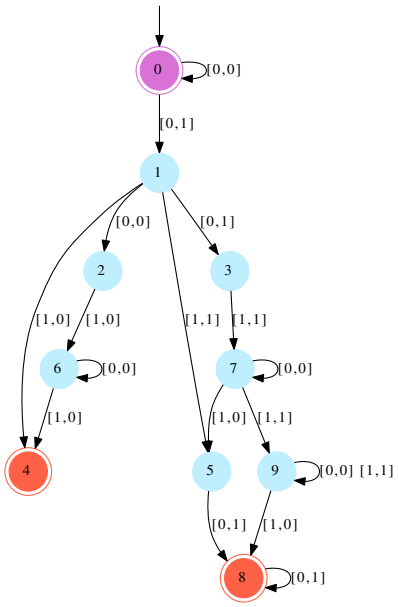


Figure 5.2: An automaton encoding $A_{\mathbf{t}}(n)$

Here is a summary of the computation:

```
          [>= i0 n] (3 => 2 states) in 0.032s
          [\out= l+i0 j+i0] (8 => 8 states) in 0.093s
        [\or ] (16 => 16 states) in 0.099s
      [\not ] (17 => 16 states) in 0.104s
    [\exists i0] (16 => 22 states) in 0.069s
   [\not ] (23 => 21 states) in 0.079s
   [>= m n+1] (6 => 3 states) in 0.031s
  [\and ] (40 => 40 states) in 0.094s
 [\exists l] (40 => 42 states) in 0.050s
[\not ] (43 => 42 states) in 0.087s
[\exists j] (42 => 11 states) in 0.035s
[\not ] (12 => 11 states) in 0.054s
[\min m] (9 => 9 states) in 0.122s
[total] (9 states) in 1.002s
```

□

5.3 Rudin-Shapiro

Theorem 15. *For the Rudin-Shapiro sequence, we have*

$$A_{\mathbf{r}}(n) = \begin{cases} 4, & \text{if } n = 1; \\ 13, & \text{if } n = 2; \\ 16, & \text{if } n = 3; \\ 13 \cdot 2^t + n - 1, & \text{if } n \geq 4 \text{ and } t = \lceil \log_2(n-1) \rceil. \end{cases}$$

Proof. We applied the same technique for the Rudin-Shapiro sequence.

The resulting automaton encoding the appearance function of the Rudin-Shapiro sequence can be seen below in Figure 5.3.

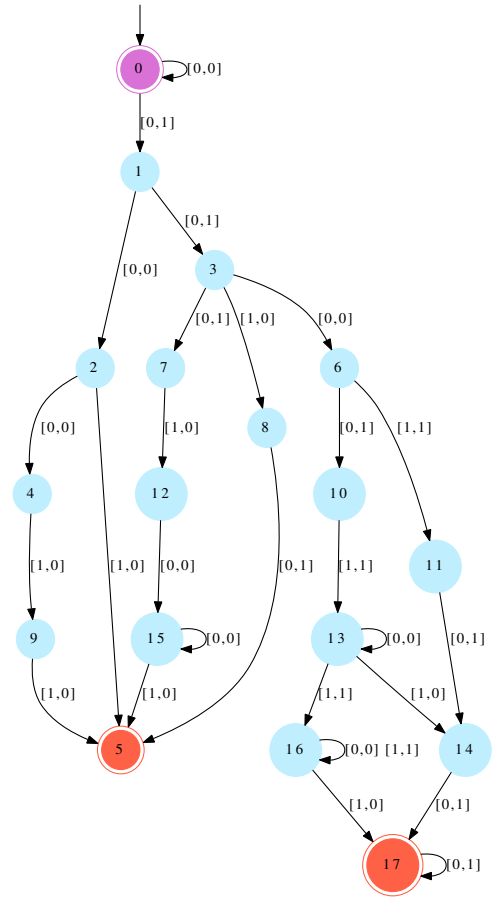


Figure 5.3: An automaton encoding $A_{\mathbf{r}}(n)$

Here is a summary of the computation:

```
       [>= i0 n] (3 => 2 states) in 0.029s
       [\out= l+i0 j+i0] (28 => 28 states) in 0.221s
      [\or ] (56 => 56 states) in 0.234s
     [\not ] (57 => 56 states) in 0.229s
    [\exists i0] (56 => 99 states) in 1.239s
   [\not ] (100 => 98 states) in 0.173s
   [>= m n+1] (6 => 3 states) in 0.025s
  [\and ] (184 => 184 states) in 0.196s
 [\exists l] (184 => 199 states) in 0.223s
[\not ] (200 => 199 states) in 0.195s
[\exists j] (199 => 20 states) in 0.056s
```

```
 [\not ] (21 => 20 states) in 0.051s
[\min m] (16 => 16 states) in 0.147s
[total] (16 states) in 3.141s
```

□

5.4 Paperfolding

Theorem 16. *For the regular paperfolding sequence, we have*

$$A_{\mathbf{p}}(n) = \begin{cases} 3, & \text{if } n = 1; \\ 7, & \text{if } n = 2; \\ 24, & \text{if } n = 3; \\ 25, & \text{if } n = 4; \\ 6 \cdot 2^t + n - 1, & \text{if } n \geq 5 \text{ and } t = \lceil \log_2(n) \rceil. \end{cases}$$

Proof. The resulting automaton encoding the appearance function of the paperfolding sequence can be seen below in Figure 5.4.

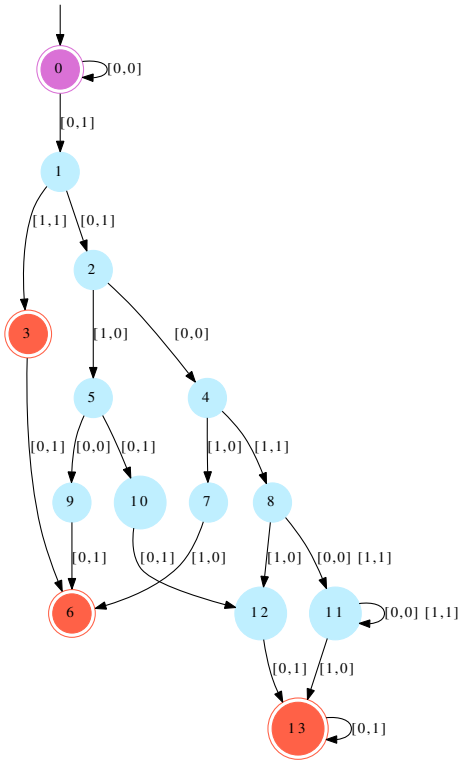


Figure 5.4: An automaton encoding $A_{\mathbf{p}}(n)$

Here is a summary of the computation:

```
        [>= i0 n] (3 => 2 states) in 0.028s
        [\out= l+i0 j+i0] (33 => 15 states) in 0.219s
       [\or ] (31 => 31 states) in 0.150s
      [\not ] (32 => 30 states) in 0.138s
     [\exists i0] (30 => 61 states) in 0.221s
    [\not ] (62 => 60 states) in 0.120s
    [>= m n+1] (6 => 3 states) in 0.040s
   [\and ] (107 => 107 states) in 0.123s
  [\exists l] (107 => 51 states) in 0.070s
 [\not ] (52 => 51 states) in 0.089s
[\exists j] (51 => 19 states) in 0.041s
[\not ] (20 => 19 states) in 0.043s
[\min m] (13 => 13 states) in 0.146s
[total] (13 states) in 1.497s
```

□

5.5 Period-doubling

Theorem 17. *For the period-doubling sequence, we have*

$$A_{\mathbf{d}}(n) = \begin{cases} 2, & \text{if } n = 1; \\ 3 \cdot 2^{t-1} + n - 1, & \text{if } n \geq 2 \text{ and } t = \lceil \log_2(n) \rceil. \end{cases}$$

Proof. The resulting automaton encoding the appearance function of the period-doubling sequence can be seen below in Figure 5.5.

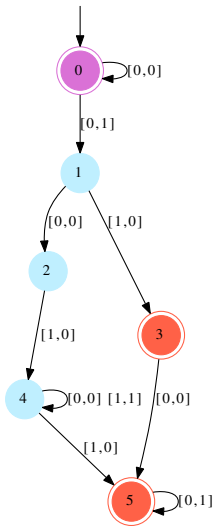


Figure 5.5: An automaton encoding $A_{\mathbf{d}}(n)$

Here is a summary of the computation:

```
      [>= i0 n] (3 => 2 states) in 0.033s
      [\out= 1+i0 j+i0] (13 => 6 states) in 0.131s
     [\or ] (13 => 13 states) in 0.081s
    [\not ] (14 => 12 states) in 0.100s
   [\exists i0] (12 => 11 states) in 0.036s
  [\not ] (12 => 10 states) in 0.063s
  [>= m n+1] (6 => 3 states) in 0.027s
 [\and ] (20 => 20 states) in 0.068s
[\exists 1] (20 => 13 states) in 0.030s
[\not ] (14 => 13 states) in 0.052s
```

```
  [\exists j] (13 => 9 states) in 0.024s
 [\not ] (10 => 9 states) in 0.047s
[\min m] (9 => 9 states) in 0.124s
[total] (9 states) in 0.857s
```

□

5.6 Baum-Sweet

Theorem 18. *For the Baum-Sweet sequence, we have*

$$A_{\mathbf{b}}(n) = \begin{cases} 3, & \text{if } n = 1; \\ 7, & \text{if } n = 2; \\ 23, & \text{if } n = 3; \\ 26, & \text{if } n = 4; \\ 27, & \text{if } n = 5; \\ 46, & \text{if } n = 6; \\ 49, & \text{if } n = 7; \\ 50, & \text{if } n = 8; \\ 89, & \text{if } n = 9; \\ 98, & \text{if } n = 10; \\ 99, & \text{if } n = 11; \\ 100, & \text{if } n = 12; \\ 23 \cdot 2^t + n - 1, & \text{if } n \geq 13,\ 2^t = (n-8) \text{ and } t \text{ is odd}; \\ 23 \cdot 2^{t-1} + n - 1, & \text{otherwise, where } t = \lceil \log_2(n-8) \rceil\ . \end{cases}$$

Proof. The resulting automaton encoding the appearance function of the Baum-Sweet sequence can be seen below in Figure 5.6.

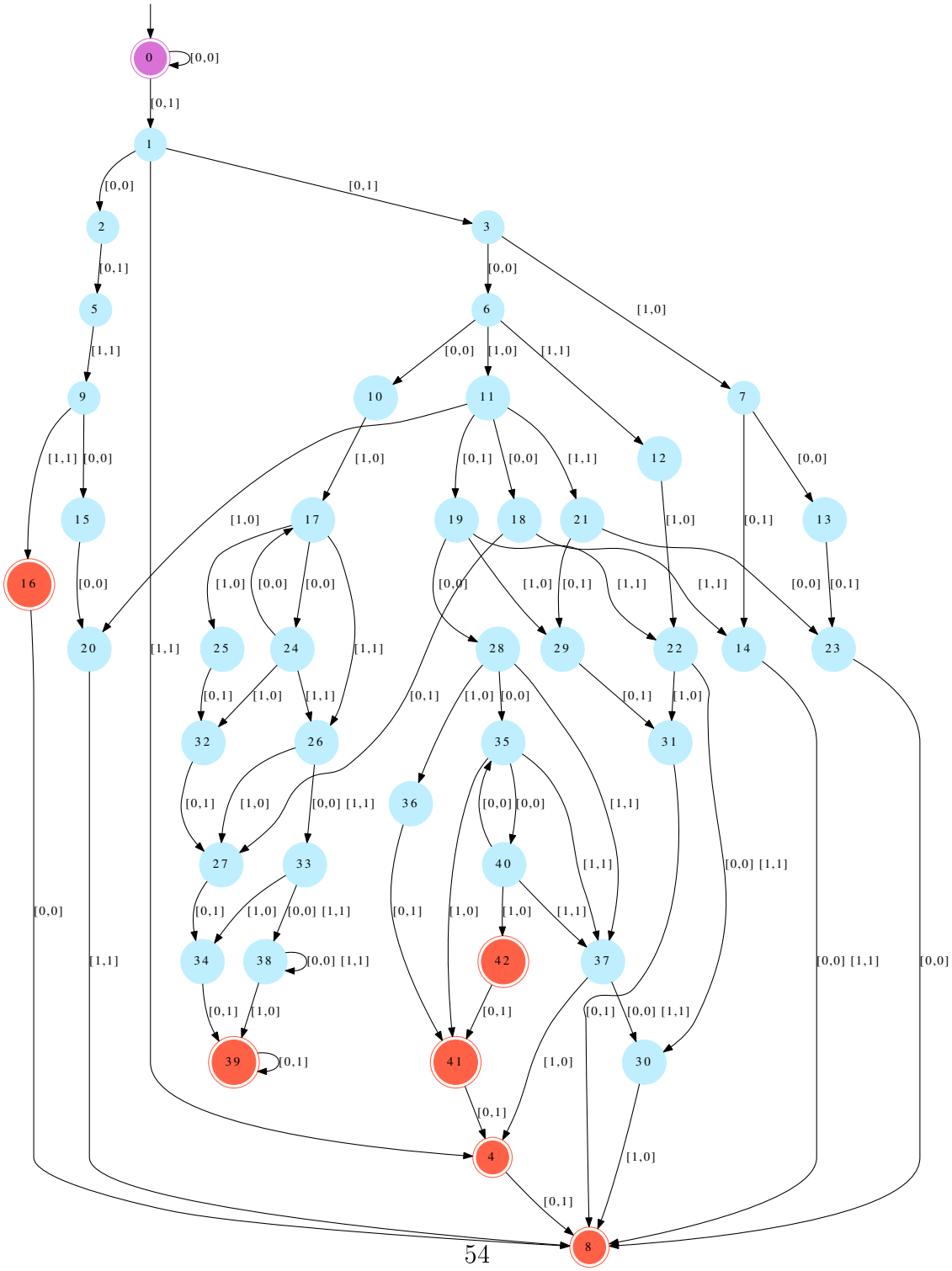


Figure 5.6: An automaton encoding $A_{\mathbf{b}}(n)$

Here is a summary of the computation:

```
          [>= i0 n] (3 => 2 states) in 0.022s
          [\out= l+i0 j+i0] (36 => 25 states) in 0.214s
        [\or ] (49 => 49 states) in 0.203s
       [\not ] (50 => 48 states) in 0.195s
      [\exists i0] (48 => 301 states) in 4.514s
     [\not ] (302 => 301 states) in 0.569s
     [>= m n+1] (6 => 3 states) in 0.039s
    [\and ] (589 => 589 states) in 1.214s
   [\exists l] (589 => 1231 states) in 2.412s
  [\not ] (1232 => 1231 states) in 1.196s
 [\exists j] (1231 => 71 states) in 0.467s
[\not ] (72 => 71 states) in 0.085s
[\min m] (30 => 30 states) in 0.233s
[total] (30 states) in 11.805s
```

□

5.7 Mephisto Waltz

Theorem 19. *For the Mephisto waltz sequence, we have*

$$A_{\mathbf{m}}(n) = \begin{cases} 3, & \text{if } n = 1; \\ 7, & \text{if } n = 2; \\ 17 \cdot 3^{t-1} + n - 1, & \text{if } n \geq 3,\ t = \lceil \log_3(n-1) \rceil \text{ and } n-2 \text{ begins with a } 1; \\ 18 \cdot 3^{t-1} + n - 1, & \text{otherwise.} \end{cases}$$

Proof. The resulting automaton encoding the appearance function of the Mephisto waltz sequence can be seen below in Figure 5.7.

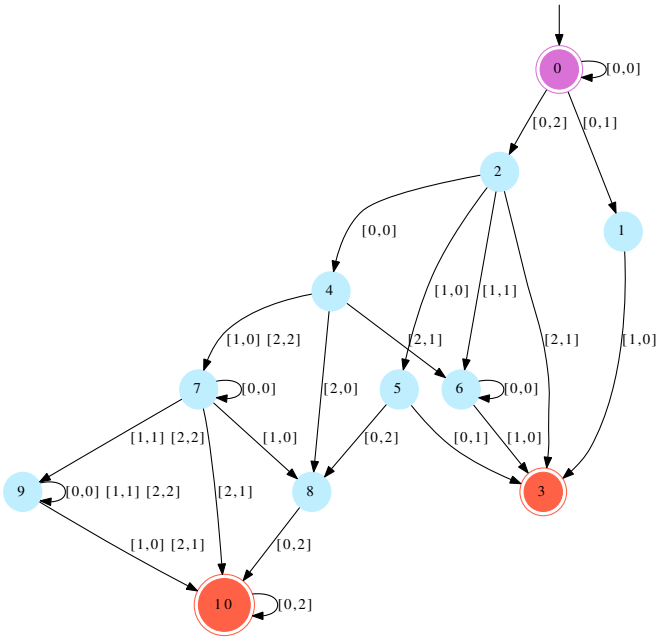


Figure 5.7: An automaton encoding $A_{\mathbf{m}}(n)$

Here is a summary of the computation:

```
        [>= i0 n] (3 => 2 states) in 0.074s
        [\out= l+i0 j+i0] (8 => 8 states) in 0.356s
       [\or ] (16 => 16 states) in 0.485s
      [\not ] (17 => 16 states) in 0.483s
     [\exists i0] (16 => 20 states) in 0.233s
    [\not ] (21 => 19 states) in 0.171s
    [>= m n+l] (6 => 3 states) in 0.051s
   [\and ] (37 => 37 states) in 0.166s
  [\exists l] (37 => 46 states) in 0.096s
 [\not ] (47 => 46 states) in 0.163s
[\exists j] (46 => 14 states) in 0.052s
[\not ] (15 => 14 states) in 0.067s
[\min m] (12 => 12 states) in 0.192s
[total] (12 states) in 2.759s
```

□

5.8 Stewart Choral

Theorem 20. *For the Stewart choral sequence, we have*

$$A_{\mathbf{s}}(n) = \begin{cases} 3, & \text{if } n = 1; \\ \dfrac{5 \cdot 3^t - 1}{2} + n, & \text{if } n \geq 2,\ t = \lceil \log_3(n) \rceil. \end{cases}$$

Proof. The resulting automaton encoding the appearance function of the Stewart choral sequence can be seen below in Figure 5.8.

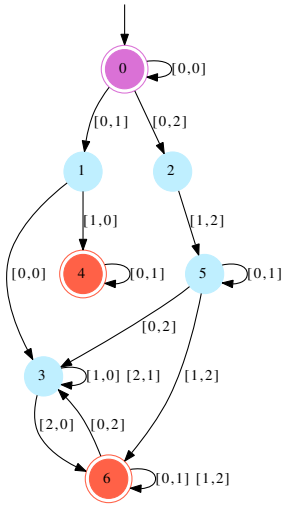


Figure 5.8: An automaton encoding $A_{\mathbf{s}}(n)$

Here is a summary of the computation:

```
      [>= i0 n] (3 => 2 states) in 0.067s
      [\out= l+i0 j+i0] (22 => 11 states) in 0.617s
     [\or ] (23 => 23 states) in 0.662s
    [\not ] (24 => 22 states) in 0.637s
   [\exists i0] (22 => 14 states) in 0.210s
  [\not ] (15 => 13 states) in 0.135s
  [>= m n+1] (6 => 3 states) in 0.057s
 [\and ] (33 => 33 states) in 0.155s
[\exists l] (33 => 24 states) in 0.064s
```

```
   [\not ] (25 => 24 states) in 0.107s
  [\exists j] (24 => 10 states) in 0.035s
 [\not ] (11 => 10 states) in 0.055s
[\min m] (7 => 7 states) in 0.156s
[total] (7 states) in 3.131s
```

In order to confirm that the appearance function is as described above, we created a DFA to accept

$$\{(n)_3 \ : \ \exists\ m \text{ such that } 2m = 5 \cdot 3^t + 2n - 1 \text{ and } m = A_{\mathbf{s}}(n) \text{ where } t = \lceil \log_3(n) \rceil\}.$$

We then verified that the resulting DFA accepted all $(n)_2$ for $n \geq 2$.

□

Chapter 6

Condensation

6.1 Introduction

An infinite word $\mathbf{x}$ is *linearly recurrent* if there is a constant C such that every word of length n appears as a factor of every factor of w of length Cn. Thus, for the optimal C, the quantity Cn can be regarded as the maximum, over all "window sizes" beginning at each position, that minimally contain all length-n factors (no smaller "window" works).

We could instead compute the *minimum*, over all starting positions and the size of all windows that contain all length-n factors. We call this the *condensation function* of $\mathbf{x}$ and formally define it as follows:

$$\begin{aligned} C_{\mathbf{x}}(n) &= \min\{m \ : \ \text{such that } \exists k \ \mathbf{x}[k..k+m-1] \text{ contains all factors of } \mathbf{x} \text{ of length } n\} \\ &= \min\{m \ : \ \exists \ k \ \forall \ j \ \exists \ell, \ k \le \ell \le k+m-n, \\ &\qquad \text{such that } \mathbf{x}[j..j+n-1] = \mathbf{x}[\ell..\ell+n-1]\}. \end{aligned}$$

This predicate is expressible over $\langle \mathbb{N}, +, <, 0, P_a, V_k \rangle$ and so it can be decided by our methods. Thus we obtain

Theorem 21. *If $\mathbf{x}$ is k-automatic, then the sequence $(C_{\mathbf{x}}(n))_{n \ge 0}$ is k-synchronized.*

In Figure 6.1 we illustrate the condensation window of the Thue-Morse sequence $\mathbf{t}$ for $n = 2$. In this example we see that $C_{\mathbf{t}}(2) = 5$.

Figure 6.1: The condensation window of the Thue-Morse sequence containing all factors of length 2

Furthermore, we computed the condensation function for some sequences of interest. The condensation functions of the Thue-Morse, Rudin-Shapiro, paperfolding, and period-doubling sequence were previously presented in [55], the remaining functions are novel.

6.2 Thue-Morse

Theorem 22. *For the Thue-Morse sequence, we have*

$$C_{\mathbf{t}}(n) = \begin{cases} 2, & \text{if } n = 1; \\ 5, & \text{if } n = 2; \\ 2^{t+1} + 2n - 2, & \text{if } n \geq 3 \text{ and } t = \lceil \log_2(n-1) \rceil. \end{cases}$$

Proof. The resulting automaton encoding the condensation function of the Thue-Morse sequence can be seen below in Figure 6.2.

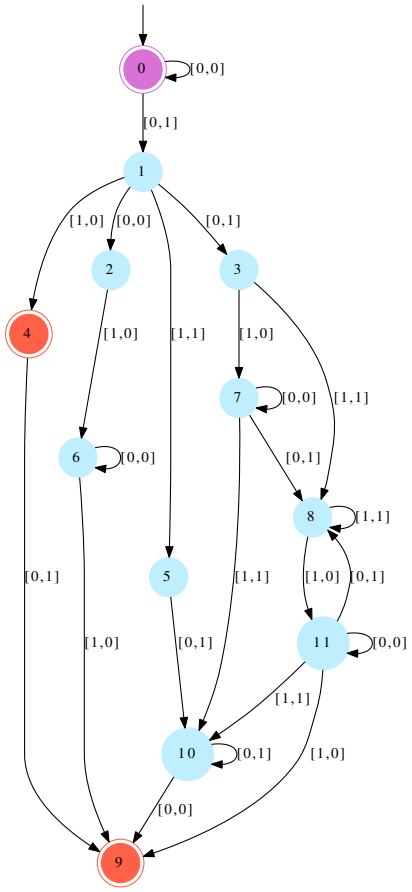


Figure 6.2: An automaton encoding $C_{\mathbf{t}}(n)$

Here is a summary of the computation:

```
      [>= i0 n] (3 => 2 states) in 0.034s
      [\out= l+i0 j+i0] (8 => 8 states) in 0.148s
     [\or ] (16 => 16 states) in 0.157s
    [\not ] (17 => 16 states) in 0.163s
   [\exists i0] (16 => 22 states) in 0.110s
  [\not ] (23 => 21 states) in 0.108s
  [>= m+k n+1] (12 => 4 states) in 0.047s
 [\and ] (61 => 61 states) in 0.168s
 [>= l k] (3 => 2 states) in 0.049s
[\and ] (97 => 97 states) in 0.242s
[\exists l] (97 => 268 states) in 0.600s
```

```
    [\not ] (269 => 268 states) in 0.526s
   [\exists j] (268 => 68 states) in 0.188s
  [\not ] (69 => 68 states) in 0.110s
 [\exists k] (68 => 11 states) in 0.047s
[\min m] (12 => 12 states) in 0.115s
[total] (12 states) in 2.987s
```

□

6.3 Rudin-Shapiro

Theorem 23. *For the Rudin-Shapiro sequence, we have*

$$C_{\mathbf{r}}(n) = \begin{cases} 2, & \text{if } n = 1; \\ 6, & \text{if } n = 2; \\ 10, & \text{if } n = 3; \\ 36, & \text{if } n = 4; \\ 38, & \text{if } n = 5; \\ 70, & \text{if } n = 6; \\ 75, & \text{if } n = 7; \\ 2^{t+3} + 2n - 2, & \text{if } n \geq 8 \text{ and } t = \lceil \log_2(n-1) \rceil. \end{cases}$$

Proof. The resulting automaton encoding the condensation function of the Rudin-Shapiro sequence can be seen below in Figure 6.3.

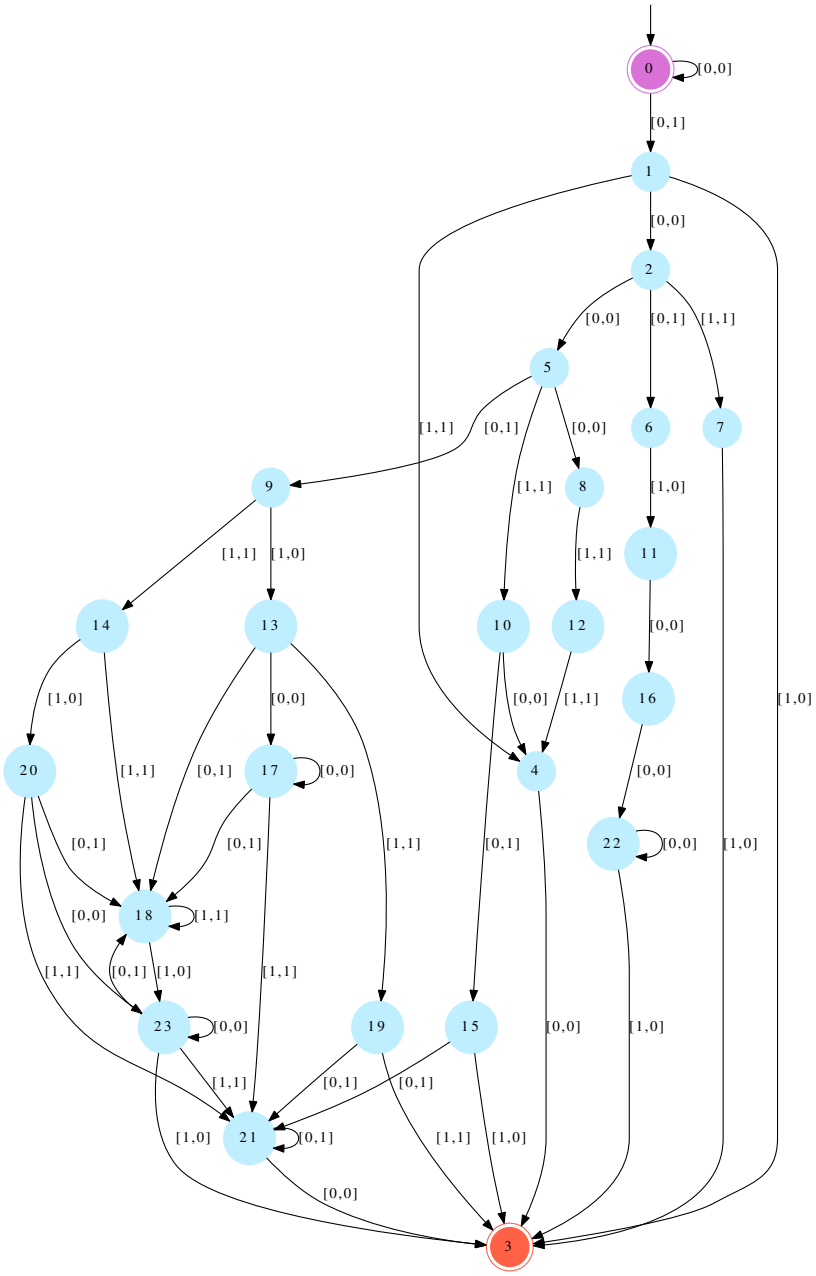


Figure 6.3: An automaton encoding $C_{\mathbf{r}}(n)$

Here is a summary of the computation:

```
          [>= i0 n] (3 => 2 states) in 0.019s
          [\out= l+i0 j+i0] (28 => 28 states) in 0.248s
        [\or ] (56 => 56 states) in 0.292s
      [\not ] (57 => 56 states) in 0.311s
     [\exists i0] (56 => 99 states) in 1.276s
    [\not ] (100 => 98 states) in 0.197s
    [>= m+k n+l] (12 => 4 states) in 0.022s
   [\and ] (273 => 273 states) in 0.336s
   [>= l k] (3 => 2 states) in 0.014s
  [\and ] (425 => 425 states) in 0.527s
 [\exists l] (425 => 1818 states) in 22.503s
[\not ] (1819 => 1818 states) in 2.560s
[\exists j] (1818 => 323 states) in 2.538s
[\not ] (324 => 323 states) in 0.189s
[\exists k] (323 => 23 states) in 0.043s
[\min m] (25 => 25 states) in 0.078s
[total] (25 states) in 31.743s
```

□

6.4 Paperfolding

Theorem 24. *For the regular paperfolding sequence, we have*

$$C_{\mathbf{p}}(n) = \begin{cases} 2, & \text{if } n = 1; \\ 5, & \text{if } n = 2; \\ 13, & \text{if } n = 3; \\ 15, & \text{if } n = 4; \\ 29, & \text{if } n = 5; \\ 34, & \text{if } n = 6; \\ 3 \cdot 2^t + 2n - 1 & \text{if } n \geq 7 \text{ and } t = \lceil \log_2 n \rceil. \end{cases}$$

Proof. The resulting automaton encoding the condensation function of the paperfolding sequence can be seen below in Figure 6.4.

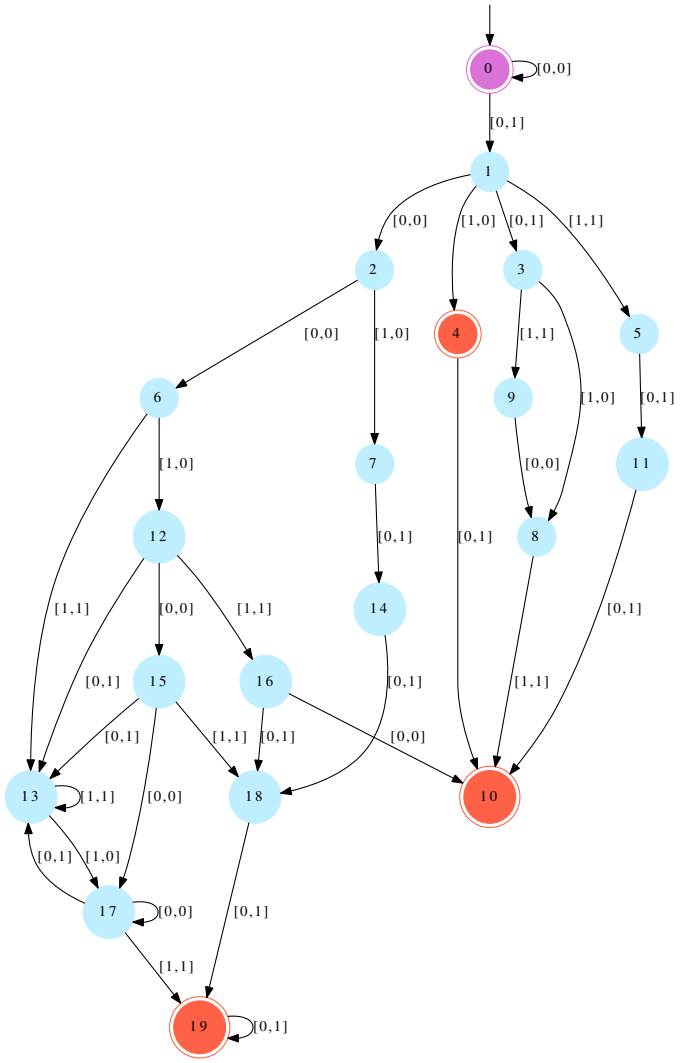


Figure 6.4: An automaton encoding $C_{\mathbf{p}}(n)$

Here is a summary of the computation:

```
  [>= i0 n] (3 => 2 states) in 0.016s
  [\out= l+i0 j+i0] (33 => 15 states) in 0.231s
 [\or ] (31 => 31 states) in 0.148s
[\not ] (32 => 30 states) in 0.145s
```

```
        [\exists i0] (30 => 61 states) in 0.221s
       [\not ] (62 => 60 states) in 0.117s
       [>= m+k n+1] (12 => 4 states) in 0.023s
      [\and ] (157 => 157 states) in 0.172s
      [>= 1 k] (3 => 2 states) in 0.013s
     [\and ] (241 => 241 states) in 0.266s
    [\exists l] (241 => 433 states) in 1.722s
   [\not ] (434 => 433 states) in 0.560s
  [\exists j] (433 => 155 states) in 0.381s
 [\not ] (156 => 155 states) in 0.094s
[\exists k] (155 => 21 states) in 0.023s
[\min m] (19 => 19 states) in 0.065s
[total] (19 states) in 4.373s
```

□

6.5 Period-doubling

Theorem 25. *For the period-doubling sequence we have*

$$C_{\mathbf{d}}(n) = \begin{cases} 2, & \text{if } n = 1; \\ 2^t + 2n - 1, & \text{if } n \geq 2 \text{ and } t = \lceil \log_2 n \rceil - 1. \end{cases}$$

Proof. The resulting automaton encoding the condensation function of the period-doubling sequence can be seen below in Figure 6.5.

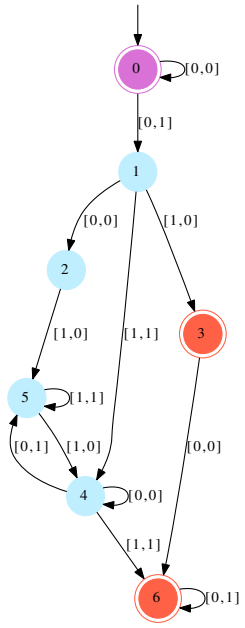


Figure 6.5: An automaton encoding $C_{\mathbf{d}}(n)$

Here is a summary of the computation:

```
          [>= i0 n] (3 => 2 states) in 0.020s
          [\out= l+i0 j+i0] (13 => 6 states) in 0.150s
        [\or ] (13 => 13 states) in 0.074s
       [\not ] (14 => 12 states) in 0.069s
      [\exists i0] (12 => 11 states) in 0.027s
     [\not ] (12 => 10 states) in 0.036s
     [>= m+k n+l] (12 => 4 states) in 0.022s
    [\and ] (30 => 30 states) in 0.051s
    [>= l k] (3 => 2 states) in 0.013s
   [\and ] (50 => 50 states) in 0.076s
  [\exists l] (50 => 73 states) in 0.055s
 [\not ] (74 => 73 states) in 0.087s
[\exists j] (73 => 66 states) in 0.038s
[\not ] (67 => 66 states) in 0.047s
[\exists k] (66 => 9 states) in 0.014s
[\min m] (9 => 9 states) in 0.053s
[total] (9 states) in 0.883s
```

□

6.6 Baum-Sweet

Theorem 26. *For the Baum-Sweet sequence, we have*

$$C_{\mathbf{b}}(n) = \begin{cases} 2, & \text{if } n = 1; \\ 5, & \text{if } n = 2; \\ 16, & \text{if } n = 3; \\ 26, & \text{if } n = 4; \\ 27, & \text{if } n = 5; \\ 46, & \text{if } n = 6; \\ 49, & \text{if } n = 7; \\ 50, & \text{if } n = 8; \\ 89, & \text{if } n = 9; \\ 98, & \text{if } n = 10; \\ 99, & \text{if } n = 11; \\ 100, & \text{if } n = 12; \\ 23 \cdot 2^t + n - 1, & \text{if } n \geq 13,\ 2^t = (n-8) \text{ and } t \text{ is odd}; \\ 23 \cdot 2^{t-1} + n - 1, & \text{otherwise, where } t = \lceil \log_2(n-8) \rceil\ . \end{cases}$$

Proof. The resulting automaton encoding the condensation function of the Baum-Sweet sequence can be seen below in Figure 6.6.

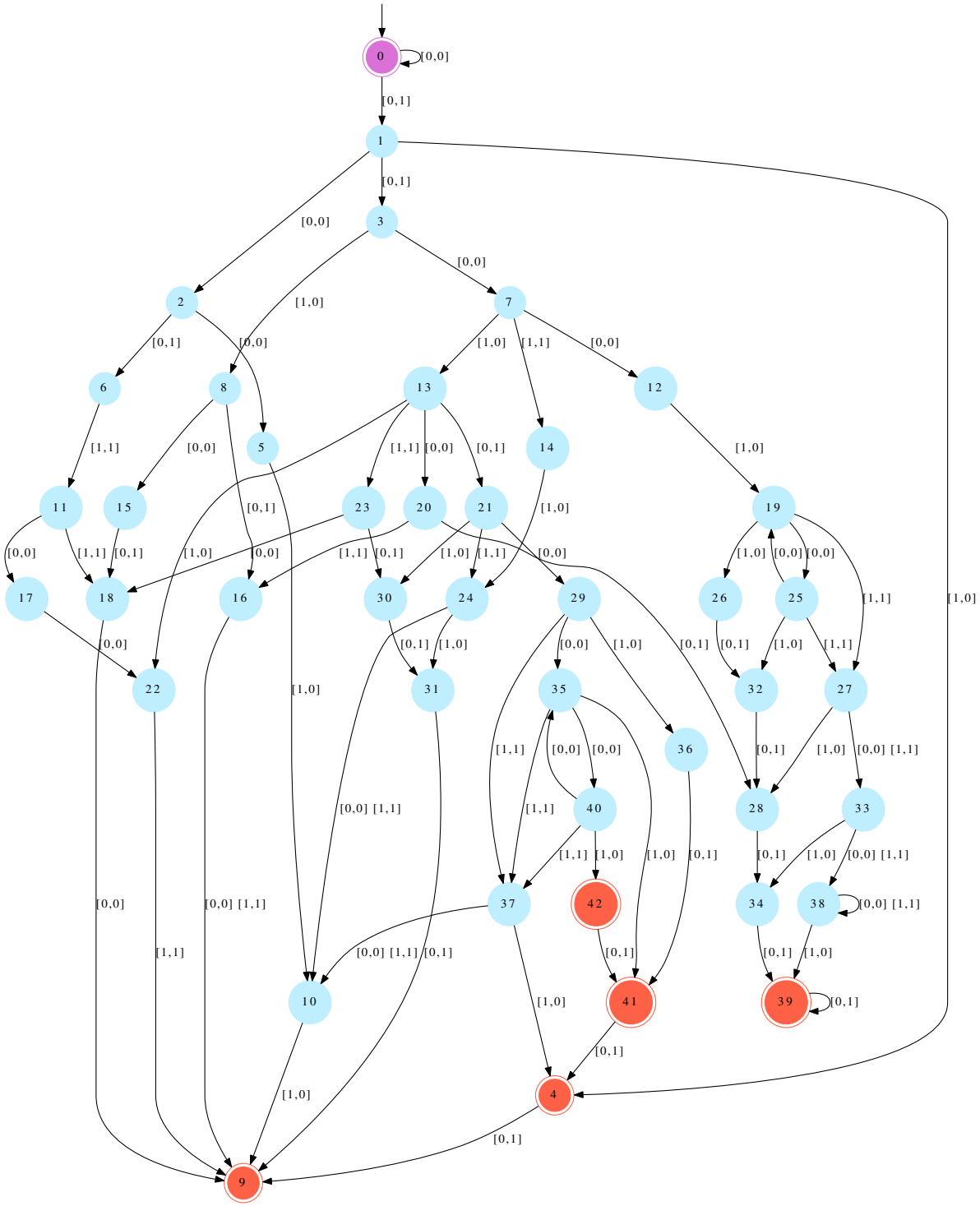


Figure 6.6: An automaton encoding $C_{\mathbf{b}}(n)$

Here is a summary of the computation:

```
           [>= i0 n] (3 => 2 states) in 0.019s
           [\out= l+i0 j+i0] (36 => 25 states) in 0.247s
          [\or ] (49 => 49 states) in 0.258s
         [\not ] (50 => 48 states) in 0.261s
        [\exists i0] (48 => 301 states) in 4.733s
       [\not ] (302 => 301 states) in 0.823s
       [>= m+k n+1] (12 => 4 states) in 0.026s
      [\and ] (879 => 879 states) in 2.554s
      [>= 1 k] (3 => 2 states) in 0.015s
     [\and ] (1377 => 1377 states) in 2.237s
    [\exists l] (1377 => 28442 states) in 9m50.703s
   [\not ] (28443 => 28442 states) in 19.963s
  [\exists j] (28442 => 156 states) in 56.565s
 [\not ] (157 => 155 states) in 0.104s
[\exists k] (155 => 71 states) in 0.038s
[\min m] (31 => 31 states) in 0.176s
[total] (31 states) in 11m25.642s
```

□

6.7 Mephisto Waltz

Theorem 27. *For the Mephisto waltz sequence we have*

$$C_{\mathbf{m}}(n) = \begin{cases} 2, & \text{if } n = 1; \\ 5, & \text{if } n = 2; \\ 7 \cdot 3^{t-1} + 2 \cdot n - 2, & \text{if } n \geq 3,\ t = \lceil \log_3(n-1) \rceil \text{ and } n-2 \text{ begins with a } 1; \\ 9 \cdot 3^{t-1} + 2 \cdot n - 2, & \text{otherwise, where } t = \lceil \log_3(n-1) \rceil. \end{cases}$$

Proof. The resulting automaton encoding the condensation function of the Mephisto waltz sequence can be seen below in Figure 6.7.

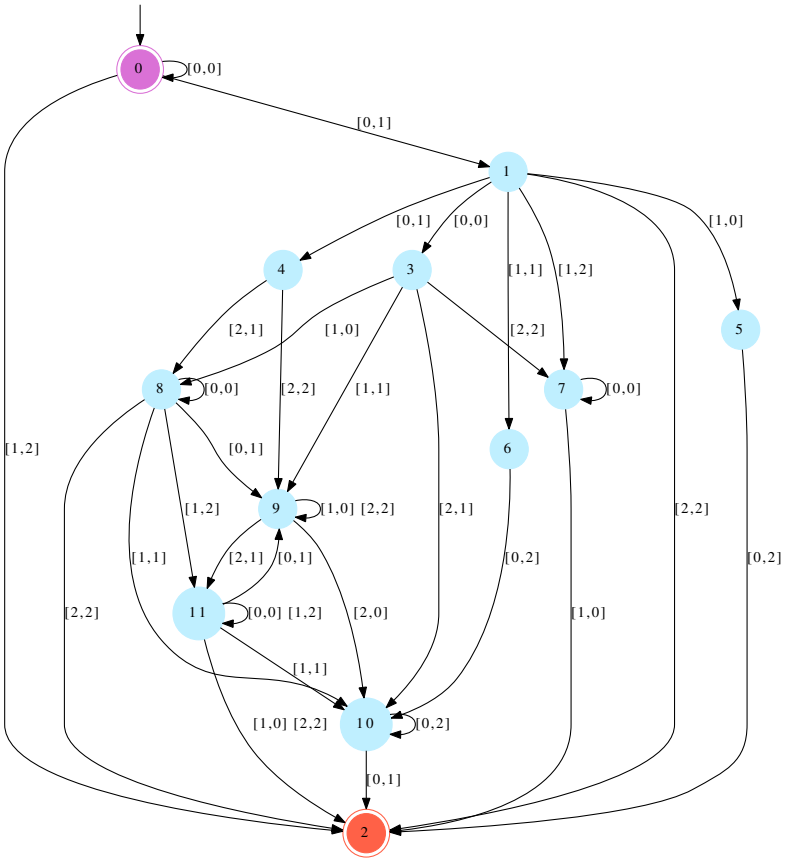


Figure 6.7: An automaton encoding $C_{\mathbf{m}}(n)$

Here is a summary of the computation:

```
        [>= i0 n] (3 => 2 states) in 0.099s
        [\out= l+i0 j+i0] (8 => 8 states) in 0.714s
       [\or ] (16 => 16 states) in 0.982s
      [\not ] (17 => 16 states) in 1.092s
     [\exists i0] (16 => 20 states) in 0.429s
    [\not ] (21 => 19 states) in 0.295s
    [>= m+k n+1] (12 => 4 states) in 0.115s
   [\and ] (56 => 56 states) in 0.477s
   [>= l k] (3 => 2 states) in 0.042s
  [\and ] (89 => 89 states) in 0.741s
 [\exists l] (89 => 226 states) in 1.223s
[\not ] (227 => 226 states) in 1.520s
```

```
   [\exists j] (226 => 81 states) in 0.359s
  [\not ] (82 => 81 states) in 0.154s
 [\exists k] (81 => 15 states) in 0.028s
[\min m] (12 => 12 states) in 0.091s
[total] (12 states) in 8.967s
```

□

6.8 Stewart Choral

Theorem 28. *For the Stewart choral sequence we have*

$$C_{\mathbf{s}}(n) = \begin{cases} 2, & \text{if } n = 1; \\ 2 \cdot 3^{t-1} + 2 \cdot n - 1, & \text{if } n \geq 2,\ t = \lceil \log_3(n) \rceil \text{ and } n-1 \text{ begins with a } 1; \\ 3 \cdot 3^{t-1} + 2 \cdot n - 1, & \text{otherwise, where } t = \lceil \log_3(n) \rceil. \end{cases}$$

Proof. The resulting automaton encoding the condensation function of the Stewart choral sequence can be seen below in Figure 6.8.

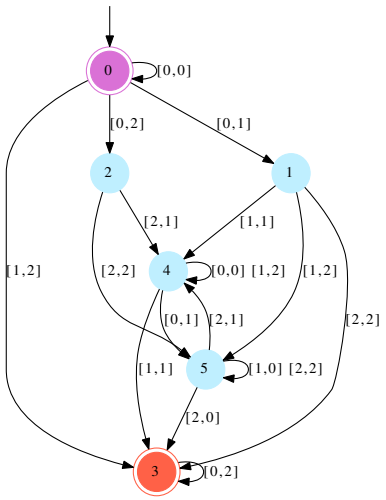


Figure 6.8: An automaton encoding $C_{\mathbf{s}}(n)$

Here is a summary of the computation:

```
            [>= i0 n] (3 => 2 states) in 0.160s
            [\out= l+i0 j+i0] (22 => 11 states) in 2.104s
          [\or ] (23 => 23 states) in 2.275s
         [\not ] (24 => 22 states) in 2.124s
        [\exists i0] (22 => 14 states) in 0.675s
       [\not ] (15 => 13 states) in 0.335s
       [>= m+k n+l] (12 => 4 states) in 0.174s
      [\and ] (46 => 46 states) in 0.657s
      [>= l k] (3 => 2 states) in 0.081s
     [\and ] (72 => 72 states) in 0.998s
    [\exists l] (72 => 110 states) in 1.142s
   [\not ] (111 => 110 states) in 1.085s
  [\exists j] (110 => 64 states) in 0.344s
 [\not ] (65 => 64 states) in 0.205s
[\exists k] (64 => 8 states) in 0.056s
[\min m] (7 => 7 states) in 0.151s
[total] (7 states) in 13.412s
```

□

Chapter 7

Power Avoidance

Remark 29. Sections 7.3 and 7.4 of this chapter are also taken verbatim from [56].

7.1 Introduction

In 1912 Axel Thue proved that the Thue-Morse sequence is overlap-free [104]. It is widely believed that this celebrated result launched the study of combinatorics on words itself.

A word w over an alphabet Σ is said to be an *overlap* if it is of the form $ayaya$ where $a \in \Sigma$ is a letter and $y \in \Sigma^*$ is a word. For example, the English word `alfalfa` is an overlap. If none of the factors of a sequence are an overlap the sequence is said to be *overlap-free*.

An overlap can be thought of as two repetitions of the string ay followed by the first letter of the string: a. Informally, such repetitions of a base word are called 'powers'. For example, if $w = xx$ then w is called *square*. Thus, the English words `tartar` and `cancan` are squares. Similarly, w of the form $w = xxx$ is called a *cube*. In this chapter we are concerned with avoiding powers in automatic sequences.

More generally, a word w is called an α *power* if

$$w = x^{\alpha} = x \cdot x \cdots x \cdot x'$$

where $\alpha = \frac{|w|}{|x|} > 1$ and x' is a prefix of x. We say that $|x|$ is the *period* of w and α is its *exponent*; we will say more on this in Chapter 8. The exponent need not be an integer.

For example, the English word

$$\underbrace{\texttt{entanglem}}_{9}\texttt{ent}$$

is a $\frac{4}{3}$ power where the factor `entanglem` is repeated $\frac{12}{9} = \frac{4}{3}$ times.

We let α^+ *powers* denote the set of all β powers where $\beta > \alpha$. For example an overlap $w = ayaya$ where the length of y is $|y| = 5$ is a $\frac{13}{6}$ power. Since $\frac{13}{6} > 2$ it is a 2^+ power. In fact, all overlaps are 2^+ powers. Conversely, we let α^- *powers* denote the set of all β powers where $\beta < \alpha$.

Words and sequences can also avoid powers. A sequence is said to be α *power free* if none of its factors are an α power. The Thue-Morse sequence is known to be overlap free, thus is is 2^+ power free.

In Sections 7.3 and 7.4 we restate the results of [56] regarding the paperfolding and Leech sequences. In Section 7.2 we confirm Thue's famous result that the Thue-Morse word is overlap free. Section 7.8 confirms a recent result of Rampersad's and the remaining sections in this chapter present results regarding other k-automatic sequences novel to this thesis.

7.2 Thue-Morse

First, we re-prove Thue's result using our automated method.

Theorem 30. *The Thue-Morse sequence is overlap-free, or in other words, does not contain any 2^+ powers.*

Proof. The following predicate accepts all indexes i at which an overlap occurs:

$$O_{\mathbf{t}}(i) = \{i \ : \ \exists\, n > 0 \text{ and } \mathbf{t}[i..i+n] = \mathbf{t}[i+n..i+2n+1]\}.$$

This is expressible and below, in Figure 7.1, is the resulting automaton.

Figure 7.1: An automaton encoding the location of overlaps in the Thue-Morse sequence

Here is a summary of the computation:

```
[> n 0] (3 => 2 states) in 0.018s
    [>= i0 n+1] (6 => 2 states) in 0.023s
    [\out= i+i0 i+n+i0] (8 => 8 states) in 0.049s
   [\or ] (16 => 16 states) in 0.049s
  [\not ] (17 => 16 states) in 0.056s
 [\exists i0] (16 => 2 states) in 0.030s
[\not ] (3 => 1 states) in 0.051s
[\and ] (1 => 1 states) in 0.036s
[\exists n] (1 => 1 states) in 0.029s
[total] (1 states) in 0.370s
```

□

In order to confirm that 2^+ is the lowest power avoided we give, in Figure 7.2, the automaton accepting all indexes i at which a square occurs.

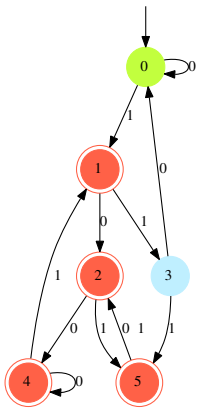


Figure 7.2: An automaton encoding the location of squares in the Thue-Morse sequence

Here is a summary of the computation:

```
[> n 0] (3 => 2 states) in 0.008s
    [>= i0 n+1] (6 => 2 states) in 0.010s
    [\out= i+i0 i+n+i0] (8 => 8 states) in 0.024s
   [\or ] (16 => 16 states) in 0.023s
  [\not ] (17 => 16 states) in 0.023s
 [\exists i0] (16 => 2 states) in 0.010s
```

```
  [\not ] (3 => 1 states) in 0.016s
 [\and ] (1 => 1 states) in 0.016s
[\exists n] (1 => 1 states) in 0.008s
[total] (1 states) in 0.148s
```

7.3 Paperfolding

In 1979, Prodinger and Urbanek [83] characterized the squares in the regular paperfolding sequence, using a case analysis. We verified this by creating an automaton to accept the language

$$\{(n)_2 \ : \ \exists \ i \ \mathbf{p}[i..i+n-1] = \mathbf{p}[i+n..i+2n-1]\}.$$

The resulting automaton (most significant-digit first) is depicted below, from which we recover the Prodinger-Urbanek result that the only squares xx in $\mathbf{x}$ have lengths $|x| = 1, 3,$ or 5.

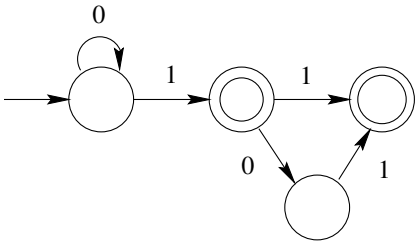


Figure 7.3: Lengths of squares in the regular paperfolding sequence

7.4 Leech

Next, we considered an old construction of Leech [69]. Using a case analysis, he showed that the the Leech sequence is squarefree. We used our method to verify this. In fact, we proved something more:

Theorem 31. *The Leech sequence*

$$\mathbf{l} = l_0 l_1 l_2 \cdots = 012102120121012021020120 21 \cdots$$

is $\frac{15}{8}^+$*-free, and this exponent is optimal. Furthermore, if* x *is a* $\frac{15}{8}$ *power occurring in* $\mathbf{l}$*, then* $|x| = 15 \cdot 13^i$ *for some* $i \geq 0$.

Proof. We used our method to verify that there are no powers $> \frac{15}{8}$. The largest intermediate automaton constructed had 360 states and the total computation time was 1140 seconds. The exponent is optimal because, for example, the factor $\mathbf{l}[25..39] = 120102101201021$ is easily seen to be a $\frac{15}{8}$ power.

We also computed a 4-state automaton that accepts the base-13 expansion of those pairs (i, n) for which a $\frac{15}{8}$ power of length n begins at position i. We give the automaton in Figure 7.4 below.

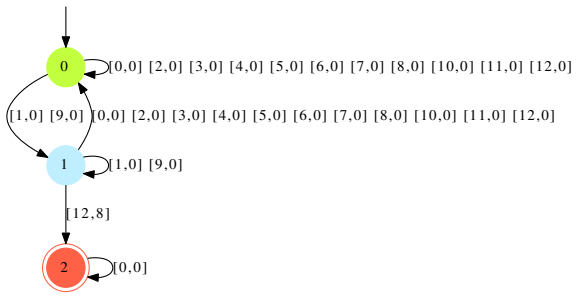


Figure 7.4: An automaton encoding the location of $(\frac{15}{8})$ powers in the Leech sequence

The largest intermediate automaton had 360 states and the total computation time was 811 seconds. The set of all accepting paths can be represented as $A^*\{[1,1],[9,1]\}[12,2][0,0]^*$, where $A = \bigcup_{0 \le j < 13}[j, 0]$, which corresponds to lengths of the form $15 \cdot 13^i$.

□

7.5 Rudin-Shapiro

Theorem 32. *The Rudin-Shapiro sequence avoids 4^+ powers but not 4 powers.*

Proof. The following predicate accepts all indexes i at which a 4^+ power occurs:

$$P_{4^+,\mathbf{r}}(i) = \{i \ : \ \exists\ n > 0 \text{ and } \mathbf{r}[i..i+3n] = \mathbf{r}[i+n..i+4n+1]\}.$$

This is expressible and below, in Figure 7.5, is the resulting automaton.

Figure 7.5: An automaton encoding the location of 4^+ powers in the Rudin-Shapiro sequence

Here is a summary of the computation:

```
  [> n 0] (3 => 2 states) in 0.024s
      [>= i0 3*n+1] (9 => 4 states) in 0.025s
      [\out= i+i0 i+n+i0] (24 => 24 states) in 0.097s
     [\or ] (76 => 76 states) in 0.098s
    [\not ] (77 => 76 states) in 0.093s
   [\exists i0] (76 => 2 states) in 0.042s
  [\not ] (3 => 1 states) in 0.044s
 [\and ] (1 => 1 states) in 0.042s
[\exists n] (1 => 1 states) in 0.015s
[total] (1 states) in 0.508s
```

In order to confirm that 4^+ is the lowest power avoided, in Figure7.6 is the automaton accepting all indexes i at which a 4 power occurs.

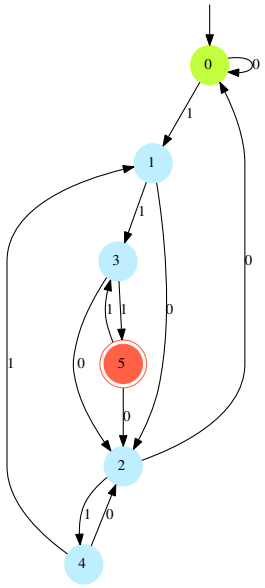


Figure 7.6: An automaton encoding the location of 4 powers in the Rudin-Shapiro sequence

Here is a summary of the computation:

```
   [> n 0] (3 => 2 states) in 0.022s
       [>= i0 3*n+1-1] (9 => 4 states) in 0.024s
       [\out= i+i0 i+n+i0] (24 => 24 states) in 0.108s
     [\or ] (76 => 76 states) in 0.099s
    [\not ] (77 => 76 states) in 0.092s
   [\exists i0] (76 => 9 states) in 0.045s
  [\not ] (10 => 8 states) in 0.049s
 [\and ] (8 => 8 states) in 0.039s
[\exists n] (8 => 8 states) in 0.019s
[total] (8 states) in 0.531s
```

□

7.6 Period-doubling

Theorem 33. *The period-doubling sequence avoids* 4 *powers but not* 4^- *powers.*

Proof. The following predicate accepts all indexes i at which a 4 power occurs:

$$P_{4,\mathbf{d}}(i) = \{i \ : \ \exists\ n > 0 \text{ and } \mathbf{d}[i..i+3n-1] = \mathbf{d}[i+n..i+4n]\}.$$

This is expressible and below, in Figure 7.7, is the resulting automaton.

Figure 7.7: An automaton encoding the location of 4 powers in the period-doubling sequence

Here is a summary of the computation:

```
   [> n 0] (3 => 2 states) in 0.017s
      [>= i0 3*n] (9 => 4 states) in 0.023s
      [\out= i+i0 i+n+i0] (10 => 8 states) in 0.071s
     [\or ] (30 => 30 states) in 0.065s
    [\not ] (31 => 29 states) in 0.068s
   [\exists i0] (29 => 2 states) in 0.026s
  [\not ] (3 => 1 states) in 0.033s
 [\and ] (1 => 1 states) in 0.033s
[\exists n] (1 => 1 states) in 0.017s
[total] (1 states) in 0.384s
```

In order to confirm that 4 is the lowest power avoided, we also compute the location of 4^- powers. In Figure 7.8 we give the automaton accepting pairs (i, n) where a 4^- power of period n occurs at position i. Furthermore, we can see that arbitrarily large 4^- power occur at arbitrarily large indexes in this sequence.

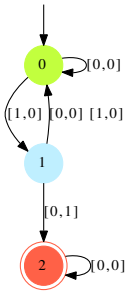


Figure 7.8: An automaton encoding the location and size of 4− powers in the period-doubling sequence

Here is a summary of the computation:

```
[> n 0] (3 => 2 states) in 0.023s
    [>= i0 3*n-1] (12 => 6 states) in 0.026s
    [\out= i+i0 i+n+i0] (10 => 8 states) in 0.074s
   [\or ] (31 => 31 states) in 0.070s
  [\not ] (32 => 30 states) in 0.056s
 [\exists i0] (30 => 5 states) in 0.025s
[\not ] (6 => 4 states) in 0.044s
[\and ] (4 => 4 states) in 0.034s
[total] (4 states) in 0.377s
```

□

7.7 Stewart Choral

Recently, Samsonov and Shur showed that the Stewart choral sequences is cube-free and that it avoids the pattern $xxyyxx$ [76]. Here, we confirm these results using our methods.

Theorem 34. *The Stewart choral sequence avoids cubes but not* 3^- *powers arbitrarily close to* 3.

Proof. The following predicate accepts all indexes i at which a cube occurs:

$$P_{3,\mathbf{s}}(i) = \{i \ : \ \exists n > 0 \text{ such that } \mathbf{s}[i..i+2n-1] = \mathbf{s}[i+n..i+3n-1]\}$$

This is also expressible in our logical theory and below, in Figure 7.9, is the resulting automaton.

Figure 7.9: An automaton encoding the location of cubes in the Stewart choral sequence

Here is a summary of the computation:

```
  [> n 0] (3 => 2 states) in 0.020s
      [>= i0 2*n] (6 => 3 states) in 0.030s
      [\out= i+i0 i+n+i0] (19 => 13 states) in 0.116s
     [\or ] (32 => 32 states) in 0.126s
    [\not ] (33 => 31 states) in 0.116s
   [\exists i0] (31 => 2 states) in 0.030s
  [\not ] (3 => 1 states) in 0.045s
 [\and ] (1 => 1 states) in 0.040s
[\exists n] (1 => 1 states) in 0.019s
[total] (1 states) in 0.576s
```

In order to confirm that 3^- power are not avoided, we computed the location of $\frac{7}{4}$ powers in the Stewart choral sequence. In Figure 7.10 we give the automaton accepting pairs (i, n) where a repetition of length $3n-1$ of period n occurs at position i. Furthermore, we can see that powers arbitrarily close to 3 occur in this sequence.

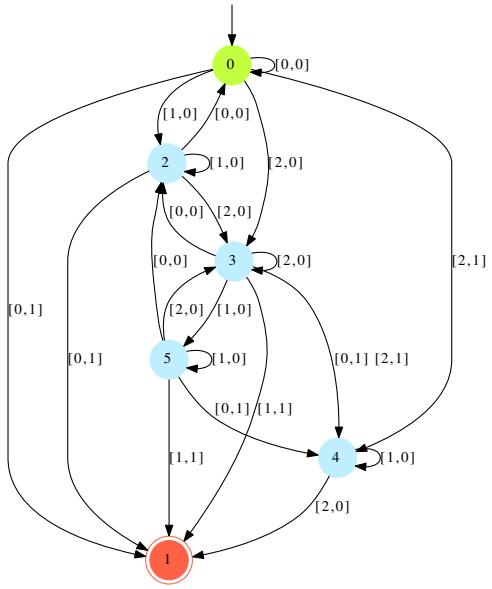


Figure 7.10: An automaton encoding the location and size of 3^- powers in the Stewart choral sequence

Here is a summary of the computation:

```
 [> n 0] (3 => 2 states) in 0.034s
     [>= i0 2*n-1] (9 => 5 states) in 0.039s
     [\out= i+i0 i+n+i0] (19 => 13 states) in 0.121s
    [\or ] (34 => 34 states) in 0.120s
   [\not ] (35 => 33 states) in 0.116s
  [\exists i0] (33 => 7 states) in 0.036s
 [\not ] (8 => 6 states) in 0.041s
[\and ] (6 => 6 states) in 0.034s
[total] (6 states) in 0.580s
```

□

Theorem 35. *The Stewart choral sequence avoids patterns of the form xxyyxx.*

Proof. The following predicate accepts all pairs $(|x|, |y|)$ such that pattern of the form

$xxyyxx$ occurs somewhere in the Stewart choral sequence:

$$P_{xxyyxx,\mathbf{s}}(n,m) = \{(n,m) \ : \ \exists\ i \text{ such that } n,m>0$$
$$\mathbf{s}[i..i+n-1] = \mathbf{s}[i+n..i+2n-1]$$
$$\mathbf{s}[i+2n..i+2n+m-1] = \mathbf{s}[i+2n+m..i+2n+2m-1]$$
$$\mathbf{s}[i..i+2n-1] = \mathbf{s}[i+2n+2m..i+4n+2m-1]\}.$$

This is expressible in our logical theory and below, in Figure 7.11, is the resulting automaton.

Figure 7.11: An automaton encoding the occurrences of $xxyyxx$ in the Stewart choral sequence

Here is a summary of the computation:

```
      [>= i0 n] (3 => 2 states) in 0.034s
      [\out= i+i0 i+n+i0] (19 => 13 states) in 0.279s
    [\or ] (25 => 25 states) in 0.216s
   [\not ] (26 => 24 states) in 0.199s
  [\exists i0] (24 => 8 states) in 0.068s
 [\not ] (9 => 7 states) in 0.054s
      [>= i0 m] (3 => 2 states) in 0.039s
      [\out= i+2*n+i0 i+2*n+m+i0] (35 => 23 states) in 0.892s
    [\or ] (43 => 43 states) in 0.369s
   [\not ] (44 => 42 states) in 0.346s
  [\exists i0] (42 => 16 states) in 0.101s
 [\not ] (17 => 15 states) in 0.082s
[\and ] (24 => 24 states) in 0.050s
      [>= i0 2*n] (6 => 3 states) in 0.051s
      [\out= i+i0 i+2*n+2*m+i0] (42 => 21 states) in 0.581s
    [\or ] (56 => 56 states) in 0.498s
   [\not ] (57 => 55 states) in 0.446s
  [\exists i0] (55 => 22 states) in 0.246s
 [\not ] (23 => 21 states) in 0.087s
 [> n 0] (3 => 2 states) in 0.023s
[\and ] (21 => 21 states) in 0.070s
[\and ] (1 => 1 states) in 0.050s
[\exists i] (1 => 1 states) in 0.021s
[total] (1 states) in 5.031s
```

□

7.8 Kurosaki's Sequence

In 2008 in the paper [68] Kurosaki defined a ternary sequence that avoids squares. In Figure 7.12 we give the DFAO in MSD representation generating the sequence. Recently, Rampersad and Camungol [21] showed that this sequence is $(\frac{7}{4})^+$ power-free and we confirm this result below.

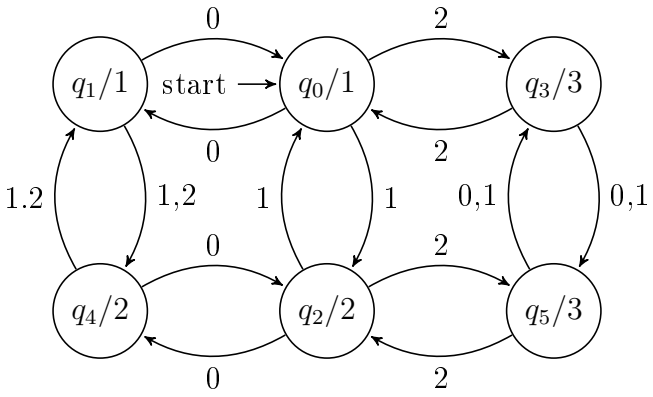


Figure 7.12: A finite automaton generating **k**: Kurosaki's sequence

Theorem 36. *The Kurosaki sequence avoids* $(\frac{7}{4})^+$ *powers but not* $(\frac{7}{4})$ *powers.*

Proof. The following predicate accepts all indexes i at which a $(\frac{7}{4})^+$ power occurs:

$$P_{\frac{7}{4},\mathbf{k}}(i) = \{i \ : \ \exists\, m, n > 0 \text{ such that } 4m > 7n \text{ and } \mathbf{k}[i..i+m-n-1] = \mathbf{k}[i+n..i+m]\}.$$

This is also expressible in our logical theory and below, in Figure 7.13, is the resulting automaton.

Figure 7.13: An automaton encoding the location of $(\frac{7}{4})^+$ powers in the Kurosaki sequence

Here is a summary of the computation:

```
    [> n 0] (3 => 2 states) in 0.024s
    [> 4*m 7*n] (84 => 11 states) in 0.115s
   [\and ] (16 => 16 states) in 0.086s
       [>= j m-n] (6 => 4 states) in 0.054s
       [\out= i+j i+n+j] (36 => 36 states) in 1.449s
     [\or ] (126 => 126 states) in 1.195s
    [\not ] (127 => 126 states) in 1.283s
   [\exists i0] (126 => 170 states) in 1.809s
  [\not ] (171 => 169 states) in 0.373s
 [\and ] (1 => 1 states) in 0.093s
[\exists m] (1 => 1 states) in 0.037s
[\exists n] (1 => 1 states) in 0.021s
[total] (1 states) in 6.884s
```

In order to confirm that $\frac{7}{4}$ power are not avoided, we computed the location of $\frac{7}{4}$ powers in the Kurosaki sequence. In Figure 7.14 we give the automaton accepting pairs (i, n) where a $\frac{7}{4}$ power of period n occurs at position i. Furthermore, we can see that arbitrarily large such power occur at arbitrarily large indexes in this sequence.

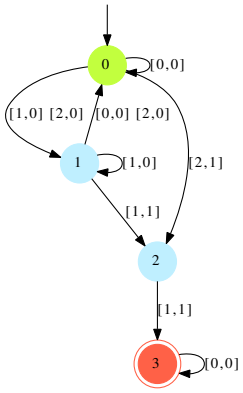


Figure 7.14: An automaton encoding the location and size of $\frac{7}{4}$ powers in the Kurosaki sequence

Here is a summary of the computation:

```
    [> n 0] (3 => 2 states) in 0.027s
    [= 4*m 7*n] (84 => 10 states) in 0.108s
  [\and ] (11 => 11 states) in 0.058s
      [>= j m-n] (6 => 4 states) in 0.053s
      [\out= i+j i+n+j] (36 => 36 states) in 1.450s
    [\or ] (126 => 126 states) in 1.189s
```

```
    [\not ] (127 => 126 states) in 1.260s
   [\exists i0] (126 => 170 states) in 1.811s
  [\not ] (171 => 169 states) in 0.376s
 [\and ] (5 => 5 states) in 0.065s
[\exists m] (5 => 5 states) in 0.028s
[total] (5 states) in 6.753s
```

□

Chapter 8

Least Periods and Quasi-periods

Remark 37. Sections 8.1, 8.2, 8.3, and 8.4 of this chapter are also taken essentially verbatim from [56].

8.1 Least Periods

In a recent paper, Currie and Saari [32] initiated the study of the least periods of infinite words. If $x = a_1 \cdots a_n$ is a finite word, then we say x has period $p \geq 1$ if $a_i = a_{i+p}$ for $1 \leq i \leq n - p$. Sometimes, abusing terminology, the word "period" is also used to refer to the word $a_1 a_2 \cdots a_p$ itself; there should be no confusion. For example, `alfalfa` has period 3 and `entanglement` has period 9.

Currie and Saari were interested in the set of all positive integers that can be the least period of some finite nonempty factor of $\mathbf{x}$. They explicitly computed the set of least periods for some famous infinite words, such as the Thue-Morse sequence. In particular, they proved that every positive integer can be the least period of the Thue-Morse sequence.

In this chapter, we prove that if $\mathbf{x}$ is k-automatic, then so is the characteristic sequence of the least periods of $\mathbf{x}$. Our method gives an explicit way to construct the automaton accepting the base-k representation of the least periods of $\mathbf{x}$. We then reprove the Currie-Saari result for Thue-Morse using a short finite computation, and we find similar results for three other classic sequences.

Theorem 38. *If $\mathbf{x}$ is a k-automatic sequence, then the characteristic sequence of least periods of $\mathbf{x}$ is (effectively) k-automatic.*

Proof. Again, using the method developed in [9, 28], it suffices to construct a predicate $L(n)$ that is true if n is a least period and false otherwise, using a logical language restricted to addition, subtraction, indexing into $\mathbf{x}$, comparisons, logical operations, and the existential and universal quantifiers.

It is easy to express the predicate P that n is a period of the factor $\mathbf{x}[i..j]$, as follows:

$$\begin{aligned} P(n,i,j) \quad \text{means} \quad & \mathbf{x}[i..j-n] = \mathbf{x}[i+n..j] \\ = \quad & \forall t \text{ with } i \le t \le j-n \text{ we have } \mathbf{x}[t] = \mathbf{x}[t+n]. \end{aligned}$$

Using this, we can express the predicate LP that n is the least period of $\mathbf{x}[i..j]$:

$$LP(n,i,j) = P(n,i,j) \text{ and } \forall n' < n \; \neg P(n',i,j).$$

Finally, we can express the predicate that n is a least period as follows

$$L(n) = \exists i,j \ge 0 \text{ with } 0 \le i+n \le j-1 \text{ such that } LP(n,i,j).$$

The construction is effective, and there is an algorithm that, given the automaton generating $\mathbf{x}$, will produce an automaton generating the characteristic sequence of least periods of $\mathbf{x}$. □

8.2 Enumeration

Given an infinite word $\mathbf{x}$ we can consider counting the number $Q_{\mathbf{x}}(n)$ of distinct factors (of arbitrary length) having least period of length n. In some cases this quantity can be infinite. For example, if an infinite word contains arbitrarily large factors of the form a^n, then $Q_{\mathbf{x}}(1) = \infty$.

We can also count the number $S_{\mathbf{x}}(n)$ of distinct factors of length n that are the least periods of some factor of $\mathbf{x}$.

We will show that both of these quantities are k-regular. A sequence $(f(n))_{n\ge 0}$ is k-regular if there exist vectors v, w and a matrix-valued morphism μ such that $f(n) = v\mu(x)w$, where x is the base-k representation of n. Actually, to be more precise, we will prove that $Q_{\mathbf{x}}(n)$ is $(\mathbb{N}_\infty, k)$-regular and that $S_{\mathbf{x}}(n)$ is $(\mathbb{N}, k)$-regular. This means that the entries in the vector and matrix are chosen from $\mathbb{N}_\infty = \mathbb{N} \bigcup \{\infty\}$ in the first case, and $\mathbb{N}$ in the second.

Theorem 39. *The sequence $Q_{\mathbf{x}}(n)$ is $(\mathbb{N}_\infty, k)$-regular, and the sequence $S_{\mathbf{x}}(n)$ is $(\mathbb{N}, k)$-regular.*

Proof. Using the ideas in [28], it suffices to show that the predicates

$$\begin{aligned} Q(n,i,l) &= \text{the factor } \mathbf{x}[i..i+l-1] \text{ has least period equal to } n \\ &\quad \text{and } \mathbf{x}[i..i+l-1] \neq \mathbf{x}[i'..i'+l-1] \text{ for all } i' < i \end{aligned}$$

and

$$\begin{aligned} P(n,i) &= \exists l \text{ such that the factor } \mathbf{x}[i..i+l-1] \text{ is of least period } n \\ &\quad \text{and } \mathbf{x}[i..i+n-1] \neq \mathbf{x}[i'..i'+n-1] \text{ for all } i' < i \end{aligned}$$

are expressible in our logical language. □

8.3 Computations

Currie and Saari [32, Thm. 2] proved

Theorem 40. *For each integer $n \geq 1$, the Thue-Morse word has a factor of least period n.*

Using our theorem solver, we were able to verify the result above using a short computation. (In contrast, Currie and Saari used four pages of rather intricate case reasoning.)

We also carried out the same computation for three other famous infinite words. Our results can be summarized as follows:

Theorem 41. *For each integer $n \geq 1$, the period-doubling sequence and the Rudin-Shapiro sequence have a factor of least period n.*

For the regular paperfolding sequence, the least periods are given by the integers whose base-2 representations are accepted by the automaton below. The least omitted least period is 18, and there are infinitely many. In the limit, exactly 57/64 of all integers are least periods of the regular paperfolding sequence.

Proof. The first results were obtained through our algorithm. A summary of our computations appears below:

Sequence name	Number of states in largest intermediate automaton	Number of states in final automaton	Seconds of CPU time
Thue-Morse	264	1	5.882
Rudin-Shapiro	1029	1	27.797
Period-doubling	89	1	4.327
Paper-folding	393	12	11.597

Table 8.1: Computation statistics of least periods of several automatic sequences

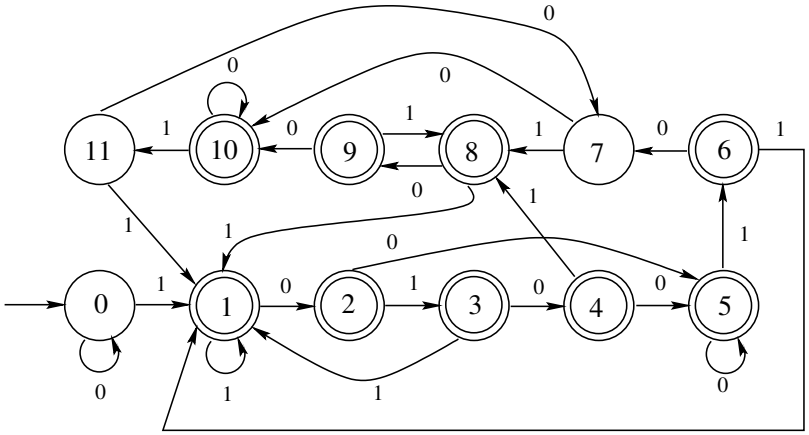


Figure 8.1: A finite automaton accepting least periods of the regular paperfolding sequence

For the result about the regular paperfolding sequence, we take the automaton computed by the algorithm (displayed in Figure 8.1) and compute the transition matrices M_a, $a \in \{0, 1\}$, containing a 1 in row i and column j if there is a transition on a from state i to state j. Then $(M^n)_{i,j}$, where $M := M_0 + M_1$, gives the number of words taking the automaton from state i to state j. A short computation gives that each row of $\lim_{n\to\infty} 2^{-n} M^n$ equals

$$\frac{1}{64}[0, 16, 8, 4, 2, 10, 5, 4, 4, 2, 6, 3].$$

All states except 7 and 11 are accepting, so the density of least periods is given by $(64 - 4 - 3)/64 = 57/64$, as claimed. □

8.4 More Enumeration

Kalle Saari suggested (personal communication) counting the number $L_{\mathbf{x}}(n)$ of distinct factors of $\mathbf{x}$ having a given least period of length n. Of course, this number could be infinite — for example, if $\mathbf{x}$ is ultimately periodic.

Theorem 42. *Let $\mathbf{x}$ be a k-automatic sequence. Then the number $L(n)$ of distinct factors having a given least period of length n is $(\mathbb{N}_\infty, k)$-regular.*

Proof. It suffices to show that the language

$$\{(n,i,l)_k \ : \ \text{there exists a factor of length } l \text{ beginning at position}$$

$$i \text{ with least period } n \text{ and this factor does not occur at any earlier position } \}$$

is regular. Then we apply the results of [28] to conclude that $L(n)$ is k-regular.

Above we created a predicate $LP(n,i,j)$ that asserts that n is the least period of $\mathbf{x}[i..j]$. So $LP(n,i,i+l-1)$ asserts that n is the least period of the factor of length l beginning at position i. So the predicate we want is

$$FOLP(n,i,j) := LP(n,i,i+l-1) \ \wedge \ \forall\, i' < i \ \mathbf{x}[i'..i'+l-1] \neq \mathbf{x}[i..i+l-1],$$

which adds the condition that $\mathbf{x}[i..i+l-1]$ is the first occurrence of that factor. Then $L(n)$ is just

$$|\{(n,i,j)_k \ : \ FOLP(n,i,j)\}|,$$

which is k-regular. □

As an example, we computed the linear representation of the k-regular sequence $L(n)$ for the Thue-Morse sequence. It is of dimension 52, and we omit it here. The first few values of $L(n)$ for Thue-Morse are given in the following table.

n	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$L(n)$	4	6	16	16	14	48	10	56	22	40	8	168	8	40	20	208

Table 8.2: The number $L(n)$ of distinct factors having a given least period of length n

The Thue-Morse sequence is known to be overlap-free, so any factor with least period n must be of length l, with $n \leq l \leq 2n$. Also, the Thue-Morse sequence, like all automatic

sequences, has linear subword complexity. Putting this together, we see that for the Thue-Morse sequence we have $L(n) = O(n^2)$.

On the other hand, let v, M_0, M_1, w be the vectors and matrices arising from the linear representation of L. We computed the characteristic polynomial of M_0, which turns out to be $P(x) := (x-4)(x-2)^3(x+1)^4(x-1)^7x^{37}$. Since every matrix satisfies its own characteristic polynomial, we know that the coefficients of M_0^n individually satisfy a linear recurrence of order 52 whose characteristic polynomial is P. Since $L(2^n) := (vM_1)M_0^n(w)$ is a linear combination of the entries of M_0^n, this means that $L(2^n)$ also satisfies the same linear recurrence. The roots of P are $0, 1, -1, 2, 4$, so this means that $L(2^n)$ can be written as an exponential polynomial in terms of powers of $0, 1, -1, 2, 4$. From our linear representation it easily follows that $L(n) = \frac{3}{4}n^2 + n$ for $n \geq 4$ that is a power of two; this matches the upper bound given above.

On the other hand, it is also easy to deduce from the matrix representation that $L(2^{2n+1} + 3) = 8$ for $n \geq 1$, so L does not grow quadratically in a uniform way.

8.5 Quasi-periods

A *quasi-period* of a sequence $\mathbf{x}$ is a word w that 'covers' any position in the sequence. For example, `010` is a quasi-period of `01001010` since it covers the word. Formally, w is the quasi-period of $\mathbf{x}$ if for all i there is some position j such that $w = \mathbf{x}[j..k]$ and $j \leq i \leq k$.

We call a sequence $\mathbf{x}$ *periodic* if it has a period, i.e., if there exists a $p > 0$ such that $\mathbf{x}[i] = \mathbf{x}[i+p]$ for all $i \geq 0$. Similarly, a sequence is called *quasi-periodic* if it has a quasi-period. This notion was originally introduced by Marcus in 2004 [74]. More recently Levé, Richomme and others have been interested in the properties of infinite quasi-periodic sequences and have published several papers on the subject [72, 71, 51]. Below, we summarize a few results regarding quasi-periodic k-automatic sequences.

Theorem 43. *The quasi-periodicity of a k-automatic sequence is decidable.*

Proof. Consider the predicate $QP_{\mathbf{x}}(i, n)$ accepting all pair (i, n) such that the factor $\mathbf{x}[i..i+n-1]$ is a quasi-period of $\mathbf{x}$. Then $QP_{\mathbf{x}}(i, n)$ can be expressed in our logical theory as

$$QP_{\mathbf{x}}(i,n) = \{(i,n) \ : \ \forall \ell \ \exists j \text{ such that } j \leq \ell, j+n > \ell \\ \text{and } \mathbf{x}[i..i+n-1] = \mathbf{x}[j..j+n-q].$$

□

Naturally, we consider whether any sequence in our list of 'usual suspects' is quasi-periodic. Below are our findings.

Theorem 44. *None of the following sequences is quasi-periodic: Thue-Morse, Rudin-Shapiro, regular paperfolding, period-doubling, Mephisto waltz, and the Stewart choral sequence.*

Proof. We have constructed the automaton accepting $QP_{\mathbf{x}}(i, n)$ for each sequence and confirmed that no pairs (i, n) are accepted. The program output can be seen in Appendix A.1. □

Furthermore, we performed a survey on the fixed points 4-uniform morphisms over Σ_2. We looked for periodic and quasi-periodic sequences. For each morphism, we first created the DFAO generating its fixed point, call it $\mathbf{x}$. Next, we computed $P_{\mathbf{x}}(n)$ a predicate that accepts n if and only if $\mathbf{x}$ is periodic with period n. If no such period existed, we computed $QP_{\mathbf{x}}(i, n)$. Out of 128 sequences, 23 were periodic and 3 were quasi-periodic without being periodic.

The full results are listed in Appendix A.2.

Chapter 9

Borders and Unbordered Factors

Remark 45. Sections of this chapter are taken verbatim from the papers [55] and [56].

9.1 Borders

A word w is *bordered* if it begins and ends with the same word x with $0 < |x| \leq |w|/2$; Otherwise it is *unbordered*. An example in English of a bordered word is `entanglement`. A bordered word is also called *bifix* in the literature, and unbordered words are also called *bifix-free* or *primary*.

Bordered and unbordered words have been actively studied in the literature, particularly with regard to the Ehrenfeucht-Silberger problem; see, for example, [42, 80, 38, 39, 60, 61, 30, 62, 84, 41], just to name a few.

Currie and Saari [32] studied the unbordered factors of the Thue-Morse sequence $\mathbf{t}$. They proved that if $n \not\equiv 1 \pmod 6$, then $\mathbf{t}$ has an unbordered factor of length n. (Also see [90], Lemma 4.10 and Problem 4.1.) However, this is not a necessary condition, as

$$\mathbf{t}[39..69] = \texttt{0011010010110100110010110100101},$$

which is an unbordered factor of length 31. Currie and Saari left it as an open problem to give a complete characterization of the integers n for which $\mathbf{t}$ has an unbordered factor of length n.

The following theorem and proof, quoted practically verbatim from [28], shows that, more generally, the characteristic sequence of n for which a given k-automatic sequence has an unbordered factor of length n, is itself k-automatic:

Theorem 46. *Let* $\mathbf{x} = a(0)a(1)a(2)\cdots$ *be a* k*-automatic sequence. Then the associated infinite sequence* $\mathbf{b} = b(0)b(1)b(2)\cdots$ *defined by*

$$b(n) = \begin{cases} 1, & \text{if } \mathbf{x} \text{ has an unbordered factor of length } n; \\ 0, & \text{otherwise;} \end{cases}$$

is k*-automatic.*

Proof. The sequence $\mathbf{x}$ has an unbordered factor of length n

iff

$\exists\ j \geq 0$ such that the factor of length n beginning at position j of $\mathbf{x}$ is unbordered

iff

there exists an integer $j \geq 0$ such that for all possible lengths l with $1 \leq l \leq n/2$, there is an integer i with $0 \leq i < l$ such that the supposed border of length l beginning and ending the factor of length n beginning at position j of $\mathbf{x}$ actually differs in the i'th position

iff

there exists an integer $j \geq 0$ such that for all integers l with $1 \leq l \leq n/2$ there exists an integer i with $0 \leq i < l$ such that $\mathbf{x}[j+i] \neq \mathbf{x}[j+n-l+i]$.

This predicate is expressible within our logical system. The characteristic sequence of these integers n is therefore k-automatic.

□

Since the proof is constructive, one can, in principle, carry out the construction to get an explicit description of the lengths for which the Thue-Morse sequence has an unbordered factor.

Doing so results in the following theorem:

Theorem 47. *There is an unbordered factor of length* n *in* $\mathbf{t}$ *if and only if the base-*2 *representation of* n *(starting with the most significant digit) is not of the form* $1(01^*0)^*10^*1$.

Proof. In the original paper [55] an alternate implementation by the second author was employed in the proof of this theorem. Here, we reprove the result using the implementation described in this thesis.

The resulting automaton can be seen below in Figure 9.1;

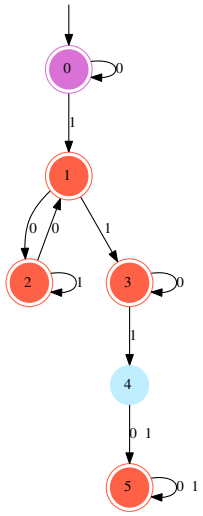


Figure 9.1: An automaton encoding the length of unbordered factors occurring in the Thue-Morse sequence

Here is a summary of the computation:

```
      [> 1 i] (3 => 2 states) in 0.010s
       [\out= j+i j+n-1+i] (12 => 12 states) in 0.039s
      [\not ] (13 => 12 states) in 0.026s
     [\and ] (20 => 20 states) in 0.035s
    [\exists i] (20 => 39 states) in 0.033s
    [> 2*1 n] (6 => 3 states) in 0.010s
   [\or ] (40 => 40 states) in 0.034s
   [> 1 1] (6 => 1 states) in 0.009s
  [\or ] (39 => 39 states) in 0.033s
 [\not ] (40 => 38 states) in 0.028s
[\exists 1] (38 => 23 states) in 0.012s
[\not ] (24 => 22 states) in 0.019s
[\exists j] (22 => 6 states) in 0.009s
[total] (6 states) in 0.318s
```

□

9.2 Additional Results on Unbordered Words

We also applied our decision procedure above to two other sequences: the Rudin-Shapiro sequence and the regular paperfolding sequence.

For a word $w \in 1(0+1)^*$, we define $a_w(n)$ to be the number of (possibly overlapping) occurrences of w in the (ordinary, unreversed) base-2 representation of n. Thus, for example, $a_{11}(7) = 2$.

Theorem 48. *The Rudin-Shapiro sequence has an unbordered factor of every length.*

Proof. We applied the same technique discussed previously for the Thue-Morse sequence. The resulting automaton accepted Σ^*.

Here is a summary of the computation:

```
        [> 1 i] (3 => 2 states) in 0.014s
         [\out= j+i j+n-1+i] (40 => 40 states) in 0.139s
        [\not ] (41 => 40 states) in 0.053s
       [\and ] (68 => 68 states) in 0.085s
      [\exists i] (68 => 128 states) in 0.765s
      [> 2*1 n] (6 => 3 states) in 0.010s
     [\or ] (136 => 136 states) in 0.084s
     [> 1 1] (6 => 1 states) in 0.009s
    [\or ] (135 => 135 states) in 0.079s
   [\not ] (136 => 134 states) in 0.061s
  [\exists 1] (134 => 99 states) in 0.060s
 [\not ] (100 => 98 states) in 0.035s
[\exists j] (98 => 1 states) in 0.010s
[total] (1 states) in 1.454s
```

□

Theorem 49. *The regular paperfolding sequence has an unbordered factor of length n if and only if the reversed representation $(n)_2$ is rejected by the automaton given in Figure 9.2.*

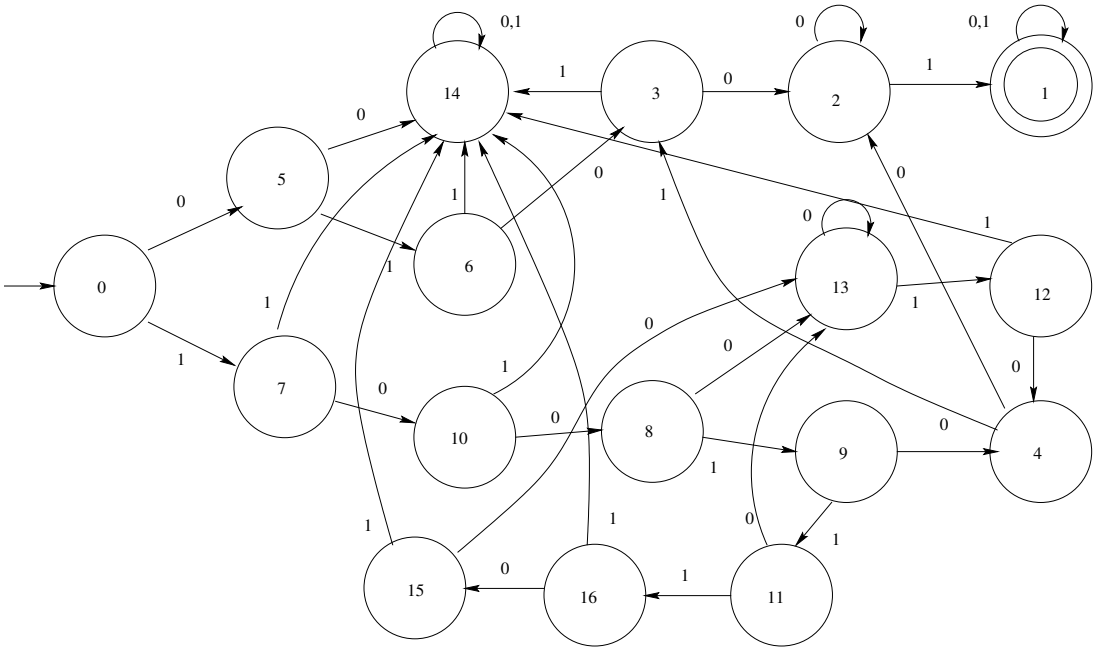


Figure 9.2: A finite automaton for unbordered factors in the regular paperfolding sequence

Proof. We applied the same technique discussed previously for the Thue-Morse sequence. Here is a summary of the computation:

```
        [> 1 i] (3 => 2 states) in 0.010s
         [\out= j+i j+n-1+i] (47 => 36 states) in 0.145s
        [\not ] (37 => 37 states) in 0.050s
       [\and ] (58 => 58 states) in 0.073s
      [\exists i] (58 => 63 states) in 0.124s
      [> 2*1 n] (6 => 3 states) in 0.010s
     [\or ] (75 => 75 states) in 0.050s
     [> 1 1] (6 => 1 states) in 0.009s
    [\or ] (75 => 75 states) in 0.055s
   [\not ] (76 => 74 states) in 0.040s
  [\exists 1] (74 => 61 states) in 0.028s
 [\not ] (62 => 60 states) in 0.027s
[\exists j] (60 => 16 states) in 0.010s
[total] (16 states) in 0.665s
```

□

9.3 Unbordered Factor Complexity

In Chapter 11 we study the subword complexity of k-automatic sequences and we show that it is k-synchronized. On the other hand, the unbordered factor complexity, i.e., the number of unbordered factors of k-automatic sequences is not synchronized in general. We prove this formally in Section 11.4. In the following sections, we show that it is at least k-regular.

In [28], the third author and co-authors made the following conjecture:

Conjecture 50. Let $f(n)$ denote the number of unbordered factors of length n in $\mathbf{t}$, the Thue-Morse sequence. Then f is given by $f(0) = 1$, $f(1) = 2$, $f(2) = 2$, and the system of recurrences

$$\begin{aligned}
f(4n+1) &= f(2n+1) \\
f(8n+2) &= f(2n+1) - 8f(4n) + f(4n+3) + 4f(8n) \\
f(8n+3) &= 2f(2n) - f(2n+1) + 5f(4n) + f(4n+2) - 3f(8n) \\
f(8n+4) &= -4f(4n) + 2f(4n+2) + 2f(8n) \\
f(8n+6) &= 2f(2n) - f(2n+1) + f(4n) + f(4n+2) + f(4n+3) - f(8n) \\
f(16n) &= -2f(4n) + 3f(8n) \\
f(16n+7) &= -2f(2n) + f(2n+1) - 5f(4n) + f(4n+2) + 3f(8n) \\
f(16n+8) &= -8f(4n) + 4f(4n+2) + 4f(8n) \\
f(16n+15) &= -8f(4n) + 2f(4n+3) + 4f(8n) + f(8n+7).
\end{aligned} \tag{9.1}$$

for $n \geq 0$.

Although this conjecture may appear unmotivated, it is characteristic of the kinds of recurrences that naturally appear for k-regular sequences, and was obtained by computing a large number of values of f and then looking for possible linear relations among subsequences of the form $(f(2^i n + j))_{n \geq 0}$.

This system suffices to calculate f efficiently, in $O(\log n)$ arithmetic steps.

In Section 9.4, we prove Conjecture 50. In Section 9.5, we discuss how to obtain relations like those above for a given k-regular sequence. In Section 9.6 we discuss the growth rate of f in detail. Finally, in Section 9.7, we give examples of other sequences with interesting numbers of unbordered factors.

9.4 Proof of the Conjecture

We now outline our computational proof of Conjecture 50.

Proof. Step 1: Using the ideas in [55], we created an automaton A of 23 states that accepts the language L of all words $(n, i)_2$ such that there is a "novel" unbordered factor of length n in $\mathbf{t}$ beginning at position i. Here "novel" means that this factor does not previously appear in any position to the left. Thus, the number of such words with first component equal to $(n)_2$ equals $f(n)$, the number of unbordered factors of $\mathbf{t}$ of length n. This automaton is illustrated below in Figure 9.3 (rotated to fit the figure more clearly).

Step 2: Using the ideas in [28], we now know that f is a 2-regular sequence, with a "linear representation" that can be deduced from the structure of A. This gives matrices M_0, M_1 of dimension 23 and vectors v, w such that $f(n) = vM_{a_1} \cdots M_{a_i} w$ where $a_1 \cdots a_i$ is the base-2 representation of n, written with the most significant digit first. They are given below.

$$M_0 = \left[\begin{array}{cccccccccccccccccccccc}
1&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&1&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&1&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&1&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&1&0&1&0&0&0&0&0&0&0&0&0&0\\
0&0&0&1&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&1&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&1&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&1&0&0\\
0&0&0&0&0&1&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&0\\
0&0&1&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&1&0&1&0&0&0&0&0&0&0&0&0&0\\
0&0&0&1&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&1&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&1&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&1&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&1&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&0&0
\end{array}\right]$$

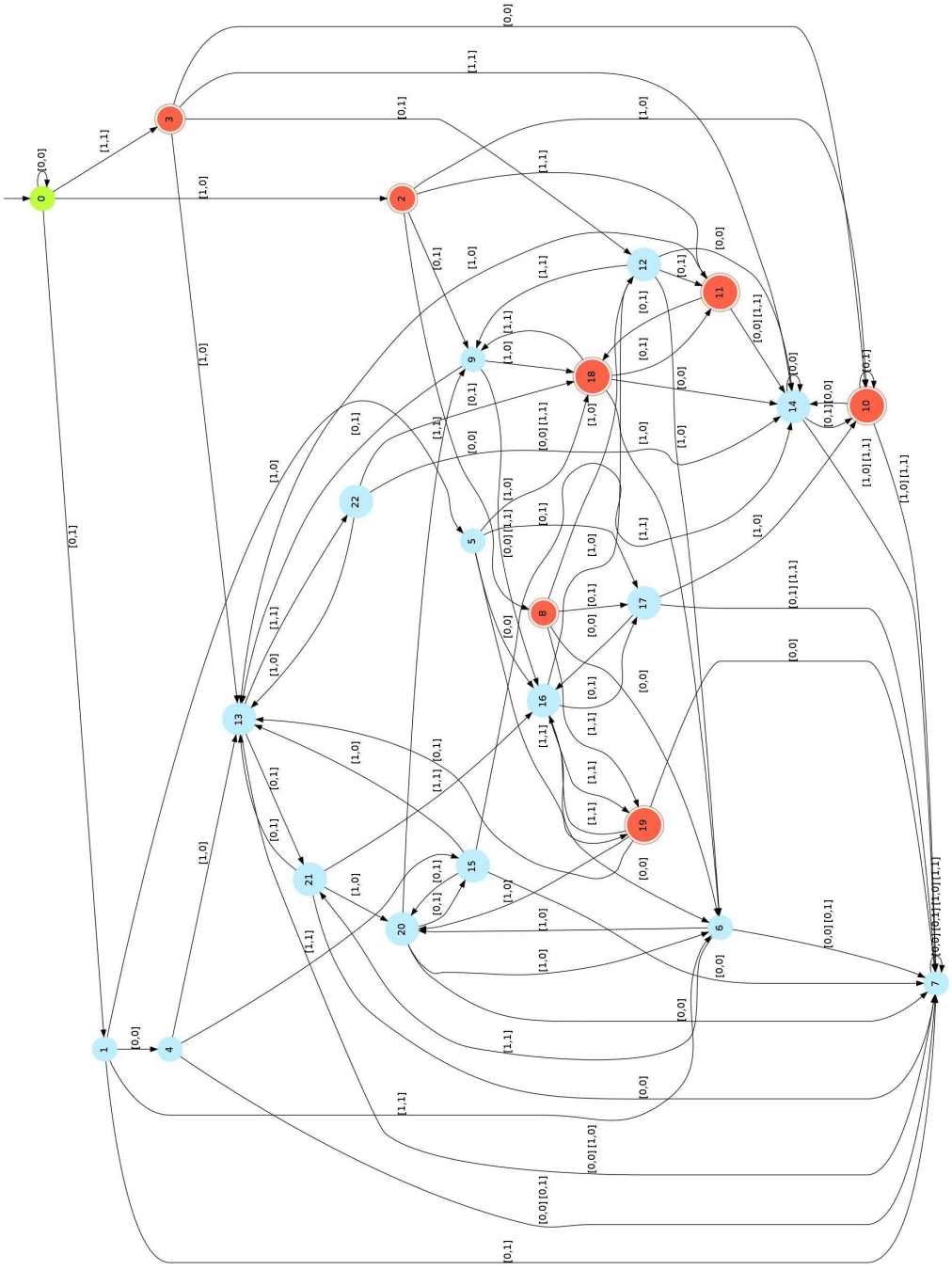


Figure 9.3: Automaton accepting $(n, i)_2$ such that there is a novel unbordered factor of length n at position i of $\mathbf{t}$

$$M_1 = \begin{bmatrix}
0&1&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&1&0&0\\
0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0\\
0&0&1&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&1&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&1&0&1&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&1&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&1&0&0&0&0&0\\
0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0\\
0&0&0&0&1&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&1&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&1&0&0&0\\
0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0
\end{bmatrix}.$$

$$\begin{aligned} v &= [1,0,0,0,0,0,0,0,1,0,0,0,1,0,0,0,0,0,0,0,0,1,0] \\ w &= [0,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]^T \end{aligned}$$

Step 3: Now each of the identities in (9.1) corresponds to a certain identity in matrices. For example, the identity $f(16n) = -2f(4n) + 3f(8n)$ can be written as

$$vMM_0M_0M_0M_0w = -2vMM_0M_0w + 3vMM_0M_0M_0w, \tag{9.2}$$

where M is the matrix product corresponding to the base-2 expansion of n. More generally, we can think of M as some arbitrary product of the matrices M_0 and M_1, starting with at least one M_1; this corresponds to an arbitrary $n \geq 1$. We can think of M as a matrix of indeterminates. Then (9.2) represents an assertion about the entries of M which can be verified. Of course, the entries of M are not completely arbitrary, since they come about as M_1 times some product of M_0 and M_1. We can compute the (positive) transitive closure of $M_0 + M_1$ and then multiply on the left by M_1; the entries that have 0's will be 0 in any product of M_1 times a product of the matrices M_0 and M_1. Thus we can replace the corresponding indeterminates by 0, which makes verifying (9.2) easier.

Another approach, which is even simpler, is to consider vM in place of M. This reduces the number of entries it is required to check from d^2 to d, where d is the dimension of the matrices.

Step 4: Finally, we have to verify the identities for $n = 0$ and $n = 1$, which is easy.

We carried out this computation in Maple for the matrices M_0 and M_1 corresponding to A, which completes the proof. The Maple program can be downloaded from
http://www.cs.uwaterloo.ca/~shallit/papers.html .

□

9.5 Determining the Relations

The verification method of the previous section can be extended to a method to mechanically *find* the relations for *any* given k-regular sequence g (instead of guessing them and verifying them), given the linear representation of g.

Suppose we are given the linear representation of a k-regular sequence g, that is, vectors v, w and matrices $M_0, M_1, \ldots, M_{k-1}$ such that

$$g(n) = vM_{a_1}M_{a_2}\cdots M_{a_j}w,$$

where $a_1a_2\cdots a_j = (n)_k$.

Now let M be arbitrary and consider vM as a vector with variable entries, say $[a_1, a_2, \ldots, a_d]$. Successively compute vMM_yw for words y of length $0, 1, 2, \ldots$ over $\Sigma_k = \{0, 1, \ldots, k-1\}$; this will give an expression in terms of the variables $a_1, \ldots, a_d$. After at most $d+1$ such relations, we find an expression for vMM_yw for some y as a linear combination of previously computed expressions. When this happens, you no longer need to consider any expression having y as a suffix. Eventually the procedure halts, and this corresponds to a system of equations like that in (50).

Consider the following example. Let $k = 2$, $v = [6, 1]$, $w = [2, 4]^T$, and

$$M_0 = \begin{bmatrix} -3 & 1 \\ 1 & 4 \end{bmatrix}$$
$$M_1 = \begin{bmatrix} 0 & 2 \\ -3 & 1 \end{bmatrix}$$

Suppose M is some product of M_0 and M_1, and suppose $vM = [a, b]$.

We find

$$\begin{aligned} vMw &= 2a + 4b \\ vMM_0w &= -2a + 18b \\ vMM_1w &= -8a - 2b \\ vMM_0M_0w &= 24a + 70b \\ vMM_1M_0w &= 36a + 24b \end{aligned}$$

and, solving the linear systems, we get

$$\begin{aligned} vMM_1w &= \frac{35}{11}vMw - \frac{9}{11}vM_0w \\ vMM_0M_0w &= 13vMw + vM_0w \\ vMM_1M_0w &= \frac{174}{11}vMw - \frac{24}{11}vM_0w. \end{aligned}$$

This gives us

$$\begin{aligned} g(2n+1) &= \frac{35}{11}g(n) + \frac{9}{11}g(2n) \\ g(4n) &= 13g(n) + g(2n) \\ g(4n+2) &= \frac{174}{11}g(n) - \frac{24}{11}g(2n) \end{aligned}$$

for $n \geq 1$.

9.6 The Growth Rate of $f(n)$

We now return to $f(n)$, the number of unbordered factors of $\mathbf{t}$ of length n. Here is a brief table of $f(n)$:

n	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
$f(n)$	1	2	2	4	2	4	6	0	4	4	4	4	12	0	4	4	8	4	8	0	8	4	4	8	24	0	4	4

Table 9.1: A first few terms of $f(n)$, the number of unbordered factors of $\mathbf{t}$ of length n

Kalle Saari (personal communication) asked about the growth rate of $f(n)$. The following results characterizes it.

Theorem 51. *We have $f(n) \leq n$ for $n \geq 4$. Furthermore, $f(n) = n$ infinitely often. Thus,* $\limsup_{n\geq 1} f(n)/n = 1$.

Proof. We start by verifying the following relations:

$$\begin{aligned}
f(4n) &= 2f(2n), \qquad (n \geq 2) && && (9.3)\\
f(4n+1) &= f(2n+1), \qquad (n \geq 0) && && (9.4)\\
f(8n+2) &= f(2n+1) + f(4n+3), \qquad (n \geq 1) && && (9.5)\\
f(8n+3) &= -f(2n+1) + f(4n+2) \qquad (n \geq 2) && && (9.6)\\
f(8n+6) &= -f(2n+1) + f(4n+2) + f(4n+3) \qquad (n \geq 2) && && (9.7)\\
f(8n+7) &= 2f(2n+1) + f(4n+3) \qquad (n \geq 3) && && (9.8)
\end{aligned}$$

These can be verified in exactly the same way that we verified Conjecture 50 earlier.

We now verify, by induction on n, that $f(n) \leq n$ for $n \geq 4$. The base case is $n = 4$, and $f(4) = 2$. Now assume $n \geq 5$. Otherwise,

- If $n \equiv 0 \pmod 4$, say $n = 4m$ and $m \geq 2$. Then $f(4m) = 2f(2m) \leq 2 \cdot 2m \leq 4m$ by (9.3) and induction.

- If $n \equiv 1 \pmod 4$, say $n = 4m + 1$ for $m \geq 1$, then $f(4m + 1) = f(2m + 1)$ by (9.4). But $f(2m + 1) \leq 2m + 1$ by induction for $m \geq 2$. The case $m = 1$ corresponds to $f(5) = 4 \leq 5$.

- If $n \equiv 2 \pmod 8$, say $n = 8m + 2$, then for $m \geq 2$ we have $f(8m + 2) = f(2m + 1) + f(4m + 3) \leq 6m + 4$ by induction, which is less than $8m + 2$. If $m = 1$, then $f(10) = 4 < 10$.

- If $n \equiv 3 \pmod 8$, say $n = 8m+3$ for $m \geq 1$, then $f(8m+3) = -f(2m+1)+f(4m+2) \leq f(4m + 2) \leq 4m + 2$ by induction.

- If $n \equiv 6 \pmod 8$, say $n = 8m + 6$, then $f(8m + 6) = -f(2m + 1) + f(4m + 2) + f(4m + 3) \leq f(4m + 2) + f(4m + 3) \leq 8m + 5$ by induction, provided $m \geq 2$. For $m = 0$ we have $f(6) = 6$ and for $m = 1$ we have $f(14) = 4$.

- If $n \equiv 7 \pmod 8$, say $n = 8m+7$, then $f(8m+7) = 2f(2m+1) + f(4m+3) \leq 2(2m+1) + 4m + 3 = 8m+5$ for $m \geq 3$, by induction. The cases $m = 0, 1, 2$ can be verified by inspection.

This completes the proof that $f(n) \leq n$.

It remains to see that $f(n) = n$ infinitely often. We do this by showing that $f(n) = n$ for n of the form $3 \cdot 2^i$, $i \geq 1$. Let us prove this by induction on i. It is true for $i = 1$ since $f(6) = 6$. Otherwise $i \geq 2$, and using (9.3) we have $f(3 \cdot 2^{i+1}) = 2f(3 \cdot 2^i) = 2 \cdot 3 \cdot 2^i = 3 \cdot 2^{i+1}$ by induction. This also implies the claim $\limsup_{n \geq 1} f(n)/n = 1$.

□

9.7 Unbordered Factors of Other Sequences

We can carry out similar computations for other famous sequences. In some cases the automata and the corresponding matrices are very large, which renders the computations time-consuming and the asymptotic behavior less transparent. We report on some of these computations, omitting the details.

Theorem 52. *Let* $\mathbf{r} = r_0 r_1 r_2 \cdots = 00010010 \cdots$ *denote the Rudin-Shapiro sequence, defined by* $r_n =$ *the number of occurrences, taken modulo* 2*, of '11' in the binary expansion of* n*. Let* $f_{\mathbf{r}}(n)$ *denote the number of unbordered factors of length* n *in* $\mathbf{r}$*. Then* $f_{\mathbf{r}}(n) \leq \frac{21}{8} n$ *for all* $n \geq 1$*. Furthermore if* $n = 2^i + 1$*, then* $f(n) = 21 \cdot 2^{i-3}$ *for* $i \geq 4$*.*

Theorem 53. *For the period-doubling sequences we have that* $f_{\mathbf{d}}(n)$*, the number of unbordered factors of* $\mathbf{d}$ *of length* n*, is equal to* 2 *for all* $n \geq 1$*.*

Chapter 10

Primitive Words and Lyndon Words

Remark 54. Sections of this chapter are taken essentially verbatim from [57].

10.1 Lyndon words

Recall that in Chapter 7 we defined a word w to be a power if it is of the form $w = x^\alpha$ for some $\alpha > 1$. Otherwise w is called *primitive*. Thus while `murmur` is a power, `murder` is primitive.

Let Σ be an ordered alphabet. We recall the usual definition of lexicographic order on the words in Σ^*. We write $w < x$ if either

(a) w is a proper prefix of x; or

(b) there exist words y, z, z' and letters $a < b$ such that $w = yaz$ and $x = ybz'$.

For example, using the usual ordering of the alphabet, we have `common` $<$ `con` $<$ `conjugate`. As usual, we write $w \leq x$ if $w < x$ or $w = x$.

A word w is a *conjugate* of a word x if there exist words u, v such that $w = uv$ and $w = vu$. Thus, for example, `enlist` and `listen` are conjugates. A word is said to be *Lyndon* if it is primitive and lexicographically least among all its conjugates. Thus, for example, `academy` is Lyndon, while `googol` and `googoo` are not. Lyndon words have received a great deal of attention in the combinatorics on words literature. For example, a finite word is Lyndon if and only if it is lexicographically less than each of its proper suffixes [40] and this can be tested in linear time.

Theorem 55. *Let* $\mathbf{x}$ *be a* k*-automatic sequence. The predicate* $P(i,j)$ *defined by "*$\mathbf{x}[i..j]$ *is primitive" is expressible.*

Proof. (due to Luke Schaeffer) It is easy to see that a word is a power if and only if it is equal to some cyclic shift of itself, other than the trivial shift. Thus a word is a power if and only if there is a d, $0 < d < j - i + 1$, such that $x[i..j-d] = x[i+d..j]$ and $x[j-d+1..j] = x[i..i+d-1]$. A word is primitive if there is no such d. □

Theorem 56. *Let* $\mathbf{x}$ *be a* k*-automatic sequence. The predicate* $LL(i,j,m,n)$ *defined by "*$\mathbf{x}[i..j] < \mathbf{x}[m..n]$*" is expressible.*

Proof. We have $\mathbf{x}[i..j] < \mathbf{x}[m..n]$ if and only if either

(a) $j - i < n - m$ and $\mathbf{x}[i..j] = \mathbf{x}[m..m+j-i]$; or

(b) there exists $t < \min(j-i, n-m)$ such that $\mathbf{x}[i..i+t] = \mathbf{x}[m..m+t]$ and $\mathbf{x}[i+t+1] < \mathbf{x}[m+t+1]$.

□

Theorem 57. *Let* $\mathbf{x}$ *be a* k*-automatic sequence. The predicate* $L(i,j)$ *defined by "*$\mathbf{x}[i..j]$ *is a Lyndon word" is expressible.*

Proof. It suffices to check that $\mathbf{x}[i..j]$ is lexicographically less than each of its proper suffixes, that is, that $LL(i,j,i',j)$ holds for all i' with $i < i' \leq j$. □

We can extend the definition of lexicographic order to infinite words in the obvious way. We can extend the definition of Lyndon words to (right-) infinite words as follows: an infinite word $\mathbf{x} = a_0 a_1 a_2 \cdots$ is Lyndon if it is lexicographically less than all its suffixes $\mathbf{x}[j..\infty] = a_j a_{j+1} \cdots$ for $j \geq 1$. Then we have the following theorems.

Theorem 58. *Let* $\mathbf{x}$ *be a* k*-automatic sequence. The predicate* $LL_\infty(i,j)$ *defined by "*$\mathbf{x}[i..\infty] < \mathbf{x}[j..\infty]$*" is expressible.*

Proof. This is equivalent to $\exists t \geq 0$ such that $\mathbf{x}[i..i+t-1] = \mathbf{x}[j..j+t-1]$ and $\mathbf{x}[i+t] < \mathbf{x}[j+t]$. □

Theorem 59. *Let* $\mathbf{x}$ *be a* k*-automatic sequence. The predicate* $L_\infty(i)$ *defined by "*$\mathbf{x}[i..\infty]$ *is an infinite Lyndon word" is expressible.*

Proof. This is equivalent to $LL_\infty(i,j)$ holding for all $j > i$. □

10.2 Lyndon Factorization

Siromoney et al. [101] proved that every infinite word $\mathbf{x} = a_0a_1a_2\cdots$ can be factorized uniquely in exactly one of the following two ways:

(a) as $\mathbf{x} = w_1w_2w_3\cdots$ where each w_i is a finite Lyndon word and $w_1 \geq w_2 \geq w_3\cdots$; or

(b) as $\mathbf{x} = w_1w_2w_3\cdots w_r\mathbf{x}$ where w_i is a finite Lyndon word for $1 \leq i \leq r$, and $\mathbf{x}$ is an infinite Lyndon word, and $w_1 \geq w_2 \geq \cdots \geq w_r \geq \mathbf{x}$.

If (a) holds we say that the Lyndon factorization of $\mathbf{x}$ is infinite; otherwise we say it is finite.

Ido and Melançon [75, 64] gave an explicit description of the Lyndon factorization of the Thue-Morse word $\mathbf{t}$ and the period-doubling sequence (among other things). For the Thue-Morse word, this factorization is given by

$$\mathbf{t} = w_1w_2w_3w_4\cdots = (011)(01)(0011)(00101101)\cdots,$$

where each term in the factorization, after the first, is double the length of the previous. Séébold [96] and Černý [27] generalized these results to other related automatic sequences.

In this section, generalizing the work of Ido, Melançon, Séébold, and Černý, we prove that the Lyndon factorization of a k-automatic sequence is itself k-automatic. Of course, we need to explain how the factorization is encoded. The easiest and most natural way to do this is to use an infinite word over $\{0, 1\}$, where the 1's indicate the positions where a new term in the factorization begins. Thus the i'th 1, for $i \geq 0$, appears at index $|w_1w_2\cdots w_i|$. For example, for the Thue-Morse word, this encoding is given by

$$100101000100000001\cdots.$$

If the factorization is infinite, then there are infinitely many 1's in its encoding; otherwise there are finitely many 1's.

In order to prove the theorem, we need a number of results. We draw a distinction between a *factor* f of $\mathbf{x}$ (which is just a word) and an *occurrence* of that factor (which specifies the exact position at which f occurs). For example, in the Thue-Morse word $\mathbf{t}$, the factor 0110 occurs as $\mathbf{x}[0..3]$ and $\mathbf{x}[11..15]$ and many other places. We call [0..3] and [11..15], and so forth, the *occurrences* of 0110. An occurrence is said to be Lyndon if the word at that position is Lyndon. We say an occurrence $O_1 = [i..j]$ is *inside* an occurrence

$O_2 = [i'..j']$ if $i' \leq i$ and $j' \geq j$. If, in addition, either $i' < i$ or $j < j'$ (or both), then we say O_1 is *strictly inside* O_2. These definitions are easily extended to the case where j or j' are equal to ∞, and they correspond to the predicates I (inside) and SI (strictly inside) given below:

$$\begin{array}{rcl} I(i,j,i',j') & \text{is} & i' \leq i \text{ and } j' \geq j \\ SI(i,j,i',j') & \text{is} & I(i,j,i',j') \text{ and } ((i' < i) \text{ or } (j' > j)) \end{array}$$

An infinite Lyndon factorization

$$\mathbf{x} = w_1 w_2 w_3 \cdots$$

then corresponds to an infinite sequence of occurrences

$$[i_1..j_1], [i_2..j_2], \cdots$$

where $w_n = \mathbf{x}[i_n..j_n]$ and $i_{n+1} = j_n + 1$ for $n \geq 1$, while a finite Lyndon factorization

$$\mathbf{x} = w_1 w_2 \cdots w_r \mathbf{x}$$

corresponds to a finite sequence of occurrences

$$[i_1..j_1], [i_2..j_2], \ldots, [i_r..j_r], [i_{r+1}..\infty]$$

where $w_n = \mathbf{x}[i_n..j_n]$ and $i_{n+1} = j_n + 1$ for $1 \leq n \leq r$.

Theorem 60. *Let* $\mathbf{x}$ *be an infinite word. Every Lyndon occurrence in* $\mathbf{x}$ *appears inside a term of the Lyndon factorization of* $\mathbf{x}$*.*

Proof. We prove the result for infinite Lyndon factorizations; the result for finite factorizations is exactly analogous.

Suppose the factorization is $\mathbf{x} = w_1 w_2 w_3 \cdots$. It suffices to show that no Lyndon occurrence can span the boundary between two terms of the factorization. Suppose, contrary to what we want to prove, that $u w_i w_{i+1} \cdots w_j v$ is a Lyndon word for some u that is a nonempty suffix of w_{i-1} (possibly equal to w_{i-1}), and v that is a nonempty prefix of w_{j+1} (possibly equal to w_{j+1}), and $i \leq j + 1$. (If $i = j + 1$ then there are no w_i's at all between u and v.)

Since u is a suffix of w_{i-1} and w_{i-1} is Lyndon, we have $u \geq w_{i-1}$. On the other hand, by the Lyndon factorization definition we have $w_{i-1} \geq w_i \geq \cdots \geq w_j \geq w_{j+1}$. But v

is a prefix of w_{j+1}, so just by the definition of lexicographic ordering we have $w_{j+1} \geq v$. Putting this all together we get $u \geq v$. So $ux \geq v$ for all words x.

On the other hand, since $uw_i \cdots w_j v$ is Lyndon, it must be lexicographically less than any proper suffix — for instance, v. So $uw_i \cdots w_j v < v$. Take $x = w_i \cdots w_j v$ to get a contradiction with the conclusion in the previous paragraph. □

Corollary 61. *The occurrence $[i..j]$ corresponds to a term in the Lyndon factorization of* **x** *if and only if*

(a) $[i..j]$ is Lyndon; and

(b) $[i..j]$ does not occur strictly inside any other Lyndon occurrence.

Proof. Suppose $[i..j]$ corresponds to a term w_n in the Lyndon factorization of **x**. Then evidently $[i..j]$ is Lyndon. If it occurred strictly inside some other Lyndon occurrence, say $[i'..j']$, then we know from Theorem 60 that $[i'..j']$ itself lies in inside some $[i_m, j_m]$, so $[i..j]$ must lie strictly inside $[i_m, j_m]$, which is clearly impossible.

Now suppose $[i..j]$ is Lyndon and does not occur strictly inside any other Lyndon occurrence. From Theorem 60 $[i..j]$ must occur inside some term of the factorization $[i'..j']$. If $[i..j] \neq [i'..j']$ then $[i..j]$ lies strictly inside $[i'..j']$, a contradiction. So $[i..j] = [i'..j']$ and hence corresponds to a term of the factorization. □

Corollary 62. *The predicate $LF(i, j)$ defined by "$[i..j]$ corresponds to a term of the Lyndon factorization of* **x***" is expressible.*

Proof. Indeed, by Corollary 61, the predicate $LF(i, j)$ can be defined by

$$L(i, j) \text{ and } \forall\, i', j'\ (SI(i, j, i', j') \implies \neg L(i', j')).$$

□

We can now prove the main result of this section.

Theorem 63. *Using the encoding mentioned above, the Lyndon factorization of a k-automatic sequence is itself k-automatic.*

Proof. Using the technique of [9], we can create an automaton that on input i expressed in base k, guesses j and checks if $LF(i, j)$ holds. If so, it outputs 1 and otherwise 0. To get the last i in the case that the Lyndon factorization is finite, we also accept i if $L_\infty(i)$ holds. □

We also have

Theorem 64. *Let* $\mathbf{x}$ *be a* k*-automatic sequence. It is decidable if the Lyndon factorization of* $\mathbf{x}$ *is finite or infinite.*

Proof. The construction given above in the proof of Theorem 63 produces an automaton that accepts finitely many distinct i (expressed in base k) if and only if the Lyndon factorization of $\mathbf{x}$ is finite. □

We computed the predicate described above and found the Lyndon factorization of the Thue-Morse sequence $\mathbf{t}$, the period-doubling sequence $\mathbf{d}$, the paperfolding sequence $\mathbf{p}$, and the Rudin-Shapiro sequence $\mathbf{r}$, and their negations. (The results for Thue-Morse and the period-doubling sequence were already given in [64], albeit in a different form.) The results are given in the theorem below.

Remark 65. There is a typo in the original paper regarding the Lyndon factorization of the Rudin-Shapiro sequence. The correct expression is given below.

Theorem 66. *The occurrences corresponding to the Lyndon factorization of each word is as follows:*

- *the Thue-Morse sequence* $\mathbf{t}$*:* $[0..2], [3..4], [5..8], [9..16], \ldots, [2^i + 1..2^{i+1}], \ldots$*;*
- *the negated Thue-Morse sequence* $\overline{\mathbf{t}}$*:* $[0..0], [1..\infty]$*;*
- *the Rudin-Shapiro sequence* $\mathbf{r}$*:* $[0..6], [7..30], \ldots, [2^i - 1..2^{i+2} - 2], \ldots$*;*
- *the negated Rudin-Shapiro sequence* $\overline{\mathbf{r}}$*:* $[0..0], [1..1], [2..2], [3..10], [11..42], [43..46], \ldots, [4^i - 4^{i-1} - 4^{i-2} - 1..4^i - 4^{i-1} - 2], [4^i - 4^{i-1} - 1..4^{i+1} - 4^i - 4^{i-1} - 1], \ldots$*;*
- *the paperfolding sequence* $\mathbf{p}$*:* $[0..6], [7..14], [15..30], \ldots, [2^i - 1..2^{i+1} - 2], \ldots$*;*
- *the negated paperfolding sequence* $\overline{\mathbf{p}}$*:* $[0..0], [1..1], [2..4], [5..9], [10..20], [21..84], \ldots, [(4^i - 1)/3..4(4^i - 1)/3], \ldots$*;*
- *the period-doubling sequence* $\mathbf{d}$*:* $[0..0], [1..4], [5..20], [21..84], \ldots,$ $[(4^i - 1)/3..4(4^i - 1)/3], \ldots$*;*
- *the negated period-doubling sequence* $\overline{\mathbf{d}}$*:* $[0..1], [2..9], [10..41], [42..169], \ldots,$ $[2(4^i - 1)/3..2(4^{i+1} - 1)/3 - 1], \ldots$.

10.3 Enumeration

The subword complexity function $\rho(n)$ of an infinite sequence $\mathbf{x}$ counts the number of distinct length-n factors of $\mathbf{x}$. We study it in greater detail in Chapter 11. There are also many variations, such as counting the number of palindromic factors or unbordered factors. If $\mathbf{x}$ is k-automatic, then all three of these are k-regular sequences [9]. We now show that the same result holds for the number $\rho_{\mathbf{x}}^{P}(n)$ of primitive factors of length n and for the number $\rho_{\mathbf{x}}^{L}$ of Lyndon factors of length n. We refer to these two quantities as the "primitive complexity" and "Lyndon complexity", respectively.

Theorem 67. *The function counting the number of length-n primitive (resp., Lyndon) factors of a k-automatic sequence $\mathbf{x}$ is k-regular.*

Proof. By the results of [28], it suffices to show that there is an automaton accepting the base-k representations of pairs (n, i) such that the number of i's associated with each n equals the number of primitive (resp., Lyndon) factors of length n.

To do so, it suffices to show that the predicate $P(n, i)$ defined by "the factor of length n beginning at position i is primitive (resp., Lyndon) and is the first occurrence of that factor in $\mathbf{x}$" is expressible. This is just

$$P(i, i+n-1) \quad \text{and} \quad \forall j < i \ \mathbf{x}[i..i+n-1] \neq \mathbf{x}[j..j+n-1],$$

(resp.,

$$L(i, i+n-1) \quad \text{and} \quad \forall j < i \ \mathbf{x}[i..i+n-1] \neq \mathbf{x}[j..j+n-1]).$$

□

We used our method to compute these sequences for the Thue-Morse sequence, and the results are given below.

Theorem 68. *Let $\rho_{\mathbf{t}}^{L}(n)$ denote the number of Lyndon factors of length n of the Thue-Morse sequence. Then*

$$\rho_{\mathbf{t}}^{L}(n) = \begin{cases} 1, & \text{if } n = 2^k \text{ or } 5 \cdot 2^k \text{ for } k \geq 1 \ ; \\ 2, & \text{if } n = 1 \text{ or } n = 5 \text{ or } n = 3 \cdot 2^k \text{ for } k \geq 0; \\ 0, & \text{otherwise.} \end{cases}$$

Theorem 69. *Let $\rho_{\mathbf{t}}^P(n)$ denote the number of primitive factors of length n of the Thue-Morse sequence. Then*

$$\rho_{\mathbf{t}}^P(n) = \begin{cases} 3 \cdot 2^t - 4, & \text{if } n = 2^t; \\ 4n - 2^t - 4, & \text{if } 2^t + 1 \leq n < 3 \cdot 2^{t-1}; \\ 5 \cdot 2^t - 6, & \text{if } n = 3 \cdot 2^{t-1}; \\ 2n + 2^{t+1} - 2, & \text{if } 3 \cdot 2^{t-1} < n < 2^{t+1}. \end{cases}$$

We can also state a similar result for the Rudin-Shapiro sequence.

Theorem 70. *Let $\rho_{\mathbf{r}}^L(n)$ denote the Lyndon complexity of the Rudin-Shapiro sequence. Then $\rho_{\mathbf{r}}^L(n) \leq 8$ for all n. This sequence is 2-automatic and there is an automaton of 2444 states that generates it.*

Proof. The proof was carried out by machine computation, and we briefly summarize how it was done.

First, we created an automaton A to accept all pairs of integers (n, i), represented in base 2, such that the factor of length n in $\mathbf{r}$, starting at position i, is a Lyndon factor, and is the first occurrence of that factor in $\mathbf{r}$. Thus, the number of distinct integers i associated with each n is $\rho_{\mathbf{r}}^L(n)$. The automaton A has 102 states.

Using the techniques in [28], we then used A to create matrices M_0 and M_1 of dimension 102×102, and vectors v, w such that $vM_xw = \rho_{\mathbf{r}}^L(n)$, if x is the base-2 representation of n. Here if $x = a_1a_2 \cdots a_i$, then by M_x we mean the product $M_{a_1}M_{a_2} \cdots M_{a_i}$.

From this we then created a new automaton A' where the states are products of the form vM_x for binary strings x and the transitions are on 0 and 1. This automaton was built using a breadth-first approach, using a queue to hold states whose targets on 0 and 1 are not yet known. From Theorem 24 in [57], we know that $\rho_{\mathbf{r}}^L(n)$ is bounded, so that this approach must terminate. It did so at 2444 states, and the product of the vM_x corresponding to each state with w gives an integer less than or equal to 8, thus proving the desired result and also providing an automaton to compute $\rho_{\mathbf{r}}^L(n)$. □

Remark 71. Note that the Lyndon complexity functions in Theorems 68 and 70 are bounded.

10.4 Finite Factorizations

Of course, the original Lyndon factorization was for finite words: every finite nonempty word x can be factored uniquely as a nonincreasing product $w_1 w_2 \cdots w_m$ of Lyndon words. We can apply this theorem to all prefixes of a k-automatic sequence. It is then natural to wonder if a *single* automaton can encode *all* the Lyndon factorizations of *all* finite prefixes. The answer is yes, as the following result shows.

Theorem 72. *Suppose* $\mathbf{x}$ *is a* k*-automatic sequence. Then there is an automaton* A *accepting*

$$\{(n,i)_k \ : \ \textit{the Lyndon factorization of } \mathbf{x}[0..n-1] \textit{ is } w_1 w_2 \cdots w_m \\ \textit{with } w_m = \mathbf{x}[i..n-1]\}.$$

Proof. As is well-known [40], if $w_1 w_2 \cdots w_m$ is the Lyndon factorization of x, then w_m is the lexicographically least suffix of x. So to accept $(n,i)_k$ we find i such that $\mathbf{x}[i..n-1] < \mathbf{x}[j..n-1]$ for $0 \le j < n$ and $i \ne j$. □

Given A, we can find the complete factorization of any prefix $\mathbf{x}[0..n-1]$ by using this automaton to find the appropriate i (as described in [58]) and then replacing n with i.

We carried out this construction for the Thue-Morse sequence, and the result is shown below in Figure 10.1.

In a similar manner, there is an automaton that encodes the factorization of *every* factor of a k-automatic sequence:

Theorem 73. *Suppose* $\mathbf{x}$ *is a* k*-automatic sequence. Then there is an automaton* A' *accepting*

$$\{(i,j,l)_k \ : \ \textit{the Lyndon factorization of } \mathbf{x}[i..j-1] \textit{ is } w_1 w_2 \cdots w_m \\ \textit{with } w_m = \mathbf{x}[l..j-1]\}.$$

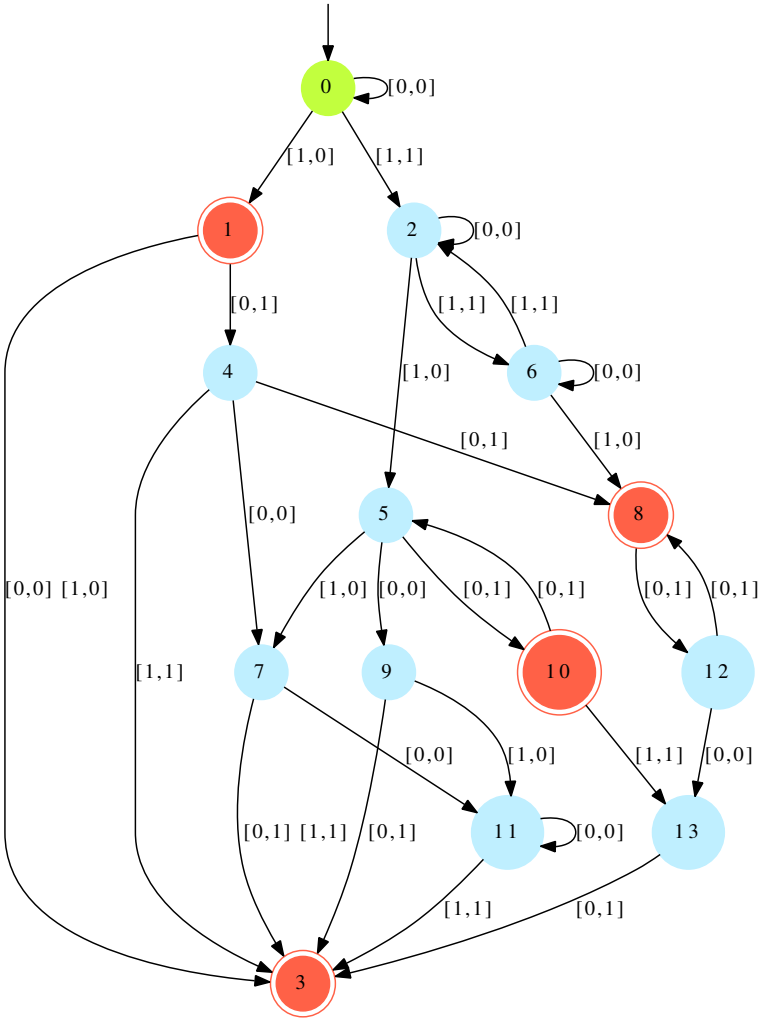


Figure 10.1: A finite automaton accepting the base-2 representation of (n, i) such that the Lyndon factorization of $\mathbf{t}[0..n-1]$ ends in the term $\mathbf{t}[i..n-1]$

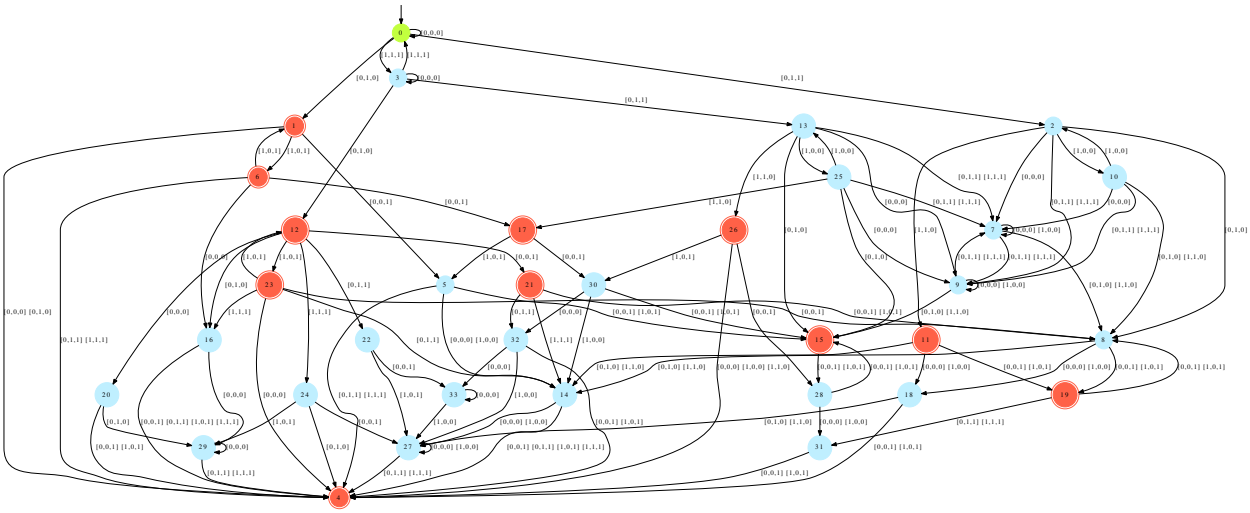


Figure 10.2: A finite automaton accepting the base-2 representation of (i, j, l) such that the Lyndon factorization of $\mathbf{t}[i..j-1]$ ends in the term $\mathbf{t}[l..j-1]$

We calculated A' for the Thue-Morse sequence using our method. It is a 34-state machine and is displayed in Figure 9.2.

Another quantity of interest is the number of terms in the Lyndon factorization of each prefix.

Theorem 74. *Let x be a k-automatic sequence. Then the sequence $(f(n))_{n\geq 0}$ defined by*

$$f(n) = \textit{the number of terms in the Lyndon factorization of } \mathbf{x}[0..n]$$

is k-regular.

Proof. We construct an automaton to accept

$$\{(n, i) \ : \ \exists j \leq n \text{ such that } L(i, j) \text{ and if } SI(i, j, i', j') \text{ and } 0 \leq i' \leq j' \leq n \text{ then } \neg L(i', j')\}.$$

□

For the Thue-Morse sequence the corresponding sequence satisfies the relations

$$\begin{aligned}
f(4n+1) &= -f(2n) + f(2n+1) + f(4n) \\
f(8n+2) &= -f(2n) + f(4n) + f(4n+2) \\
f(8n+3) &= -f(2n) + f(4n) + f(4n+3) \\
f(8n+6) &= -f(2n) - f(4n+2) + 3f(4n+3) \\
f(8n+7) &= -f(2n) + 2f(4n+3) \\
f(16n) &= -f(2n) + f(4n) + f(8n) \\
f(16n+4) &= -f(2n) + f(4n) + f(8n+4) \\
f(16n+8) &= -f(2n) + f(4n+3) + f(8n+4) \\
f(16n+12) &= -f(2n) - 2f(4n+2) + 3f(4n+3) + f(8n+4)
\end{aligned}$$

for $n \geq 1$, which allows efficient calculation of this quantity.

Chapter 11

Subword Complexity

Remark 75. This chapter is taken essentially verbatim from [58].

Theorem 76. *Suppose $(f(n))_{n\geq 0}$ is k-synchronized. Then there is an algorithm that, given the base-k representation of n, will compute the base-k representation of $f(n)$ in $O(\log n)$ time.*

Proof. We know there is a DFA $M = (Q, \Sigma_k \times \Sigma_k, \delta, q_0, F)$ accepting

$$L = \{(n, f(n))_k \ : \ n \geq 0\}.$$

Let $w = (n)_k$. It is easy to construct a $(|w| + 2)$-state DFA that accepts words with 0^*w in the first coordinate and no leading $[0, 0]$'s. Such a DFA accepts the language $K = \{(n, m)_k \ : \ m \geq 0\}$. We now construct a DFA H for the language $K \cap L$, where H has $|Q|(|w| + 2) = \Theta(|w|)$ states, using the familiar direct product construction from automata theory.

The only word in $K \cap L$ is $(n, f(n))_k$, so we apply a linear-time directed graph reachability algorithm (such as breadth-first or depth-first search) to the underlying transition graph of H. This finds the unique path $x \in (\Sigma_k \times \Sigma_k)^*$ from the initial state in W to an accepting state. Then x is labeled $(n, f(n))_k$, so reading the second coordinate yields the base-k representation of $f(n)$.

□

In this chapter we show that if $\mathbf{x}$ is a k-automatic sequence, then the subword complexity $\rho_{\mathbf{x}}(n)$ is k-synchronized. As an application, we generalize and simplify recent results of

Goldstein [53, 54]. Furthermore, we obtain analogous results for the number of length-n primitive words and the number of length-n powers.

We remark that there are a number of quantities about k-automatic sequences already known to be k-synchronized. These include the separator sequence [25], the repetitivity index [22], the recurrence function [28], and the "appearance" function [28]. The latter two examples were not explicitly stated to be k-synchronized in [28], but the result follows immediately from the proofs in that paper.

11.1 Subword Complexity

Cobham [29] proved that if $\mathbf{x}$ is a k-automatic sequence, then $\rho_{\mathbf{x}}(n) = O(n)$. Cassaigne [26] proved that any infinite word $\mathbf{x}$ satisfying $\rho_{\mathbf{x}}(n) = O(n)$ also satisfies $\rho_{\mathbf{x}}(n+1) - \rho_{\mathbf{x}}(n) = O(1)$. Carpi and D'Alonzo [23] showed that the subword complexity function $\rho_{\mathbf{x}}(n)$ is a k-regular sequence.

Charlier, Rampersad, and Shallit [28] found this result independently, using a somewhat different approach. They used the following idea. Call an occurrence of the factor $t = \mathbf{x}[i..i+n-1]$ "novel" if t does not appear as a factor of $\mathbf{x}[0..i+n-2]$. In other words, the leftmost occurrence of t in $\mathbf{x}$ is at position i. Then the number of factors of length n in $\mathbf{x}$ is equal to the number of novel occurrences of factors of length n. The property that $\mathbf{x}[i..i+n-1]$ is novel can be expressed as a predicate, as follows:

$$\{(n,i)_k \ : \ \forall j, 0 \le j < i \ \mathbf{x}[i..i+n-1] \ne \mathbf{x}[j..j+n-1]\} = \{(n,i)_k \ : \ \forall j, 0 \le j < i \ \exists m, 0 \le m < n \ \mathbf{x}[i+m] \ne \mathbf{x}[j+m]\}. \quad (11.1)$$

As shown in [28], the base-k representation of the integers satisfying any predicate of this form (expressible using quantifiers, integer addition and subtraction, indexing into a k-automatic sequence $\mathbf{x}$, logical operations, and comparisons) can be accepted by an explicitly-constructable deterministic finite automaton. From this, it follows that the sequence $\rho_{\mathbf{x}}(n)$ is k-regular, and hence can be computed explicitly in terms of the product of certain matrices and vectors depending on the base-k expansion of n.

We show that, in fact, the subword complexity function $\rho_{\mathbf{x}}(n)$ is k-synchronized. The main observation needed is the following (Theorem 78): in any sequence of linear complexity, the novel occurrences of factors are "clumped together" in a bounded number of contiguous blocks. This makes it easy to count them.

More precisely, let $\mathbf{x}$ be an infinite word and for any n consider the set of novel occurrences $E_{\mathbf{x}}(n) := \{i \ : \ \text{the occurrence } \mathbf{x}[i..i+n-1] \text{ is novel } \}$. We consider how $E_{\mathbf{x}}(n)$ evolves with increasing n.

As an example, consider the Thue-Morse sequence $\mathbf{t} = t_0 t_1 t_2 \cdots = 0110100110010110 \cdots$, defined by letting t_n be the number of 1's in the binary expansion of n, taken modulo 2. The gray squares in the rows of of Figure 11.1 depict the members of $E_{\mathbf{t}}(n)$ for the Thue-Morse sequence for $1 \leq n \leq 9$.

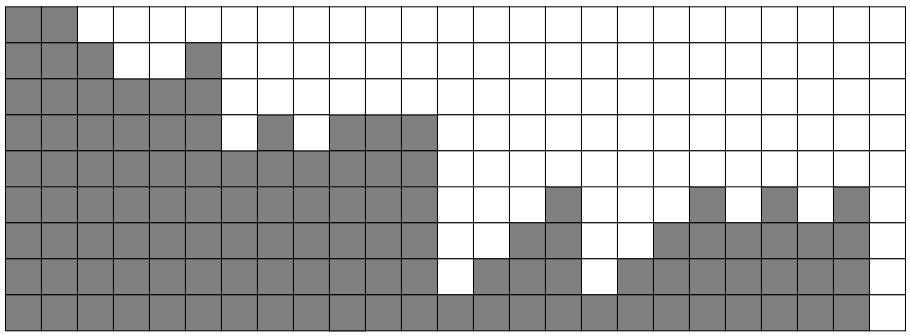


Figure 11.1: Evolution of novel occurrences of factors in the Thue-Morse sequence

Lemma 77. *Let* $\mathbf{x}$ *be an infinite word. If the factor of length* n *beginning at position* i *is a novel occurrence, so is*

(a) the factor of length $n+1$ *beginning at position* i*;*

(b) the factor of length $n+1$ *beginning at position* $i-1$ *(for* $i \geq 1$*).*

Proof. (a) Suppose the factor of length $n+1$ also occurs at some position $j < i$. Then the factor of length n also occurs at position j, contradicting the fact that it was a novel occurrence at i.

(b) Suppose the factor of length $n+1$ beginning at position $i-1$ occurs at some earlier position $j < i-1$. We can write the factor as ax, where a is a single letter and x is a word, so the factor of length n beginning at position i must also occur at position $j+1 < i$. But then it is not a novel occurrence. □

Theorem 78. *Let* $\mathbf{x}$ *be an infinite word. For* $n \geq 1$*, the number of contiguous blocks in* $E_{\mathbf{x}}(n)$ *is at most* $\rho_{\mathbf{x}}(n) - \rho_{\mathbf{x}}(n-1) + 1$.

Proof. We prove the claim by induction on n. For $n = 1$ the claim says there are at most $\rho_{\mathbf{x}}(1)$ contiguous blocks, which is evidently true, since there are at most $\rho_{\mathbf{x}}(1)$ novel factors of length 1.

Now assume the claim is true for all $n' < n$; we prove it for n. Consider the evolution of the novel occurrences of factors in going from length $n-1$ to n. Every occurrence that was previously novel is still novel, and furthermore in every contiguous block except the first, we get novel occurrences at one position to the left of the beginning of the block. So if row $n-1$ has t contiguous blocks, then we get $t-1$ novel occurrences at the beginning of each block, except the first. (Of course, the first block begins at position 0, since any factor beginning at position 0 is novel, no matter what the length is.) The remaining $\rho_{\mathbf{x}}(n)-\rho_{\mathbf{x}}(n-1)-(t-1)$ novel occurrences could be, in the worst case, in their own individual contiguous blocks. Thus row n has at most $t + \rho_{\mathbf{x}}(n) - \rho_{\mathbf{x}}(n-1) - (t-1) = \rho_{\mathbf{x}}(n) - \rho_{\mathbf{x}}(n-1) + 1$ contiguous blocks. □

In our Thue-Morse example, it is well-known that $\rho_{\mathbf{t}}(n) - \rho_{\mathbf{t}}(n-1) \leq 4$, so the number of contiguous blocks in any row is at most 5. This is achieved, for example, for $n = 6$.

Example 79. We give an example of a recurrent infinite word over a finite alphabet where the number of contiguous blocks in $E_{\mathbf{x}}(n)$ is unbounded. Consider the word

$$\mathbf{w} = \prod_{n \geq 1} (n)_2 = 1101110010111011110000 \cdots .$$

Then for each $n \geq 5$ the first occurrence of each of the words $0^{n-1}1$, $0^{n-2}11, \ldots, 0^2 1^{n-2}$ have a non-novel occurrence immediately following them, which shows there at at least $n-2$ blocks in $E_{\mathbf{w}}(n)$.

Corollary 80. *If* $\rho_{\mathbf{x}}(n) = O(n)$*, then there is a constant* C *such that every row* $E_{\mathbf{x}}(n)$ *in the evolution of novel occurrences consists of at most* C *contiguous blocks.*

Proof. By the result of Cassaigne [26], we know that there exists a constant C such that $\rho_{\mathbf{x}}(n) - \rho_{\mathbf{x}}(n-1) \leq C - 1$. By Theorem 78, we know there are at most C contiguous blocks in any $E_{\mathbf{x}}(n)$.

□

Theorem 81. *Let* $\mathbf{x}$ *be a* k*-automatic sequence. Then its subword complexity function* $\rho_{\mathbf{x}}(n)$ *is* k*-synchronized.*

Proof. Following [28], it suffices to show how to accept the language

$$\{(n, m)_k \ : \ n \geq 0 \text{ and } m = \rho_{\mathbf{x}}(n)\}$$

with a finite automaton. Here is a sketch of the argument. From our results above, we know that there is a finite constant $C \geq 1$ such that the number of contiguous blocks in any row of the factor evolution diagram is bounded by C. So we simply "guess" the endpoints of every block and then verify that each factor of length n starting at the positions inside blocks is a novel occurrence, while all other factors are not. Finally, we verify that m is the sum of the sizes of the blocks.

To fill in the details, we observe above in (11.1) that the predicate "the factor of length n beginning at position i of $\mathbf{x}$ is a novel occurrence" is solvable by a finite automaton. Similarly, given endpoints a, b and n, the predicates "every factor of length n beginning at positions a through b is a novel occurrence" and "no factor of length n beginning at positions a through b is a novel occurrence" are also solvable by a finite automaton. The length of each block is just $b - a + 1$, and it is easy to create an automaton that will check if the sums of the lengths of the blocks equals m, which is supposed to be $\rho_{\mathbf{x}}(n)$.

□

Applying Theorem 76 we get

Corollary 82. *Given a* k*-automatic sequence* $\mathbf{x}$*, there is an algorithm that, on input* n *in base* k*, will produce* $\rho_{\mathbf{x}}(n)$ *in base* k *in time* $O(\log n)$*.*

As another application, we can recover and improve some recent results of Goldstein [53, 54]. He showed how to compute the quantities $\limsup_{n \geq 1} \rho_{\mathbf{x}}(n)/n$ and $\liminf_{n \geq 1} \rho_{\mathbf{x}}(n)/n$ for the special case of k-automatic sequences that are the fixed points of k-uniform morphisms related to certain groups. Corollary 83 below generalizes these results to all k-automatic sequences.

Corollary 83. *There is an algorithm, that, given a* k*-automatic sequence* $\mathbf{x}$*, will compute* $\sup_{n \geq 1} \rho_{\mathbf{x}}(n)/n$*,* $\limsup_{n \geq 1} \rho_{\mathbf{x}}(n)/n$*, and* $\inf_{n \geq 1} \rho_{\mathbf{x}}(n)/n$*,* $\liminf_{n \geq 1} \rho_{\mathbf{x}}(n)/n$*.*

Proof. We already showed how to construct an automaton accepting $\{(n, \rho_{\mathbf{x}}(n))_k \ : \ n \geq 1\}$. Now we just use the results from [97, 93]. Notice that the lim sup corresponds to what is called the largest "special point" in [93].

□

Example 84. Continuing our example of the Thue-Morse sequence, Figure 11.2 displays a DFA accepting $\{(n, \rho_{\mathbf{t}}(n))_k \ : \ n \geq 0\}$. Inputs are given with the most significant digit first; the "dead" state and transitions leading to it are omitted.

Given an infinite word $\mathbf{x}$, we can also count the number of contiguous blocks in each $E_{\mathbf{x}}(n)$ for $n \geq 0$. (For the Thue-Morse sequence this gives the sequence $1, 1, 2, 1, 3, 1, 5, 3, 3, 1, \ldots$.) If $\mathbf{x}$ is k-automatic, then this sequence is also, as the following theorem shows:

Theorem 85. *If $\mathbf{x}$ is k-automatic then the sequence $(e(n))_{n\geq 0}$ counting the number of contiguous blocks in the n'th step $E_{\mathbf{x}}(n)$ of the evolution of novel occurrences of factors in $\mathbf{x}$ is also k-automatic.*

Proof. Since we have already shown that the number of contiguous blocks is bounded by some constant C if $\mathbf{x}$ is k-automatic, it suffices to show for each $i \leq C$ we can create an automaton to accept the language

$$\{(n)_k \ : \ E_{\mathbf{x}}(n) \text{ has exactly } i \text{ contiguous blocks } \}.$$

To do so, on input n in base k we guess the endpoints of the i contiguous nonempty blocks, verify that the length-n occurrences at those positions are novel, and that all other occurrences are not novel.

□

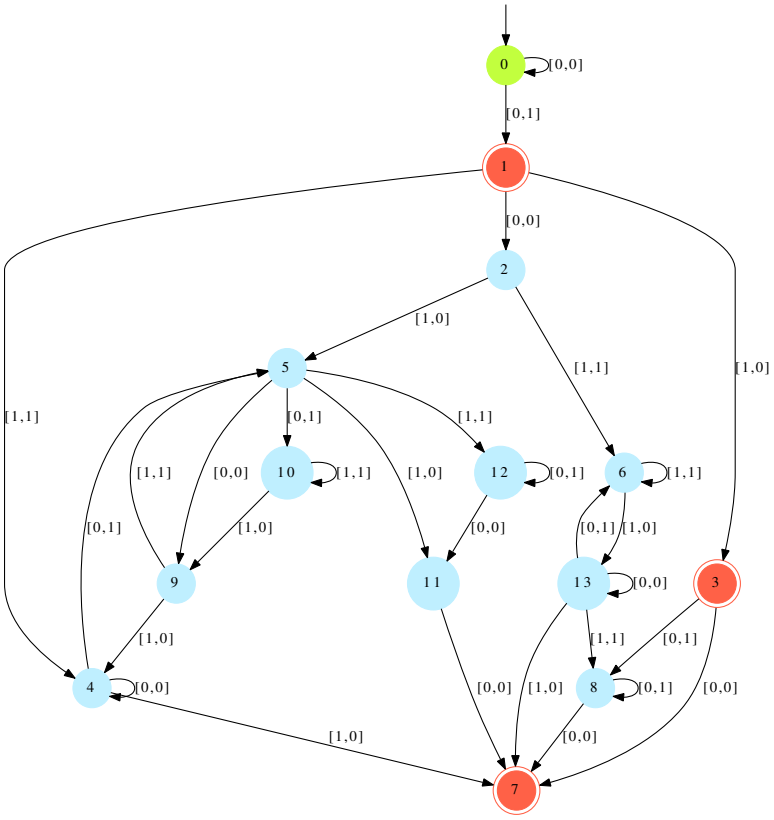


Figure 11.2: Automaton computing the subword complexity of the Thue-Morse sequence

Example 86. Figure 11.3 below gives the automaton computing the number $e(n)$ of contiguous blocks of novel occurrences of length-n factors for the Thue-Morse sequence. Here is a brief table:

n	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$e(n)$	1	1	2	1	3	1	5	3	3	1	5	5	5	3	3

Table 11.1: The first few terms of $e(n)$

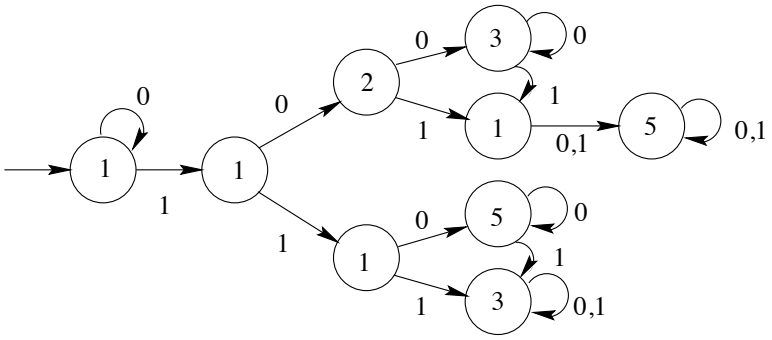


Figure 11.3: Automaton computing number of contiguous blocks of novel occurrences of length-n factors in the Thue-Morse sequence

11.2 Implementation

We wrote a program that, given an automaton generating a k-automatic sequence $\mathbf{x}$, will produce a deterministic finite automaton accepting the language $\{(n, \rho_{\mathbf{x}}(n))_k \ : \ n \geq 0\}$. We used the following variant which does not require advance knowledge of the bound on the first difference of $\rho_{\mathbf{x}}(n)$:

1. Construct an automaton R that accepts (n, s, e, ℓ) if, for factors of length n, the next contiguous block of novel occurrences after position s ends at position e and has length ℓ. If there are no blocks past s, accept $(n, s, s, 0)$.

2. Construct an automaton M_0 that accepts $(n, 0, 0)$.

3. Construct an automaton M_{j+1} that accepts (n, S, e) if there exist s and S' such that

   (i) M_j accepts (n, S', s)

   (ii) R accepts $(n, s, e, S - S')$.

4. If $M_{j+1} = M_j$ then we are done. We create an automaton that accepts (n, S) if there exists e such that M_j accepts (n, S, e).

Besides the automaton depicted in Figure 11.1, we ran our program on the paperfolding sequence and the period-doubling sequence. The former gives a DFA of 20 states and the latter of 7 states. In the original paper we omitted them for space considerations. Here, the results are depicted below in Figures 11.4 and 11.5.

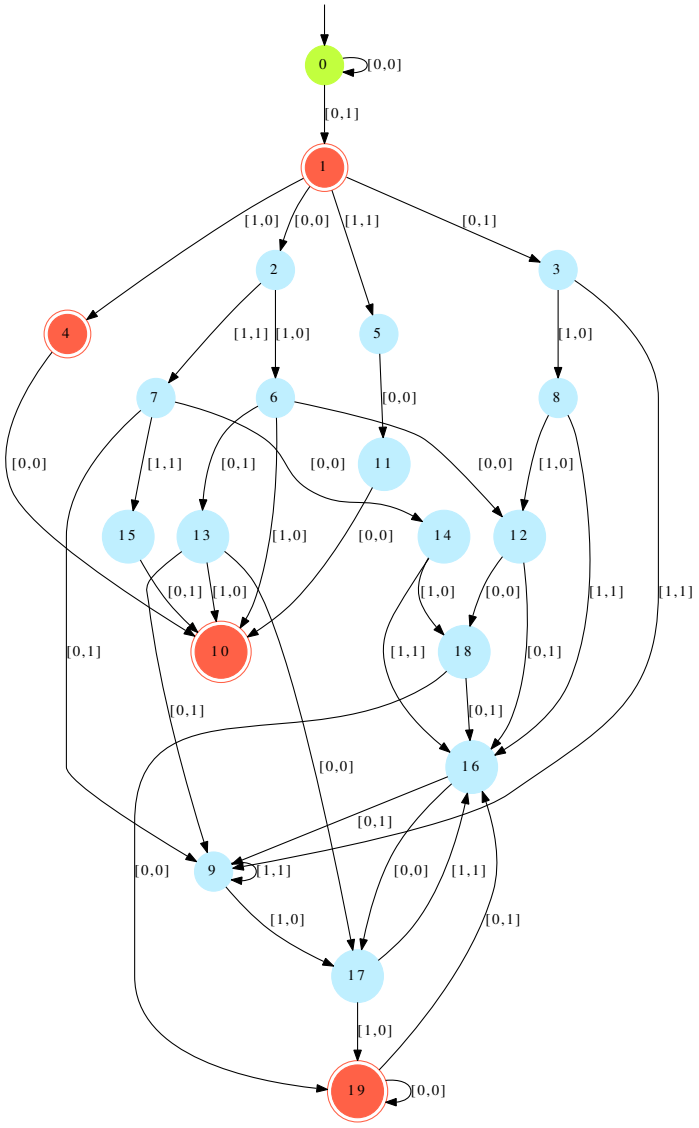


Figure 11.4: Automaton computing the subword complexity of the paperfolding sequence

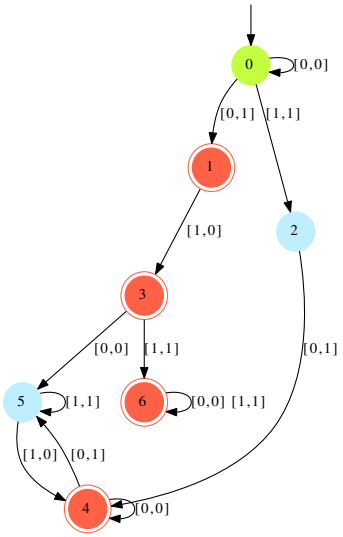


Figure 11.5: Automaton computing the subword complexity of the period-doubling sequence

11.3 Powers and Primitive Words

The following lemma says that if we consider the starting positions of length-n powers in a word x, then there must be large gaps between contiguous blocks of such starting positions.

Lemma 87. *Let z be a finite or infinite word, and let $n \geq 2$ be an integer. Suppose there exist integers i, j such that*

(a) $w_1 := z[i..i+n-1]$ *is a power;*

(b) $w_2 := z[j..j+n-1]$ *is a power;*

(c) $i < j \leq i + n/3$.

Then $z[t..t-n-1]$ is a power for $i \leq t \leq j$. Furthermore, if x_1 is the Lyndon root of w_1, then x_1 is also the Lyndon root of each word $z[t..t-n-1]$.

Proof. Let x_1 be the primitive root of w_1 and x_2 be the primitive root of w_2. Since x_1 and x_2 are powers, there exist integers $p_1, p_2 \geq 2$ such that $w_1 = x_1^{p_1}$ and $w_2 = x_2^{p_2}$.

Since w_1 and w_2 are both of length n, and since their starting positions are related by $i < j \leq i + n/3$, it follows that the word $v := z[j..i+n-1]$ is common to both w_1 and w_2, and $|v| = i + n - j \geq i + 2n/3 + n/3 - j \geq 2n/3$.

Now there are three cases to consider:

$$\text{(a) } |x_1| > |x_2|; \qquad \text{(b) } |x_1| < |x_2|; \qquad \text{(c) } |x_1| = |x_2|.$$

Case (a): We must have $p_2 > p_1 \geq 2$, so $p_2 \geq 3$. Since v is a suffix of w_1, it has period $|x_1| \leq n/2$. Since v is a prefix of w_2, it has period $|x_2| \leq n/3$. Then $|v| \geq 2n/3 \geq |x_1| + |x_2|$. By a theorem of Fine and Wilf [44], it now follows that v, and hence x_1, has period $p := \gcd(|x_1|, |x_2|) \leq |x_2| < |x_1|$. Now p is less than $|x_1|$ and also divides it, so this means x_1 is a power, a contradiction, since we assumed x_1 is primitive. So this case cannot occur.

Case (b) gives a similar contradiction.

Case (c): We have $p_1 = p_2 \geq 2$. Then the last occurrence of x_1 in w_1 lies inside x_2^2, and so x_1 is a conjugate of x_2. Hence w_1 is a conjugate of w_2. It now follows that $z[t..t+n-1]$ is a conjugate of w_1 for every t, $i \leq t \leq j$. But the conjugate of a power is itself a power, and we are done.

□

Remark 88. The bound of $n/3$ in the statement of Lemma 87 is best possible, as shown by the following class of examples. Let h be the morphism that maps $1 \to 21$ and $2 \to 22$, and consider the word $h^i(122122121212)$. This word is of length $12 \cdot 2^i$, and contains squares of length $3 \cdot 2^{i+1}$ starting in the first $3 \cdot 2^i$ positions, and cubes of length $3 \cdot 2^{i+1}$ ending in the last $2^i + 1$ positions. This achieves a gap of $n/3 + 1$ infinitely often.

Theorem 89. *If* $\mathbf{x}$ *is a* k*-automatic sequence, then the appearance function* $A_{\mathbf{x}}(n) = O(n)$.

Proof. First, recall from Theorem 13 that the appearance function is k-synchronized. From [25, Prop. 2.5] we know k-synchronized functions are $O(n)$.

□

As before, we now consider the occurrences of length-n powers in $\mathbf{x}$:

Lemma 90. *If* $\mathbf{x}$ *is* k*-automatic, then there are only a constant number of maximal blocks of novel occurrences of length-*n *powers in* $\mathbf{x}$.

Proof. To begin with, we consider maximal blocks of length-n powers in $\mathbf{x}$ (not considering whether they are novel occurrences). From Theorem 89 we know that every length-n factor must occur at a position $< Cn$, for some constant C (depending on $\mathbf{x}$). We first argue that the number of maximal blocks of length-n powers, up to the position of the last length-n power to occur for the first time, is at most $3C$.

Suppose there $\geq 3C+1$ such blocks. Then Lemma 87 says that any two such blocks must be separated by at least $n/3$ positions. So the first occurrence of the last factor to occur occurs at a position $\geq (3C)(n/3) = Cn$, a contradiction.

So using a constant number of blocks, in which each position of each block starts a length-n factor that is a power, we cover the starting positions of all such factors. It now remains to process these blocks to remove occurrences of length-n powers that are not novel.

The first thing we do is remove from each block the positions starting length-n factors that have already occurred *in that block*. This has the effect of truncating long blocks. The new blocks have the property that each factor occurring at the starting positions in the blocks never appeared before in that block.

Above we already proved that inside each block, the powers that begin at each position are all powers of some conjugate of a fixed Lyndon word. Now we process the blocks associated with the same Lyndon root together, from the first (leftmost) to the last. At each step, we remove from the current block all the positions where length-n factors begin that have appeared in any previous block. When all blocks have been processed, we need to see that there are still at most a constant number of contiguous blocks remaining.

Suppose the associated Lyndon root is y, with $|y| = d$. Each position in a block is the starting position of a power of a conjugate of y, and hence corresponds to a right rotation of y by some integer i, $0 \leq i < d$. Thus each block B_j actually corresponds to some I_j that is a contiguous subblock of $0, 1, \ldots, d-1$ (thought of as arranged in a circle).

As we process the blocks associated with y from left to right we replace I_j with $I'_j := I_j - (I_1 \cup \cdots \cup I_{j-1})$. Now if $I \subseteq \{0, 1, \ldots, d-1\}$ is a union of contiguous subblocks, let $\#I$ be the number of contiguous subblocks making up I. We claim that

$$\#I'_1 + \#I'_2 + \cdots + \#I'_n + \#(\bigcup_{1 \leq i \leq n} I'_i) \leq 2n. \tag{11.2}$$

To see this, suppose that when we set $I'_n := I_n - (I_1 \cup \cdots \cup I_{n-1})$, the subblock I_n has an intersection with t of the lower-numbered subblocks. Forming the union $(\bigcup_{1 \le i \le n} I'_i)$ then obliterates t subblocks and replaces them with 1. But I'_n has $t-1$ new subblocks, plus at most 2 at either edge (see Figure 11.6). This means that the left side of (11.2) increases by at most $(1-t)+(t-1)+2 = 2$. Doing this n times gives the result.

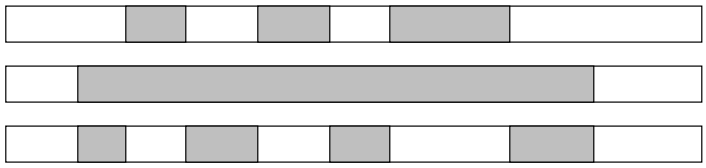


Figure 11.6: How the number of blocks changes

Now at the end of the procedure there will be at least one interval in the union of all the I_i, so $\#I'_1 + \#I'_2 + \cdots + \#I'_n \le 2n-1$, and we have proved (11.2).

Earlier we showed that there are at most $3C$ maximal blocks of length-n powers, up to the position of the last length-n power to occur for the first time. Then, after processing these blocks to remove positions corresponding to factors that occurred earlier, we will have at most $2(3C) = 6C$ blocks remaining.

□

Corollary 91. *If* $\mathbf{x}$ *is* k*-automatic, then*

- *the function counting the number of distinct length-*n *factors that are powers is* k*-synchronized;*
- *the function counting the number of distinct length-*n *factors that are primitive words is* k*-synchronized.*

Proof. Suppose $\mathbf{x}$ is k-automatic, and generated by the DFAO M. From the Lyndon-Schützenberger theorem [73], we know that a word x is a power if and only if there exist nonempty words y, z such that $x = yz = zy$. Thus, we can express the predicate $P(i,j) :=$ "$\mathbf{x}[i..j]$ is a power" as follows: "there exists d, $0 < d < j-i+1$, such that $\mathbf{x}[i..j-d] = \mathbf{x}[i+d..j]$ and $\mathbf{x}[j-d+1..j] = \mathbf{x}[i..i+d-1]$". Furthermore, we can express the predicate $P'(i,n) :=$ "$\mathbf{x}[i..i+n-1]$ is a length-n power and the first occurrence of that power in $\mathbf{x}$", as

$$P(i, i+n-1) \wedge (\forall i',\ 0 \le i' < i,\ \neg P(i', i'+n-1)).$$

From Lemma 90 we know that the novel occurrences of length-n powers are clustered into a finite number of blocks. Then, as in the proof of Theorem 81, we can guess the endpoints of these blocks, and verify that the length-n factors beginning at the positions inside the blocks are novel occurrences of powers, while those outside are not, and sum the lengths of the blocks, using a finite automaton built from M. Thus, the function counting the number of length-n powers in $\mathbf{x}$ is k-synchronized.

The number of length-n primitive words in $\mathbf{x}$ is then also k-synchronized, since it is expressible as the total number of words of length n, minus the number of length-n powers.

□

Remark 92. Using the technique above, we can prove analogous results for the functions counting the number of length-n words that are α-powers, for any fixed rational number $\alpha > 1$.

11.4 Unsynchronized Sequences

It is natural to wonder whether other aspects of k-automatic sequences are always k-synchronized. We give an example that is not.

Charlier et al. [28] showed that $u_{\mathbf{x}}(n)$, the number of unbordered factors of length n of a sequence $\mathbf{x}$, is k-regular if $\mathbf{x}$ is k-automatic. They also gave a conjecture for recursion relations defining $u_{\mathbf{t}}(n)$ where $\mathbf{t}$ is the Thue-Morse sequence; this conjecture has recently been verified by Goč and Shallit [56] and is presented in Chapter 9 of this Thesis.

We give here an example of a k-automatic sequence where the number of unbordered factors of length n is not k-synchronized.

Consider the characteristic sequence of the powers of 2: $\mathbf{c} := 0110100010\cdots$.

Theorem 93. *The sequence* $\mathbf{c}$ *is* 2*-automatic, but the function* $u_{\mathbf{c}}(n)$ *counting the number of unbordered factors is not* 2*-synchronized.*

Proof. It is not hard to verify that $\mathbf{c}$ is 2-automatic and that $\mathbf{c}$ has exactly $r+2$ unbordered factors of length 2^r+1, for $r \geq 2$ — namely, the factors beginning at positions 2^i for $0 \leq i \leq r$, and the factor beginning at position 2^r+1. However, if $u_{\mathbf{c}}(n)$ were 2-synchronized, then reading an input where the first component looks like 0^i10^r1 (and hence a representation of $2^{r-1}+1$) for large r would force the transitions to enter a cycle. If the corresponding transitions for the second component contained a nonzero entry, this would force $u_{\mathbf{c}}(n)$ to

grow linearly with n when n is of the form 2^r+1. Otherwise, the corresponding transitions for the second component are just 0's, in which case $u_{\mathbf{c}}(n)$ is bounded above by a constant, for n of the form 2^r+1. Both cases lead to a contradiction.

□

Chapter 12

Conclusion and Open Problems

In this thesis we considered various properties of k-automatic sequences. In Chapter 3 we described a decidable logical theory in which many of these sequences are expressible. In the following chapters we proceeded to show many examples of properties that feasibly decidable in our system. We catalogued the results for a list of well-known k-automatic sequences. Some of our results are novel, others improve on previous 'hand derived' theorems and yet others confirm known results. Not every property we embarked to decide has been feasible and below we give a list of still open problems.

12.1 Keränen's Word

A word w is an *abelian square* if $w = xy$ where $|x| = |y|$ and x is a permutation of y. In [65] Keränen gave a description of an 85-automatic sequence that avoids abelian squares over a 4 letter alphabet. The sequence is defined as the fixed point of the following morphism:

a →abcacdcbcdcadcdbdabacabadbabcbdbcbacbcdcacbabdabacadcbcdcacdbcbacbcdcacdcbdcdadbdcbca
b →bcdbdadcdadbadacabcbdbcbacbcdcacdcbdcdadbdcbcabcbdbadcdadbdacdcbdcdadbdadcadabacadcdb
c →cdacabadabacbabdbcdcacdcbdcdadbdadcadabacadcdbcdcacbadabacabdadcadabacabadbabcbdbadac
d →dabdbcbabcbdcbcacdadbdadcadabacabadbabcbdbadacdadbdcbabcbdbcabadbabcbdbcbacbcdcacbabd

Shallit conjectured (personal communication with the author of this thesis) that the appearance function of this Keränen sequence satisfies:

$$A_{\mathbf{e}}(n) = \begin{cases} 7, & \text{if } n = 1; \\ 21, & \text{if } n = 2; \\ 102, & \text{if } n = 3; \\ 1617, & \text{if } n = 4; \\ 340 \cdot 5^t + n - 85, & \text{if } n \geq 5 \text{ and } t = \lceil \log_{85}(n-1) \rceil. \end{cases}$$

We were unable to confirm this conjecture as the computation quickly became infeasible due to the large out-degrees of each state of the intermediate automata.

12.2 Lempel-Ziv and Crochemore Factorizations

In Chapter 10 we show that the Lyndon factorization of a k-automatic sequence is itself k-automatic. It is natural to ask if we can do the same for the Crochemore factorization. The *Crochemore factorization* of an infinite word $\mathbf{x} = z_1 z_2 z_3 \cdots$ satisfies that each z_i is either

(a) a single letter not seen before or

(b) the longest factor occurring already in an earlier position.

Another similar factorization is the Lempel-Ziv factorization. The *Lempel-Ziv factorization* $\mathbf{x} = y_1 y_2 \cdots$ is defined by setting y_m to be the shortest prefix of $y_m y_{m+1} \cdots$ that occurs only once in $y_1 y_2 \cdots y_m$.

In order to show that the Lyndon factorization is k-automatic we took advantage of a 'local' property of the terms of Lyndon factorization. Perhaps similar local properties can be proven about the terms of the Crochemore an Lempel-Ziv factorizations as well.

12.3 Further Work

In Subsection 2.5.3 we introduced the regular paperfolding sequence. Recall that the regular paperfolding sequence corresponds to folding a piece of paper consistently in the same direction over and over again. Instead, the *general paperfolding sequence* is a defined on an (infinite) sequence of folding instructions (f_i) and therefore is actually an uncountable

class of sequences. The general paperfolding sequence can be defined as $\mathbf{g} = \lim_{n\to\infty} g(n)$ where $g(0) = f_0$ and

$$g(n) = g(n-1)\ f_n\ \overline{g(n-1)^R}.$$

for all $n \geq 1$.

Alternatively, the paperfolding sequence can be described by the formula

$$g_n = \begin{cases} f_r, & \text{if } m = 1 \pmod 4; \\ \overline{f_r}, & \text{if } m = 3 \pmod 4, \end{cases}$$

where $n + 1 = m \cdot 2^r$ and m is odd.

It turns out that we can build a DFAO that reads in $(n)_2$ and a finite prefix of (f_i) to generate $\mathbf{g}_n$. In principle, this could mean that many of the interesting properties such as the appearance function could be computed for this uncountable class of sequences. The implementation has proven to be difficult, in particular because the treatment of trailing zeroes we describe in Subsection 3.7.1 is no longer applicable. Perhaps an approach where the basic building block consisted of Büchi automata (first described here [20]) would be more fruitful. Certainly, it merits further research.

References

[1] A. M. Shur. A. V. Samsonov. Binary patterns in binary cube-free words: Avoidability and growth. *Proc. 14th Mons Days of Theoretical Computer Science*, pages 1–7, 2012.

[2] J.-P. Allouche, M. Baake, J. Cassaigne, and D. Damanik. Palindrome complexity. *Theoret. Comput. Sci.*, 292:9–31, 2003.

[3] J.-P. Allouche and M. Bousquet-Mélou. Facteurs des suites de Rudin-Shapiro généralisées. *Bull. Belgian Math. Soc.*, 1:145–164, 1994.

[4] J.-P. Allouche, N. Rampersad, and J. Shallit. Periodicity, repetitions, and orbits of an automatic sequence. *Theoret. Comput. Sci.*, 410:2795–2803, 2009.

[5] J.-P. Allouche, K. Scheicher, and R. F. Tichy. Regular maps in generalized number systems. *Math. Slovaca*, 50:41–58, 2000.

[6] J.-P. Allouche and J. O. Shallit. The ring of k-regular sequences. *Theoret. Comput. Sci.*, 98:163–197, 1992.

[7] J.-P. Allouche and J. O. Shallit. The ubiquitous Prouhet-Thue-Morse sequence. In C. Ding, T. Helleseth, and H. Niederreiter, editors, *Sequences and Their Applications, Proceedings of SETA '98*, pages 1–16. Springer-Verlag, 1999.

[8] J.-P. Allouche and J. O. Shallit. The ring of k-regular sequences, II. *Theoret. Comput. Sci.*, 307:3–29, 2003.

[9] Allouche, J.-P., N. Rampersad, and J. Shallit. Periodicity, repetitions, and orbits of an automatic sequence. *Theoret. Comput. Sci.*, 410:2795–2803, 2009.

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

[11] Allouche, J.-P. and J. O. Shallit. The ring of k-regular sequences. *Theoret. Comput. Sci.*, 98:163–197, 1992.

[12] J. Bell, E. Charlier, A. Fraenkel, and M. Rigo. A decision problem for ultimately periodic sets in non-standard numeration systems. *Internat. J. Algebra Comput.*, 19:809–839, 2009.

[13] J. Berstel. *Axel Thue's Papers on Repetitions in Words: a Translation*. Number 20 in Publications du Laboratoire de Combinatoire et d'Informatique Mathématique. Université du Québec à Montréal, February 1995.

[14] J. Berstel. An exercise on Fibonacci representations. *RAIRO Inform. Théor. App.*, 35:491–498, 2001.

[15] J. Berstel and C. Reutenauer. *Rational Series and Their Languages*. Springer-Verlag, 1988.

[16] J. Berstel and C. Reutenauer. *Noncommutative Rational Series With Applications*, volume 137 of *Encyclopedia of Mathematics and Its Applications*. Cambridge University Press, 2011.

[17] S. Brown, N. Rampersad, J. Shallit, and T. Vasiga. Squares and overlaps in the Thue-Morse sequence and some variants. *RAIRO Inform. Théor. App.*, 40:473–484, 2006.

[18] V. Bruyère and G. Hansel. Bertrand numeration systems and recognizability. *Theoret. Comput. Sci.*, 181:17–43, 1997.

[19] Véronique Bruyère, Georges Hansel, Christian Michaux, and Roger Villemaire. Logic and p-recognizable sets of integers. *Bull. Belg. Math. Soc*, 1:191–238, 1994.

[20] J. R. Büchi. On a decision method in restricted second-order arithmetic. In *International Congress on Logic, Methodology, and Philosophy of Science*, pages 1–11. Stanford University Press, 1962.

[21] Serina Camungol and Narad Rampersad. Concerning kurosakiâĂŹs squarefree word. Submitted, 2013.

[22] A. Carpi and V. D'Alonzo. On the repetitivity index of infinite words. *Internat. J. Algebra Comput.*, 19:145–158, 2009.

[23] A. Carpi and V. D'Alonzo. On factors of synchronized sequences. *Theoretical Computer Science*, 411(44âĂŞ46):3932 – 3937, 2010.

[24] A. Carpi and V. D'Alonzo. On factors of synchronized sequences. To appear, *Theor. Comput. Sci.*, 2011.

[25] A. Carpi and C. Maggi. On synchronized sequences and their separators. *RAIRO Inform. Théor. App.*, 35:513–524, 2001.

[26] J. Cassaigne. Special factors of sequences with linear subword complexity. In J. Dassow, G. Rozenberg, and A. Salomaa, editors, *Developments in Language Theory II*, pages 25–34. World Scientific, 1996.

[27] A. Černý. Lyndon factorization of generalized words of Thue. *Discrete Math. & Theoret. Comput. Sci.*, 5:17–46, 2002.

[28] E. Charlier, N. Rampersad, and J. Shallit. Enumeration and decidable properties of automatic sequences. In G. Mauri and A. Leporati, editors, *Developments in Language Theory, 15th International Conference, DLT 2011*, volume 6795 of *Lecture Notes in Computer Science*, pages 165–179. Springer, 2011.

[29] A. Cobham. Uniform tag sequences. *Math. Systems Theory*, 6:164–192, 1972.

[30] J. C. Costa. Biinfinite words with maximal recurrent unbordered factors. *Theoret. Comput. Sci.*, 290:2053–2061, 2003.

[31] J. D. Currie. Lexicographically least words in the orbit closure of the Rudin-Shapiro word. `http://arxiv.org/pdf/0905.4923`, 2010.

[32] J. D. Currie and K. Saari. Least periods of factors of infinite words. *RAIRO Inform. Théor. App.*, 43:165–178, 2009.

[33] D. Damanik. Local symmetries in the period-doubling sequence. *Disc. Appl. Math.*, 100:115–121, 2000.

[34] F. M. Dekking, M. Mendès France, and A. J. van der Poorten. Folds! *Math. Intelligencer*, 4:130–138, 173–181, 190–195, 1982. Erratum, **5** (1983), 5.

[35] F. Durand. A characterization of substitutive sequences using return words. *Discrete Math.*, 179:89–101, 1998.

[36] F. Durand. Linearly recurrent subshifts have a finite number of non-periodic subshift factors. *Ergod. Theory & Dynam. Sys.*, 20:1061–1078, 2000.

[37] F. Durand, B. Host, and C. Skau. Substitution dynamical systems, bratteli diagrams, and dimension groups. *Ergod. Theory & Dynam. Sys.*, 19:953–993, 1999.

[38] J.-P. Duval. Une caractérisation de la période d'un mot fini par la longueur de ses facteurs primaires. *C. R. Acad. Sci. Paris*, 290:A359–A361, 1980.

[39] J.-P. Duval. Relationship between the period of a finite word and the length of its unbordered segments. *Discrete Math.*, 40:31–44, 1982.

[40] J. P. Duval. Factorizing words over an ordered alphabet. *J. Algorithms*, 4:363–381, 1983.

[41] J.-P. Duval, T. Harju, and D. Nowotka. Unbordered factors and Lyndon words. *Discrete Math.*, 308:2261–2264, 2008.

[42] A. Ehrenfeucht and D. M. Silberger. Periodicity and unbordered segments of words. *Discrete Math.*, 26:101–109, 1979.

[43] I. Fagnot. Sur les facteurs des mots automatiques. *Theoret. Comput. Sci.*, 172:67–89, 1997.

[44] N. J. Fine and H. S. Wilf. Uniqueness theorems for periodic functions. *Proceedings of the American Mathematical Society*, 16(1):109–114, 1965.

[45] M. J. Fischer and M. O. Rabin. Super-exponential complexity of presburger arithmetic. In *Proceedings of the SIAM-AMS Symposium in Applied Mathematics*, volume 7, pages 27 – 41, 1974.

[46] A. Frid. Infinite permutations vs. infinite words. In P. Ambrož, S. Holub, and Z. Masáková, editors, *WORDS 2011, 8th International Conference*. Elect. Proc. Theor. Comput. Sci., 2011. Available at `http://arxiv.org/abs/1108.3616v1`.

[47] A. Frid and L. Q. Zamboni. On automatic infinite permutations. Presented at *Journées Montoises*, 2010.

[48] C. Frougny and B. Solomyak. On representation of integers in linear numeration systems. In M. Pollicott and K. Schmidt, editors, *Ergodic Theory of $\mathbb{Z}^d$ Actions (Warwick, 1993–1994)*, volume 228 of *London Mathematical Society Lecture Note Series*, pages 345–368. Cambridge University Press, 1996.

[49] E. Garel. Séparateurs dans les mots infinis engendrés par morphismes. *Theoret. Comput. Sci.*, 180:81–113, 1997.

[50] William Gasarch and James Glenn. Implementing ws1s via finite automata. In *In Automata Implementation, WIA '96, Proceedings, volume 1260 of LNCS*, pages 50–63. Springer-Verlag, 1997.

[51] A. Glen, F. Levé, and G Richomme. Quasiperiodic and lyndon episturmian words. *Theoretical Computer Science*, 409(3):578 – 600, 2008.

[52] James Glenn and William Gasarch. Implementing ws1s via finite automata: Performance issues. In Derick Wood and Sheng Yu, editors, *Automata Implementation*, volume 1436 of *Lecture Notes in Computer Science*, pages 75–86. Springer Berlin Heidelberg, 1998.

[53] I. Goldstein. Asymptotic subword complexity of fixed points of group substitutions. *Theoret. Comput. Sci.*, 410:2084–2098, 2009.

[54] I. Goldstein. Subword complexity of uniform D0L words over finite groups. *Theoret. Comput. Sci.*, 412:5728–5743, 2011.

[55] D. Goč, D. Henshall, and J. Shallit. Automatic theorem-proving in combinatorics on words. In N. Moreira and R. Reis, editors, *CIAA 2012*, volume 7381 of *Lecture Notes in Computer Science*, pages 180–191. Springer-Verlag, 2012.

[56] D. Goč, H. Mousavi, and J. Shallit. On the number of unbordered factors. In Adrian-Horia Dediu, Carlos MartÃŋn-Vide, and Bianca Truthe, editors, *Language and Automata Theory and Applications*, volume 7810 of *Lecture Notes in Computer Science*, pages 299–310. Springer Berlin Heidelberg, 2013.

[57] D. Goč, K. Saari, and J. Shallit. Primitive words and Lyndon words in automatic and linearly recurrent sequences. In *LATA 2013*, pages 311–322, 2013.

[58] D. Goč, L. Schaeffer, and J. Shallit. Subword complexity and k-synchronization. In Marie-Pierre Béal and Olivier Carton, editors, *Developments in Language Theory*, volume 7907 of *Lecture Notes in Computer Science*, pages 252–263. Springer Berlin Heidelberg, 2013.

[59] V. Halava, T. Harju, T. Kärki, and M. Rigo. On the periodicity of morphic words. In *Developments in Language Theory 2010*, volume 6224 of *Lecture Notes in Computer Science*, pages 209–217. Springer-Verlag, 2010.

[60] T. Harju and D. Nowotka. Periodicity and unbordered words: a proof of the extended Duval conjecture. *J. Assoc. Comput. Mach.*, 54:Article 20, 2007.

[61] S. Holub. A proof of the extended Duval's conjecture. *Theoret. Comput. Sci.*, 339:61–67, 2005.

[62] S. Holub and D. Nowotka. On the relation between periodicity and unbordered factors of finite words. *Internat. J. Found. Comp. Sci.*, 21:633–645, 2010.

[63] J. Honkala. A decision method for the recognizability of sets defined by number systems. *RAIRO Inform. Théor. App.*, 20:395–403, 1986.

[64] A. Ido and G. Melançon. Lyndon factorization of the Thue-Morse word and its relatives. *Discrete Math. & Theoret. Comput. Sci.*, 1:43–52, 1997.

[65] Veikko Keränen. Abelian squares are avoidable on 4 letters. In W. Kuich, editor, *Automata, Languages and Programming*, volume 623 of *Lecture Notes in Computer Science*, pages 41–52. Springer Berlin Heidelberg, 1992.

[66] D. Krieger and J. Shallit. Every real number greater than 1 is a critical exponent. *Theoret. Comput. Sci.*, 381:177–182, 2007.

[67] W. Kuich and A. Salomaa. *Semirings, Automata, Languages*. Springer-Verlag, 1986.

[68] Tetsuo Kurosaki. Direct definition of a ternary infinite square-free sequence. *Inf. Process. Lett.*, 106(5):175–179, May 2008.

[69] John Leech. A problem on strings of beads. *Math. Gaz.*, 41:277–278, 1957.

[70] J. Leroux. A polynomial time Presburger criterion and synthesis for number decision diagrams. In *20th IEEE Symposium on Logic in Computer Science (LICS 2005)*, pages 147–156. IEEE Press, 2005.

[71] F. Levé and G. Richomme. Quasiperiodic infinite words: Some answers. *Bulletin of the European Association for Theoretical Computer Science*, 84:170–174, October 2004.

[72] F. Levé and G. Richomme. Quasiperiodic sturmian words and morphisms. *Theoretical Computer Science*, 372(1):15 – 25, 2007.

[73] R. C. Lyndon and M. P. Schützenberger. The equation $a^M = b^N c^P$ in a free group. *Michigan Math. J.*, 9:289–298, 1962.

[74] S. Marcus. Quasiperiodic infinite words. *Bulletin of the European Association for Theoretical Computer Science*, 82:170–174, February 2004.

[75] G. Melançon. Lyndon factorization of infinite words. In C. Puech and R. Reischuk, editors, *STACS 96, 13th Annual Symposium on Theoretical Aspects of Computer Science*, volume 1046 of *Lecture Notes in Computer Science*, pages 147–154. Springer-Verlag, 1996.

[76] Robert Mercas, Pascal Ochem, Alexey V. Samsonov, and Arseny M. Shur. Binary patterns in binary cube-free words: Avoidability and growth. *CoRR*, abs/1301.4682, 2013.

[77] M. Morse and G. A. Hedlund. Symbolic dynamics. *American Journal of Mathematics*, 60(4):pp. 815–866, 1938.

[78] B. Mossé. Reconnaissabilité des substitutions et complexité des suites automatiques. *Bull. Soc. Math. France*, 124:329–346, 1996.

[79] F. Nicolas and Yu. Pritykin. On uniformly recurrent morphic sequences. *Internat. J. Found. Comp. Sci.*, 20:919–940, 2009.

[80] P. T. Nielsen. A note on bifix-free sequences. *IEEE Trans. Inform. Theory*, IT-19:704–706, 1973.

[81] D. Nowotka. *Periodicity and Unbordered Factors of Words*. PhD thesis, University of Turku, Finland, 2004.

[82] MojÅijesz Presburger and Dale Jacquette. On the completeness of a certain system of arithmetic of whole numbers in which addition occurs as the only operation. *History and Philosophy of Logic*, 12(2):225–233, 1991.

[83] H. Prodinger and F. J. Urbanek. Infinite 0–1-sequences without long adjacent identical blocks. *Discrete Math.*, 28:277–289, 1979.

[84] N. Rampersad, J. Shallit, and M.-w. Wang. Inverse star, borders, and palstars. *Inform. Process. Lett.*, 111:420–422, 2011.

[85] B. Reznick. Some binary partition functions. In *Analytic Number Theory*, volume 85 of *Progr. Math.*, pages 451–477. Birkhäuser, 1990.

[86] J. R. Roche. On stewart's choral sequence. Presented at the Mathematical Society of the Philippines 2008 Annual Convention, Diliman, Quezon City, Philippines, 2008.

[87] E. Rowland and J. Shallit. k-automatic sets of rational numbers. In A.-H. Dediu and C. Martin-Vide, editors, *Proc. LATA 2012*, volume 7183 of *Lecture Notes in Computer Science*, pages 490–501. Springer, 2012.

[88] E. Rowland and J. Shallit. k-automatic sets of rational numbers. In A. H. Dediu and C. Martín-Vide, editors, *LATA 2012 Proceedings*, volume 7183 of *Lecture Notes in Computer Science*, pages 490–501. Springer-Verlag, 2012.

[89] W. Rudin. Some theorems on Fourier coefficients. *Proc. Amer. Math. Soc.*, 10:855–859, 1959.

[90] K. Saari. *On the Frequency and Periodicity of Infinite Words*. PhD thesis, University of Turku, Finland, 2008.

[91] J. Sakarovitch. *Elements of Automata Theory*. Cambridge University Press, 2009.

[92] A. Salomaa and M. Soittola. *Automata-Theoretic Aspects of Formal Power Series*. Springer-Verlag, 1978.

[93] L. Schaeffer and J. Shallit. The critical exponent is computable for automatic sequences. *Int. J. Found. Comput. Sci.*, to appear, 2012.

[94] Luke Schaeffer. Ostrowski numeration and the local period of sturmian words. In *LATA*, pages 493–503, 2013.

[95] M.-P. Schützenberger. On a theorem of R. Jungen. *Proc. Amer. Math. Soc.*, 13:885–890, 1962.

[96] P. Séébold. Lyndon factorization of the Prouhet words. *Theoret. Comput. Sci.*, 307:179–197, 2003.

[97] J. Shallit. The critical exponent is computable for automatic sequences. In P. Ambroz, S. Holub, and Z. Másaková, editors, *Proceedings 8th International Conference Words 2011*, volume 63 of *Elect. Proc. Theor. Comput. Sci.*, pages 231–239. 2011.

[98] H. S. Shapiro. Extremal problems for polynomials and power series. Master's thesis, MIT, 1952.

[99] D. M. Silberger. Borders and roots of a word. *Portugal. Math.*, 30:191–199, 1971.

[100] D. M. Silberger. How many unbordered words? *Ann. Soc. Math. Polon. Ser. I: Comment. Math.*, 22:143–145, 1980.

[101] R. Siromoney, L. Mathew, V. Dare, and K. Subramanian. Infinite Lyndon words. *Inform. Process. Lett.*, 50:101–104, 1994.

[102] I. Stewart. Mathematical recreations: The never-ending chess game. *Scientific American*, 273(4):182–183, October 1995.

[103] A. Thue. Über unendliche Zeichenreihen. *Norske vid. Selsk. Skr. Mat. Nat. Kl.*, 7:1–22, 1906. Reprinted in *Selected Mathematical Papers of Axel Thue*, T. Nagell, editor, Universitetsforlaget, Oslo, 1977, pp. 139–158.

[104] A. Thue. Über die gegenseitige Lage gleicher Teile gewisser Zeichenreihen. *Norske vid. Selsk. Skr. Mat. Nat. Kl.*, 1:1–67, 1912. Reprinted in *Selected Mathematical Papers of Axel Thue*, T. Nagell, editor, Universitetsforlaget, Oslo, 1977, pp. 413–478.

[105] S. Widmer. Permutation complexity of the Thue-Morse word. *Adv. in Appl. Math.*, 47:309–329, 2011.

APPENDICES

Appendix A

Program Output

A.1 Quasi-periods and the Usual Suspects

Here is a summary of the computation:

```
Thue-Morse:
      [>= l j] (3 => 2 states) in 0.019s
      [> j+n l] (6 => 3 states) in 0.023s
     [\and ] (4 => 4 states) in 0.051s
         [>= i0 n] (3 => 2 states) in 0.029s
         [\out= i+i0 j+i0] (8 => 8 states) in 0.088s
        [\or ] (16 => 16 states) in 0.093s
       [\not ] (17 => 16 states) in 0.104s
      [\exists i0] (16 => 22 states) in 0.058s
     [\not ] (23 => 21 states) in 0.065s
    [\and ] (31 => 31 states) in 0.065s
   [\exists j] (31 => 116 states) in 0.060s
  [\not ] (117 => 116 states) in 0.126s
 [\exists l] (116 => 1 states) in 0.032s
[\not ] (2 => 1 states) in 0.041s
[total] (1 states) in 0.905s

Rudin-Shapiro:
      [>= l j] (3 => 2 states) in 0.031s
      [> j+n l] (6 => 3 states) in 0.024s
     [\and ] (4 => 4 states) in 0.055s
         [>= i0 n] (3 => 2 states) in 0.028s
         [\out= i+i0 j+i0] (28 => 28 states) in 0.213s
        [\or ] (56 => 56 states) in 0.217s
       [\not ] (57 => 56 states) in 0.232s
      [\exists i0] (56 => 99 states) in 1.170s
     [\not ] (100 => 98 states) in 0.165s
    [\and ] (152 => 152 states) in 0.129s
```

```
   [\exists j] (152 => 294 states) in 0.166s
  [\not ] (295 => 294 states) in 0.267s
 [\exists l] (294 => 1 states) in 0.055s
[\not ] (2 => 1 states) in 0.037s
[total] (1 states) in 2.910s

Paperfolding:
      [>= l j] (3 => 2 states) in 0.027s
      [> j+n l] (6 => 3 states) in 0.027s
     [\and ] (4 => 4 states) in 0.043s
         [>= i0 n] (3 => 2 states) in 0.028s
         [\out= i+i0 j+i0] (33 => 15 states) in 0.205s
        [\or ] (31 => 31 states) in 0.140s
       [\not ] (32 => 30 states) in 0.140s
      [\exists i0] (30 => 61 states) in 0.209s
     [\not ] (62 => 60 states) in 0.113s
    [\and ] (96 => 96 states) in 0.081s
   [\exists j] (96 => 135 states) in 0.077s
  [\not ] (136 => 135 states) in 0.164s
 [\exists l] (135 => 1 states) in 0.044s
[\not ] (2 => 1 states) in 0.035s
[total] (1 states) in 1.401s

Period-doubling:
      [>= l j] (3 => 2 states) in 0.023s
      [> j+n l] (6 => 3 states) in 0.024s
     [\and ] (4 => 4 states) in 0.046s
         [>= i0 n] (3 => 2 states) in 0.025s
         [\out= i+i0 j+i0] (13 => 6 states) in 0.141s
        [\or ] (13 => 13 states) in 0.079s
       [\not ] (14 => 12 states) in 0.080s
      [\exists i0] (12 => 11 states) in 0.039s
     [\not ] (12 => 10 states) in 0.057s
    [\and ] (20 => 20 states) in 0.051s
   [\exists j] (20 => 33 states) in 0.033s
  [\not ] (34 => 33 states) in 0.070s
 [\exists l] (33 => 1 states) in 0.026s
[\not ] (2 => 1 states) in 0.045s
[total] (1 states) in 0.774s

Stewart choral:
      [>= l j] (3 => 2 states) in 0.038s
      [> j+n l] (6 => 3 states) in 0.050s
     [\and ] (4 => 4 states) in 0.076s
         [>= i0 n] (3 => 2 states) in 0.066s
         [\out= i+i0 j+i0] (22 => 11 states) in 0.611s
        [\or ] (23 => 23 states) in 0.635s
       [\not ] (24 => 22 states) in 0.604s
      [\exists i0] (22 => 14 states) in 0.213s
     [\not ] (15 => 13 states) in 0.133s
    [\and ] (29 => 29 states) in 0.134s
   [\exists j] (29 => 49 states) in 0.106s
  [\not ] (50 => 49 states) in 0.160s
 [\exists l] (49 => 1 states) in 0.033s
[\not ] (2 => 1 states) in 0.047s
[total] (1 states) in 3.098s
```

```
Mephisto waltz:
      [>= l j] (3 => 2 states) in 0.036s
      [> j+n l] (6 => 3 states) in 0.045s
     [\and ] (4 => 4 states) in 0.079s
         [>= i0 n] (3 => 2 states) in 0.076s
         [\out= i+i0 j+i0] (8 => 8 states) in 0.360s
       [\or ] (16 => 16 states) in 0.482s
      [\not ] (17 => 16 states) in 0.485s
     [\exists i0] (16 => 20 states) in 0.239s
    [\not ] (21 => 19 states) in 0.171s
   [\and ] (29 => 29 states) in 0.112s
  [\exists j] (29 => 66 states) in 0.097s
 [\not ] (67 => 66 states) in 0.197s
[\exists l] (66 => 1 states) in 0.041s
[\not ] (2 => 1 states) in 0.044s
[total] (1 states) in 2.634s
```

A.2 Search for Quasi-periods

Here is a summary of the computation:

```
0->0000, 1->0000         is periodic. in 0.261s
0->0000, 1->0001         is periodic. in 0.231s
0->0000, 1->0010         is periodic. in 0.245s
0->0000, 1->0011         is periodic. in 0.265s
0->0000, 1->0100         is periodic. in 0.242s
0->0000, 1->0101         is periodic. in 0.258s
0->0000, 1->0110        is periodic. in 0.245s
0->0000, 1->0111         is periodic. in 0.239s
0->0000, 1->1000         is periodic. in 0.259s
0->0000, 1->1001         is periodic. in 0.248s
0->0000, 1->1010         is periodic. in 0.258s
0->0000, 1->1011         is periodic. in 0.230s
0->0000, 1->1100         is periodic. in 0.229s
0->0000, 1->1101         is periodic. in 0.271s
0->0000, 1->1110         is periodic. in 0.246s
0->0000, 1->1111         is periodic. in 0.246s
0->0001, 1->0000
0->0001, 1->0001         is periodic. in 0.300s
0->0001, 1->0010
0->0001, 1->0011
0->0001, 1->0100
0->0001, 1->0101
0->0001, 1->0110
0->0001, 1->0111
0->0001, 1->1000
0->0001, 1->1001
0->0001, 1->1010
0->0001, 1->1011
0->0001, 1->1100
0->0001, 1->1101
```

```
0->0001, 1->1110
0->0001, 1->1111
0->0010, 1->0000
0->0010, 1->0001        is quasi-periodic. in 21.769s
0->0010, 1->0010         is periodic. in 0.352s
0->0010, 1->0011
0->0010, 1->0100        is quasi-periodic. in 24.125s
0->0010, 1->0101
0->0010, 1->0110
0->0010, 1->0111
0->0010, 1->1000
0->0010, 1->1001
0->0010, 1->1010
0->0010, 1->1011
0->0010, 1->1100
0->0010, 1->1101
0->0010, 1->1110
0->0010, 1->1111
0->0011, 1->0000
0->0011, 1->0001
0->0011, 1->0010
0->0011, 1->0011         is periodic. in 0.339s
0->0011, 1->0100
0->0011, 1->0101
0->0011, 1->0110
0->0011, 1->0111
0->0011, 1->1000
0->0011, 1->1001
0->0011, 1->1010
0->0011, 1->1011
0->0011, 1->1100
0->0011, 1->1101
0->0011, 1->1110
0->0011, 1->1111
0->0100, 1->0000
0->0100, 1->0001
0->0100, 1->0010
0->0100, 1->0011
0->0100, 1->0100         is periodic. in 0.353s
0->0100, 1->0101
0->0100, 1->0110
0->0100, 1->0111
0->0100, 1->1000
0->0100, 1->1001
0->0100, 1->1010
0->0100, 1->1011
0->0100, 1->1100
0->0100, 1->1101
0->0100, 1->1110
0->0100, 1->1111
0->0101, 1->0000
0->0101, 1->0001
0->0101, 1->0010        is quasi-periodic. in 26.644s
0->0101, 1->0011
0->0101, 1->0100
0->0101, 1->0101         is periodic. in 0.360s
0->0101, 1->0110
```

```
0->0101, 1->0111
0->0101, 1->1000
0->0101, 1->1001
0->0101, 1->1010
0->0101, 1->1011
0->0101, 1->1100
0->0101, 1->1101
0->0101, 1->1110
0->0101, 1->1111
0->0110, 1->0000
0->0110, 1->0001
0->0110, 1->0010
0->0110, 1->0011
0->0110, 1->0100
0->0110, 1->0101
0->0110, 1->0110         is periodic. in 0.376s
0->0110, 1->0111
0->0110, 1->1000
0->0110, 1->1001
0->0110, 1->1010
0->0110, 1->1011
0->0110, 1->1100
0->0110, 1->1101
0->0110, 1->1110
0->0110, 1->1111
0->0111, 1->0000
0->0111, 1->0001
0->0111, 1->0010
0->0111, 1->0011
0->0111, 1->0100
0->0111, 1->0101
0->0111, 1->0110
0->0111, 1->0111         is periodic. in 0.358s
0->0111, 1->1000
0->0111, 1->1001
0->0111, 1->1010
0->0111, 1->1011
0->0111, 1->1100
0->0111, 1->1101
0->0111, 1->1110
0->0111, 1->1111
[total] in 35m46.713s
```
