

State Complexity of Linear Relations and Linear Subsequences of Automatic Sequences

by

Delaram Moradi

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, 2026

© Delaram Moradi 2026

Author's Declaration

This thesis consists of material all of which I authored or co-authored: see Statement of Contributions included in the 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.

Statement of Contributions

This thesis includes content taken nearly verbatim or verbatim from the arXiv paper [35] co-authored with Narad Rampersad and my supervisor Jeffrey Shallit and the arXiv paper [34] co-authored with Narad Rampersad and my supervisor Jeffrey Shallit. Content from [35] is used in chapters 1, 2, and 4. Content from [34] is used in chapters 1, 3, and 4. Some results in these papers are mostly due to Narad Rampersad. The initial idea of Theorem 41 and the proof of Theorem 15, Corollary 17, and Theorem 24 were mostly due to Narad Rampersad. Moreover, Narad Rampersad introduced some references and helped in finding some errors.

Abstract

In this thesis, we study the state complexity of specific formal languages; for example, we study the number of states required in the minimal automaton reading the representation of two integers i, j in parallel and accepting them if and only if $i + c = j$ for some constant integer $c \geq 1$. We also study the state complexity of linear subsequences of automatic sequences; for example, we study the number of states required in the minimal automaton generating the linear subsequence $(h(i + c))_{i \geq 0}$ for some automatic sequences $(h(i))_{i \geq 0}$ and some constant integer $c \geq 1$. Moreover, we study the runtime complexity of generating automata for specific formal languages and linear subsequences of automatic sequences using a reasonable interpretation of Büchi arithmetic; for example we study the runtime complexity of creating an automaton reading the representation of two integers i, j in parallel and accepting them if and only if $ni = j$ for some constant $n \geq 2$. We also state some open problems.

The above topics are studied both for automata with input in base- k representation for some integer $k \geq 2$ and for automata with input in Fibonacci representation. Most results are for automata reading input in most-significant-digit-first format and some results are stated for automata reading input in least-significant-digit-first format.

Acknowledgements

I would like to thank Jeffrey Shallit for supervising me; I would like to thank Narad Rampersad and Jeffrey Shallit for co-authoring with me; and I would like to thank Kevin Hare and Lila Kari for agreeing to read my thesis.

Table of Contents

Author's Declaration	ii
Statement of Contributions	iii
Abstract	iv
Acknowledgements	v
List of Figures	viii
List of Tables	x
1 Introduction	1
1.1 Motivation	1
1.2 Preliminaries	2
2 Automata With Input in Base k	11
2.1 Introduction	11
2.2 Background	12
2.3 Recognizing Relations	13
2.3.1 Addition	13
2.3.2 Multiplication	21

2.4	Linear Subsequences of Automatic Sequences	22
2.4.1	Connection with Subword Complexity	28
2.4.2	Thue-Morse Sequence	29
2.4.3	Sequences Whose Shifts Have Small State Complexity	33
2.5	Construction by Büchi Arithmetic	35
2.6	Conclusion	44
3	Automata With Input in Fibonacci Representation	45
3.1	Introduction	45
3.2	Background	46
3.2.1	Finite Automata	46
3.2.2	Fibonacci Representation	47
3.3	Recognizing Relations	50
3.3.1	Addition	50
3.3.2	Multiplication	53
3.4	Linear Subsequences	57
3.4.1	Application to a Problem of Bosma and Don	61
3.5	Construction by Büchi Arithmetic	62
3.6	Conclusion	71
4	Conclusion	72
4.1	Conclusion	72
4.2	More Open Problems	72
	References	74

List of Figures

1.1	Transition diagram for a DFA M accepting L_e .	4
1.2	Transition diagram for an NFA M accepting $L = \{10, 101\}$.	6
1.3	Transition diagram for a DFAO M that outputs the number of 1s read modulo 3 for input in base 2.	7
1.4	An automaton with base-2 input generating the Thue-Morse sequence.	8
2.1	An automaton with msd-first input accepting $x \times y \times z$ if and only if $[x]_k + [y]_k = [z]_k$.	14
2.2	An automaton with lsd-first input accepting $x \times y \times z$ if and only if $[x]_k + [y]_k = [z]_k$.	15
2.3	An automaton with msd-first base-3 input accepting $x \times y$ if and only if $[x]_3 + 7 = [y]_3$.	18
2.4	An automaton with msd-first base-3 input accepting $x \times y$ if and only if $[x]_3 + 6 = [y]_3$.	18
2.5	An automaton with msd-first base-3 input accepting $x \times y$ if and only if $[x]_3 + 4 = [y]_3$.	19
2.6	An automaton with msd-first base-2 input accepting $x \times y$ if and only if $[x]_2 + 5 = [y]_2$.	19
2.7	A double-state followed by a single-state in an automaton accepting $x \times y$ if and only if $[x]_k + c = [y]_k$.	20
2.8	An automaton with msd-first base-2 input accepting $x \times y$ if and only if $3[x]_2 + 1 = [y]_2$.	22
2.9	An automaton with base-2 lsd-first input generating a 2-automatic sequence $(h(i))_{i \geq 0}$.	24

2.10	An automaton with base-2 input generating the Thue-Morse sequence. . .	29
2.11	An automaton with msd-first base-2 input generating $(t(i + 1))_{i \geq 0}$	30
2.12	An automaton with msd-first base-2 input generating $(P_2(i + 10))_{i \geq 0}$	36
3.1	An automaton generating the Fibonacci word $\mathbf{f}$	48
3.2	Part of an automaton accepting $x \times y$ if and only if $[x] + 40 = [y]$	52
3.3	An automaton with input in Fibonacci representation accepting $x \times y$ if and only if $2[x] = [y]$	54
3.4	An automaton with input in Fibonacci representation generating the Thue-Morse sequence.	58
3.5	An automaton with input in Fibonacci representation generating $\mathbf{f}[2i]$. . .	61

List of Tables

3.1	The canonical Fibonacci representation of integers 0 to 10.	47
-----	---	----

Chapter 1

Introduction

This chapter includes content taken nearly verbatim or verbatim from the arXiv paper [35] co-authored with Narad Rampersad and Jeffrey Shallit and the arXiv paper [34] co-authored with Narad Rampersad and Jeffrey Shallit.

1.1 Motivation

The state complexity of a formal language L , which is the number of states in the minimal deterministic automaton accepting L , has been studied for a long time; for example by Yu et al. [49], Alexeev [1], and Sutner [44]. The reader can refer to a survey by Gao et al. [20] to learn more about some known results on the topic. This thesis includes discussions on the state complexity of specific languages; for example, in Theorem 8 we find an exact formula (as a function of c) for the state complexity of the language of integer pairs i, j represented in base k where $i + c = j$ and the automaton's input is in most-significant-digit-first format.

Similarly, the state complexity of an automatic sequence, which is the number of states in the minimal deterministic automaton with output generating it, has been studied; for example by Zantema [50], Zantema and Bosma [51], and van Spaendonck [46]. This thesis includes discussions on the state complexity of specific automatic sequences; for example, in Theorem 11 we introduce a family of deterministic finite automata with output generating a k -automatic sequence $(h(i))_{i \geq 0}$ with m states such that the state complexity of the k -automatic sequence $(h(i + 1))_{i \geq 0}$ is at least $2m - 1$ where the automata input are in the least-significant-digit-first format.

The results on state complexity of languages and automatic sequences in this thesis expand on previously known results and answer open questions.

There are software packages, such as **Walnut** [36, 42], that implement a reasonable interpretation of Büchi arithmetic and can algorithmically create automata from logical statements in that theory. Büchi arithmetic is the logical theory of the natural numbers together with addition and the function $V_k(n)$, the largest power of k dividing n , for some fixed $k \geq 2$. This theory is powerful enough to express finite automata [12]. Users of **Walnut** want to know how big the resulting automata are and how long does it take to create them. In order to obtain the runtime complexity, we have to examine different considerations besides state complexity because in constructions by Büchi arithmetic, the intermediate automata can be larger than the resulting minimized automaton; for example, consider the following **Walnut** command from the paper by Khodier et al. [27] where **TR** is the Tribonacci word, the sequence [A080843](#) in the On-Line Encyclopedia of Integer Sequences (OEIS) [25].

```
def trib_eqfac "?msd_trib At (t<n) => TR[i+t]=TR[j+t]":
```

The resulting automaton of this command is one that accepts integers i, j, n if and only if the Tribonacci word starting from index i to $i + n - 1$ is the same as the Tribonacci word starting from index j to $j + n - 1$. The resulting automaton by **Walnut** has 26 states while the largest intermediate automaton has 323,831,403 states. In particular, to obtain the runtime complexity of constructing an automaton by a reasonable interpretation of Büchi arithmetic we need to consider (a) how an expression in Büchi arithmetic is translated to an automaton, (b) what are the sizes of the intermediate automata created in this translation, and (c) what the total running time of the procedure is. This thesis includes discussions on the runtime complexity of computing automata for specific languages and automatic sequences using a reasonable interpretation of Büchi arithmetic.

In this thesis, we study the above topics for automata with input in base- k numeration systems in Chapter 2 and for automata with input in the Fibonacci numeration system in Chapter 3.

1.2 Preliminaries

The concepts introduced in this section are preliminaries for the rest of the thesis.

There are many textbooks on formal languages and automata theory; for example, textbooks written by Martin [32], Sipser [43], and Shallit [40]. We start by reviewing some well-known basics of formal languages and automata theory.

Consider a simple problem with some input and a “yes” or “no” output; for example, “Is x the binary representation of an even number?”. We can create a set consisting of all the inputs that result in the output “yes” for the problem. If what matters about a problem is its output for a specific input, this set captures the problem in a mathematical sense and the set is called a *formal language* or simply a *language*. For example, the language corresponding to the problem above is $\{\lambda, 0, 00, 10, 000, 010, 100, 110, \dots\}$ where λ corresponds to the empty input and is a representation of 0; we call this language L_e . Representing problems by a language allows us to study them; this is not easily possible if the problem is expressed in natural language.

Each element in the language is a *word* or *string* and each word is made of *letters* or *symbols*; for example, 10 is a word made of letters 1 and 0. More generally, we can define a set of letters Σ and any combination of the letters is a word. We use Σ^l , where $l \geq 0$, to denote the set of length- l words made from letters in Σ ; for example, for $\Sigma = \{0, 1\}$, we have $\Sigma^2 = \{00, 01, 10, 11\}$. We use Σ^* to denote the set of all words consisting of letters in Σ ; this includes the length-0 empty word λ .

We know we can design algorithms to solve a problem. The algorithm takes some input and returns the solution to the problem. Similarly, if we define abstract models of computation, we can design instances of the model that take some input and output the solution to the problem. The purpose of defining abstract models of computation is the same as defining the concept of formal languages; they allow us to study problems.

One simple model of computation is *deterministic finite automaton* or *DFA*. A DFA gets some input and either accepts or rejects the input. A DFA has a finite number of *states*. At the start of the computation, the DFA is in its *initial state*. The DFA gets an input word, letter by letter. After reading each letter, based on its current state and the letter read, the *transition function* determines which state the automaton is in next. A subset of the states are the *accepting states*. If on some input, the automaton ends in an accepting state, the automaton accepts that input; otherwise, it rejects that input. The set of words accepted by a DFA is the language it accepts. Recall that a problem with “yes” or “no” output can be represented by a language. So a specific DFA can be designed to accept the language corresponding to a problem with “yes” or “no” output.

More formally, a DFA is a 5-tuple $M = (Q, \Sigma, \delta, q_0, A)$ where

- Q is a finite non-empty set of states;

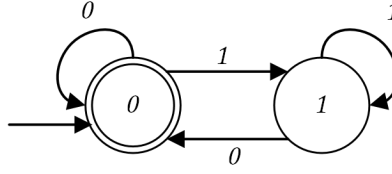


Figure 1.1: Transition diagram for a DFA M accepting L_e .

- Σ is the input alphabet;
- $\delta : Q \times \Sigma \rightarrow Q$ is the transition function;
- q_0 is the initial state;
- $A \subseteq Q$ is the set of accepting states.

For example, the DFA $M = (Q, \Sigma, \delta, q_0, A)$ reading the input starting from the most significant digit first is defined formally as follows and accepts the language L_e .

$$\begin{aligned}
 Q &= \{0, 1\}, \\
 \Sigma &= \{0, 1\}, \\
 \delta(q, a) &= a, \\
 q_0 &= 0, \\
 A &= \{0\}.
 \end{aligned}$$

A DFA can be represented by a diagram where states are circles, transitions are labeled arrows connecting circles, and accepting states are double circles; for example, Figure 1.1 is the diagram for the DFA M defined above.

The definition of the transition function δ only needs to include the case of one state and one letter as input; but we can recursively extend the notion of δ so it gets one state and a word as input as well. If we have $x' = xa$, then $\delta(q, x') = \delta(\delta(q, x), a)$. This recursive definition extends the domain of δ from $Q \times \Sigma$ to $Q \times \Sigma^*$.

Another simple model of computation is *non-deterministic finite automaton* or *NFA*. An NFA is very similar to a DFA. One difference is if the automaton is in some state and reads a letter it can transition to multiple states and the other difference is the transition function is not necessarily defined for all combinations of states and alphabet; in other words, the transition function is defined on sets of states instead of individual states. The

input to an NFA is accepted by the NFA if the set of states the automaton reaches on it includes some accepting state(s). Sometimes transitions on input λ are allowed in NFAs, but in this thesis we do not need them.

More formally, an NFA is a 5-tuple $M = (Q, \Sigma, \delta, q_0, A)$ where

- Q is a finite non-empty set of states;
- Σ is the input alphabet;
- $\delta : Q \times \Sigma \rightarrow 2^Q$ is the transition function;
- q_0 is the initial state;
- $A \subseteq Q$ is the set of accepting states.

For example, the NFA $M = (Q, \Sigma, \delta, q_0, A)$ reading the input starting from the most significant digit first is defined formally as follows and accepts the language $L = \{10, 101\}$.

$$\begin{aligned}
 Q &= \{0, 1, 2, 3, 4, 5\}, \\
 \Sigma &= \{0, 1\}, \\
 \delta(q, a) &= \begin{cases} \{1, 3\}, & \text{if } q = 0, a = 1; \\ \{2\}, & \text{if } q = 1, a = 0; \\ \{4\}, & \text{if } q = 3, a = 0; \\ \{5\}, & \text{if } q = 4, a = 1; \end{cases} \\
 q_0 &= 0, \\
 A &= \{2, 5\}.
 \end{aligned}$$

Similar to DFAs, an NFA can be represented by a diagram where states are circles, transitions are labeled arrows connecting circles, and accepting states are double circles; for example, Figure 1.2 is the diagram for the NFA M defined above.

Similar to the case of DFAs, the transition function of an NFA can be extended. First we show how it can be extended to a set of states. If we have $Q' = \{q_0, \dots, q_n\}$, then $\delta(Q', a) = \delta(q_0, a) \cup \dots \cup \delta(q_n, a)$. Next we show how the input of δ can be extended to a word. If we have $x' = xa$, then $\delta(q, x') = \delta(\delta(q, x), a)$ and $\delta(Q', x') = \delta(\delta(Q', x), a)$. This recursive definition extends the domain of δ from $Q \times \Sigma$ to $2^Q \times \Sigma^*$. An input x is accepted by an NFA with transition function δ and initial state q_0 if there is $q \in A$ such that $q \in \delta(q_0, x)$.

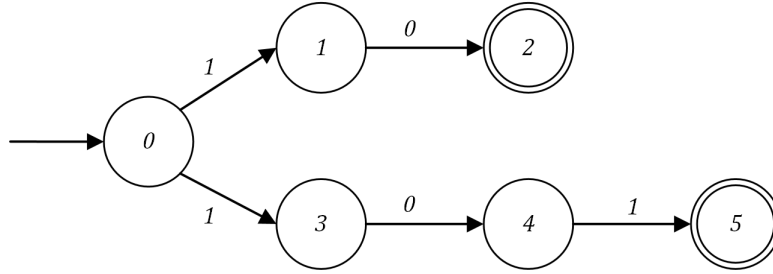


Figure 1.2: Transition diagram for an NFA M accepting $L = \{10, 101\}$.

A state q in a DFA or NFA is called *dead* if it is not possible to reach any accepting states from q by a (possibly empty) path. A minimal DFA or NFA for a language has at most one dead state. By convention, we allow dead states to not be counted or displayed. Usually, the transition function δ of a DFA is taken to be a total function, but we allow it to be a partial function in order to gracefully handle dead states. A state is called *accessible* if it is reachable from the start state, and *co-accessible* if one can reach an accepting state from it.

In a DFA or NFA with initial state q_0 , We say the two states q_1 and q_2 are *equivalent* if for all x, y, z where $\delta(q_0, x) = q_1$ and $\delta(q_0, y) = q_2$, $\delta(q_0, xz)$ and $\delta(q_0, yz)$ are either both accepting or both rejecting. We say a DFA or NFA is minimal if there are no two equivalent states in it.

DFA's and NFA's only have binary output, accept or reject, “yes” or “no”. They are used for recognizing whether a word belongs to a language and studying problems with “yes” or “no” output.

Another model of computation is *deterministic finite automaton with output* or *DFAO*. DFAOs work similar to DFA's in regards to states and transitions; the important difference is that DFAOs can have more than two possible outputs. In DFA's, states are partitioned into accepting and rejecting states. In DFAOs, an *output function* is defined that assigns an output to each state.

More formally, a DFAO is a 6-tuple $M = (Q, \Sigma, \delta, q_0, \Delta, \tau)$ where

- Q is a finite non-empty set of states;
- Σ is the input alphabet;
- $\delta : Q \times \Sigma \rightarrow Q$ is the transition function;

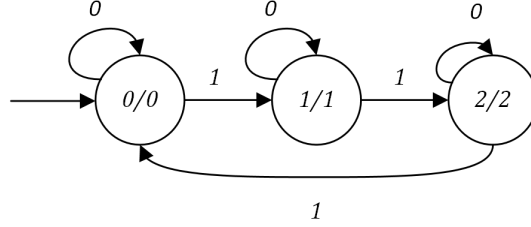


Figure 1.3: Transition diagram for a DFAO M that outputs the number of 1s read modulo 3 for input in base 2.

- q_0 is the initial state;
- Δ is the finite non-empty output alphabet; and
- $\tau : Q \rightarrow \Delta$ is the output function.

For example, the DFAO $M = (Q, \Sigma, \delta, q_0, \Delta, \tau)$ reading the input starting from the most significant digit first is defined formally as follows and outputs the number of 1s read modulo 3 for input in base 2.

$$\begin{aligned}
 Q &= \{0, 1, 2\}, \\
 \Sigma &= \{0, 1\}, \\
 \delta(q, a) &= (q + a) \bmod 3, \\
 q_0 &= 0, \\
 \Delta &= \{0, 1, 2\}, \\
 \tau(q) &= q.
 \end{aligned}$$

Similar to DFAs and NFAs, a DFAO can be represented by a diagram where states are circles, transitions are labeled arrows connecting circles, and each state's output is written in its circle next to the state's name; for example, Figure 1.3 is the diagram for the DFAO M defined above.

Similar to DFAs and NFAs, the transition function of a DFAO only needs to be defined with the input of a state and a letter, but we can recursively extend the notion of the transition function to the input of a state and a word. If we have $x' = xa$, then $\delta(q, x') = \delta(\delta(q, x), a)$.

As mentioned by van Spaendonck [46], the notion of state equivalence can be defined for DFAOs as well. In a DFAO with initial state q_0 , we say the two states q_1 and q_2 are

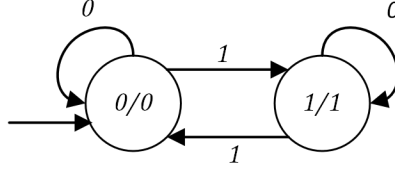


Figure 1.4: An automaton with base-2 input generating the Thue-Morse sequence.

equivalent if for all x, y, z where $\delta(q_0, x) = q_1$ and $\delta(q_0, y) = q_2$, we have $\tau(\delta(q_0, xz)) = \tau(\delta(q_0, yz))$. Similarly, a DFAO is minimal if there are no two equivalent states in it.

Now we introduce *sequences*. We show a sequence by the notation $(h(i))_{i \geq 0}$ where $h(i)$ is the i -th letter in the sequence starting from index 0. So $(h(i))_{i \geq 0} = h(0)h(1)h(2)\dots$. For example, the Thue-Morse sequence is $(t(i))_{i \geq 0} = t(0)t(1)\dots = 01101001\dots$ where $t(i)$ is the number of 1s in base-2 representation of i modulo 2. The Thue-Morse sequence is the sequence [A010060](#) of the On-Line Encyclopedia of Integer Sequences (OEIS) [25]. You can refer to the book by Allouche and Shallit [3] to read more about automatic sequences.

Fix a specific DFAO named M . We can enumerate all natural numbers, set their representation as input to M , and write the output letters one after another; the result is a sequence. We call sequences generated by a DFAO in this manner an *automatic sequence*. More formally, a sequence $(h(i))_{i \geq 0}$ is generated by a DFAO $M = (Q, \Sigma, \delta, q_0, \Delta, \tau)$, if and only if for all x and i where x is a representation of i , we have $\tau(\delta(q_0, x)) = h(i)$. For example, the DFAO in Figure 1.4 generates the Thue-Morse sequence.

For a given automatic sequence $(h(i))_{i \geq 0}$ there is a unique associated *interior sequence* $(h'(i))_{i \geq 0}$ with output values from the states of the minimal DFAO generating $(h(i))_{i \geq 0}$, defined by taking the minimal DFAO $(Q, \Sigma, \delta, q_0, \Delta, \tau)$ and replacing τ by the identity map i on Q . Here “unique” means up to renaming of the letters. It follows that $(h(i))_{i \geq 0}$ is the image of $(h'(i))_{i \geq 0}$ under the coding $q \rightarrow \tau(q)$.

For a DFAO $M = (Q, \Sigma, \delta, q_0, \Delta, \tau)$, let the DFAO $\hat{M} = (Q, \Sigma, \delta, q_0, Q, i)$, where $i : Q \rightarrow Q$ is the identity map, be the *intrinsic* DFAO of M , and let $\hat{\mathbf{M}}$ be the sequence generated by the intrinsic DFAO $\hat{M}$. The interior sequence $(h'(i))_{i \geq 0}$ of an automatic sequence $(h(i))_{i \geq 0}$ is the sequence generated by the intrinsic DFAO of the minimal DFAO for $(h(i))_{i \geq 0}$.

The *state complexity* of a language is the number of states in a minimal DFA recognizing it. We say the *state complexity* of an automatic sequence is the number of states in the minimal DFAO generating it.

Remark 1. We sometimes provide complexity bounds using big- O notation, and sometimes these bounds involve several variables. For these bounds we use the convention that the bounds hold for all sufficiently large values of all the variables.

Remark 2. We occasionally refer to $O(\log i)$ where i can be 0 or 1. In these cases, we adopt the convention that $O(\log i) = O(1)$ for $i \in \{0, 1\}$.

Remark 3. Sometimes the time complexity bounds mentioned include the addition of $O(\log i)$ or $O(i^j)$ components for some constant i ; these addition components are not significant compared to multiplication in $O(\log i)$ or $O(i^j)$ components for some constant i .

A *subword* or *factor* of a word is a contiguous block of letters within that word; for example, 11 is a subword of 01110. If $\mathbf{x}$ is an infinite word, the number of distinct length- n subwords present in $\mathbf{x}$ is denoted by the *subword complexity* function $\rho_{\mathbf{x}}(n)$.

The input to an automaton is a word consisting of zero or more letters. In this thesis, the input word to an automaton is a representation of a non-negative integer in a specific numeration system; for example, the word 01110 is a base-2 representation of the integer 14. The *canonical representation* of an integer in a numeration system is its representation in that numeration system with no leading zeros; for example, the canonical representation of 14 in base 2 is 1110 and the canonical representation of 0 in base 2 is λ .

We can specify the order the automaton input is read; *msd-first* stands for most significant digit first and *lsd-first* stands for least significant digit first. We design automata such that they have the correct behavior or output no matter how many leading zeros there are read in the input.

Sometimes, an automaton reads more than one input word. In this case, the input words are read in parallel; in other words, the automaton transitions from one state to another by reading a letter from each word. More formally, for input words $x = a_0 \cdots a_l$ and $y = b_0 \cdots b_l$ over the alphabet Σ , an automaton reading two input words in msd-first format, reads $[a_0, b_0] \cdots [a_l, b_l]$ over the alphabet $\Sigma \times \Sigma$ and we say the input to the automaton is $x \times y$; these notions are extended, in a similar fashion, to automata that read more than two input words or read input in lsd-first format. For example, if an automaton reads three words in parallel in msd-first format and the input words are 1010, 0001, and 1110, then the input to the automaton is $[1, 0, 1][0, 0, 1][1, 0, 1][0, 1, 0]$.

We need to make sure the final automata always have the correct output regardless of the number of leading zeros read; this is accomplished as follows. We always create automata such that they have a self-loop on $0 \times \cdots \times 0$. To do so in an automaton M with initial state q_0 and transition function δ , we change the initial state to the state q'_0 where

$\delta(q_0, [0^j \times \cdots \times 0^j]) = q'_0$ for some j and $\delta(q'_0, [0 \times \cdots \times 0]) = q'_0$. The rationale is that our automata should always have the same correct output regardless of the number of leading zeros read and by creating the automaton in this manner we can always assume we have considered enough leading zeros for the automaton computations to be correct. A similar idea for base-2 is mentioned by Wolper and Boigelot [48].

Chapter 2

Automata With Input in Base k

This chapter includes content taken nearly verbatim or verbatim from the arXiv paper [35] co-authored with Narad Rampersad and Jeffrey Shallit.

2.1 Introduction

This chapter includes discussions and results on the state complexity of specific languages and automatic sequences with input in base k .

The k -automatic sequences for integers $k \geq 2$ are an interesting class of sequences introduced by Cobham [17, 18] and has been studied for more than 50 years; see [3] for a monograph on automatic sequences. A sequence $(h(i))_{i \geq 0}$ is said to be k -automatic if there is a deterministic finite automaton with output (DFAO) computing the i -th term, given the base- k representation of i as input. A DFAO with base- k input is called a k -DFAO. For example, the 2-DFAO in Figure 1.3 generates the 2-automatic sequence 011212201...

If $x = e_{i-1} \cdots e_0$ is a finite word, then $[x]_k$ denotes the integer value of x when considered as a base- k expansion; that is,

$$[x]_k = \sum_{0 \leq j < i} e_j k^j.$$

Thus, for example, $[101011]_2 = 43$ for the most-significant-digit-first (msd-first) format.

A classical result of Cobham [18, p. 174] states that if $(h(i))_{i \geq 0}$ is k -automatic, then so is the linear subsequence $(h(ni + c))_{i \geq 0}$ for integers $n \geq 1$ and $c \geq 0$. In this chapter, we are interested in the state complexity of linear subsequences of automatic sequences

and related topics. Our results on the state complexity of linear subsequences expand on the paper by Zantema and Bosma [51] and we answer one of their open questions. More details can be found in Section 2.4.

One can also consider automata where the least significant digit is read first. For least-significant-digit first (lsd-first) input, Boudet and Comon [10] and for msd-first input, Wolper and Boigelot [48] have provided constructions for base-2 automata recognizing arithmetical relations. Their constructions relate to some of our results in Section 2.3; however, our results are for all bases k and for specific relations.

At first glance, it might seem that results for the state complexity of problems involving msd-first and lsd-first representations would be closely related. While this is true for arithmetic, it is not true for subsequences of automatic sequences. Roughly speaking, this is because a deterministic finite-state transducer can carry out addition and multiplication by constants on lsd-first representations, but not msd-first representations.

Boudet and Comon [10] and Wolper and Boigelot [48] also discussed the size of the automaton for an arbitrary formula in Presburger arithmetic. Furthermore, Amat et al. [4] have developed a tool named **Patapsco** that computes a lower bound for the number of states in the automaton for a given Presburger arithmetic formula; this tool computes the lower bound without actually constructing the automaton.

In Section 2.2, we introduce the required background. The rest of the chapter is dedicated to discussing our results and their relevance to previously known results. In Section 2.3, we discuss results regarding the state complexity of recognizing additive and multiplicative relations with base- k input. In Section 2.4, we study the state complexity of linear subsequences of a k -automatic sequence, and find a novel relationship between this state complexity and subword complexity. We also solve an open problem of Zantema and Bosma. This section contains our main novel results. In Section 2.5, we analyze the runtime complexity of constructing some of the automata studied in previous sections using an interpretation of Büchi arithmetic. Finally, in Section 2.6, we conclude this chapter.

2.2 Background

For the remainder of this chapter, we assume that the inputs to automata are represented in base k and the automatic sequences are k -automatic for some constant integer $k \geq 2$. Furthermore, we use Σ_k for the set $\{0, \dots, k-1\}$.

The *canonical base- k representation* of an integer i is the word x , without leading zeros, such that $[x]_k = i$. Furthermore, we use the notation $(i)_k$ for the canonical representation

of i in base k . For example, $(13)_2 = 1101$. Sometimes, if the context is clear, we just use $[x]$ instead of $[x]_k$ and (i) instead of $(i)_k$. We say a word z is a *valid right extension* of a word x if x is a prefix of z .

We now recall a known result about the subword complexity of automatic sequences.

Theorem 4. *Let $\mathbf{x}$ be an automatic sequence generated by an m -state DFAO $M = (Q, \Sigma, \delta, q_0, \Delta, \tau)$ with msd-first input and let $\hat{\mathbf{M}}$ be the sequence generated by the intrinsic DFAO $\hat{M} = (Q, \Sigma, \delta, q_0, Q, \iota)$, where ι is the identity map. Then $\rho_{\mathbf{x}}(n) \leq kn\rho_{\hat{\mathbf{M}}}(2) \leq knm^2$ for all $n \geq 1$.*

Proof. See [3, Theorem 10.3.1]. We point out that the statement of the theorem in [3] only gives the inequality $\rho_{\mathbf{x}}(n) \leq knm^2$, but the proof actually presented there provides the other inequality. $\square$

2.3 Recognizing Relations

We say a DFA recognizes a relation (such as the addition relation $i + j = l$) if it takes the values of the variables in parallel, represented in some base k , and accepts if and only if the relation holds. In this section we review some of the basic arithmetic operations, what is known about their state complexity, and some new results.

2.3.1 Addition

We start by reviewing a well-known construction for recognizing the addition relation.

Theorem 5. *For both lsd-first and msd-first input, there is a 2-state automaton that recognizes the relation $[x]_k + [y]_k = [z]_k$.*

Proof. The automaton for msd-first base 2 and lsd-first base 2 input can be created by the methods explained, respectively, by Wolper and Boigelot [48] and Boudet and Comon [10]. Furthermore, the automaton for msd-first base 2 can be found, for example, in Hodgson [22] and the automaton for msd-first base k can be found, for example, in Waxweiler [47, Proposition 4.4.2]; the msd-first base k construction can be easily modified to obtain the lsd-first base k construction.

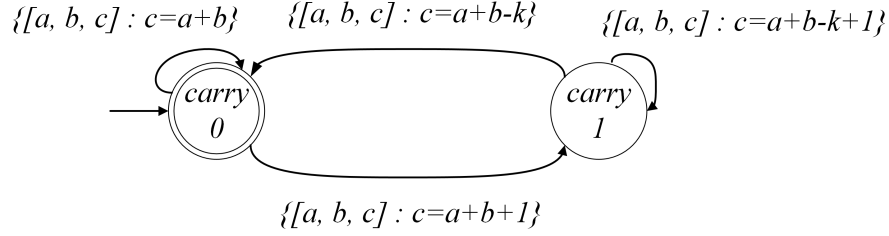


Figure 2.1: An automaton with msd-first input accepting $x \times y \times z$ if and only if $[x]_k + [y]_k = [z]_k$.

We only need to do the addition while reading the input and keep track of the carry (0 or 1) by 2 states. The automaton for addition in msd-format depicted in Figure 2.1 is $M = (Q, \Sigma_k, \delta, q_0, A)$ where

$$\begin{aligned}
 Q &= \{q_0, q_1\}, \\
 \delta(q, [a, b, c]) &= \begin{cases} q_0, & \text{if } q = q_0, c = a + b; \\ q_1, & \text{if } q = q_0, c = a + b + 1; \\ q_1, & \text{if } q = q_1, c = a + b - k + 1; \\ q_0, & \text{if } q = q_1, c = a + b - k; \end{cases} \\
 A &= \{q_0\}.
 \end{aligned}$$

Similarly, the automaton for addition in lsd-format depicted in Figure 2.2 is $M = (Q, \Sigma_k, \delta, q_0, A)$ where

$$\begin{aligned}
 Q &= \{q_0, q_1\}, \\
 \delta(q, [a, b, c]) &= \begin{cases} q_0, & \text{if } q = q_0, c = a + b; \\ q_0, & \text{if } q = q_1, c = a + b + 1; \\ q_1, & \text{if } q = q_1, c = a + b - k + 1; \\ q_1, & \text{if } q = q_0, c = a + b - k; \end{cases} \\
 A &= \{q_0\}.
 \end{aligned}$$

Correctness is left to the reader. □

Now we consider what happens when we specify one summand to be a fixed constant.

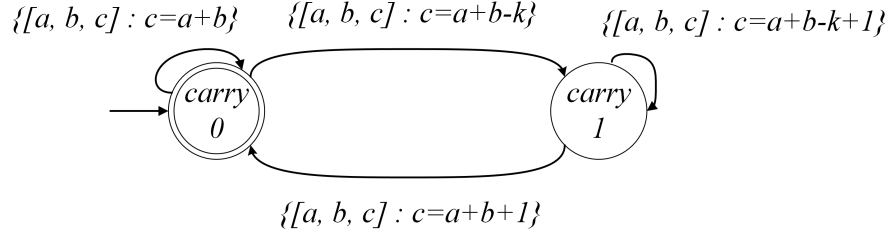


Figure 2.2: An automaton with lsd-first input accepting $x \times y \times z$ if and only if $[x]_k + [y]_k = [z]_k$.

Theorem 6. *Let $c \geq 1$ be a fixed integer constant. For both lsd- and msd-first input, there exists an automaton with $O(\log_k c)$ states accepting $x \times y$ where $[x]_k + c = [y]_k$.*

Proof. We first design the automaton M_c accepting $[x]_k + c = [y]_k$ for msd-first input. The automaton for the special case of base 2 can be created using the construction provided by Wolper and Boigelot [48], which is similar to what we do here.

On input $x \times y$, we need to keep track of the difference $D_k(x, y) = [y]_k - [x]_k$ and accept the input if and only if the difference is c . Each state of the automaton is a possible difference d , but we need to find a suitable range I for d so that the automaton is actually finite. We have

$$[yb]_k - [xa]_k = k([y]_k - [x]_k) + b - a. \quad (2.1)$$

Therefore, if $D(x, y)$ is outside the range $[0, c]$ it will not return to it for any right extension of $x \times y$, and we can take I to be the interval $[0, c]$. The transition function is based on Equation (2.1). After reading each input letter pair, the difference either stays the same or increases (unless it is no longer in the range I). Furthermore, if we consider a state $d \neq 0$, there are only two other states that have a direct transition to d , namely $\lfloor d/k \rfloor$ and $\lceil d/k \rceil$.

So we construct the automaton $M_c = (Q, \Sigma_k^2, \delta, q_0, A)$ where $\delta(d, [a, b]) = kd + b - a$, $q_0 = 0$, and $A = \{c\}$. To create the set Q , we start with $\{c\}$. Then we add the states $c' = \lfloor c/k \rfloor$ and $c'' = \lceil c/k \rceil$ to Q . We know $|c'' - c'| \leq 1$; therefore, the set $\{\lfloor c'/k \rfloor, \lfloor c''/k \rfloor, \lceil c'/k \rceil, \lceil c''/k \rceil\}$ has size at most 2. We repeat the same step (adding the floor and ceiling of division by k to Q) until reaching the state 0. This step is repeated $O(\log_k c)$ times and there are always at most two new states created at each step. Therefore, the number of states in M_c is $O(\log_k c)$.

We now design the automaton M_c for lsd-first input. The automaton for the special case of base 2 can also be created using the construction provided by Boudet and Comon [10].

For the lsd-first input, we only need to use the simple manual addition method. We first write $(c)_k = \dots c_1 c_0$ with $\lfloor \log_k c + 1 \rfloor$ letters. At first, after reading each letter pair, the automaton keeps track of the carry (0 or 1) and how many letters from $(c)_k$ have been processed (the level starting from 0). After processing all letters in $(c)_k$, in an extra final level (level $\lfloor \log_k c + 1 \rfloor$), the automaton only keeps track of the carry.

So we construct the automaton $M_c = (Q, \Sigma_k^2, \delta, q_0, A)$ as follows. The set Q consists of pairs $[l, r]$ where l is the level and r is the carry.

$$\begin{aligned} Q &= \{[l, r] \in \mathbb{N}^2 : l \leq \lfloor \log_k c + 1 \rfloor, r \in \{0, 1\}\}, \\ \delta([l, r], [a, b]) &= \begin{cases} [l, (r + a - b)/k], & \text{if } l = \lfloor \log_k c + 1 \rfloor; \\ [l + 1, (r + c_l + a - b)/k], & \text{otherwise;} \end{cases} \\ q_0 &= [0, 0], \\ A &= \{[l, r] \in Q : l = \lfloor \log_k c + 1 \rfloor, r = 0\}. \end{aligned}$$

Note that in the transition function, if the new carry r computed is not 0 or 1, the automaton transitions to the dead state.

The automaton has $O(\log_k c)$ levels of states and each level has two states for carry 0 and carry 1; therefore, the automaton has $O(\log_k c)$ states. $\square$

Theorem 7. *Let $c \geq 1$ be a fixed constant. For both lsd- and msd-first base- k input, there exists an automaton with $O(\log_k c)$ states accepting $x \times y$ where $[x]_k - c = [y]_k$.*

Proof. For the msd-first base 2 input, the construction provided by Wolper and Boigelot [48] can be used here too. For the msd-first base k input, recall the automaton M_c for msd-first in the proof of Theorem 6. We only need to modify the previous automaton. Here, instead of $[y]_k - [x]_k$, we keep track of $[x]_k - [y]_k$ and the transition function is

$$\delta(d, [a, b]) = kd + a - b.$$

The automaton construction, the range I , and acceptance criteria of $d = c$ stay the same.

For the lsd-first base 2 input, the construction provided by Boudet and Comon [10] can be used here too. For the lsd-first base k input, the lsd-first automaton from the proof of Theorem 6 can be modified. The new transition function is

$$\delta([l, r], [a, b]) = \begin{cases} [l, (r + b - a)/k], & \text{if } l = \lfloor \log_k c + 1 \rfloor; \\ [l + 1, (r + c_l + b - a)/k], & \text{otherwise.} \end{cases}$$

Other components of the automaton stay the same. □

Recall the definition of the k -adic valuation: $\nu_k(n) = e$ if $k^e \mid n$ but $k^{e+1} \nmid n$. Note that $\nu_k(c)$ is the number of consecutive 0s at the end of $(c)_k$.

We also use the following notation: if some condition is described inside brackets, the brackets are Iverson brackets. The Iverson bracket $[P]$ equals 1 if the condition P is satisfied, and 0 otherwise. The next theorem appears to be new.

Theorem 8. *Let $c \geq 1$ be a fixed constant. For msd-first input, the number of states in the minimal DFA recognizing $x \times y$ where $[x]_k + c = [y]_k$ is exactly*

$$2|(c)_k| + 1 - \nu_k(c) - [(c)_k \text{ starts with } 1] - [k = 2, (c)_2 \text{ starts with } 10], \text{ and } c \text{ is not a power of } 2]. \quad (2.2)$$

Proof. Consider the construction of the automaton M_c for msd-first input described in the proof of Theorem 6. Let us group the new states created at each step of the construction into a *level* including an initial level for the state c . We call a level with 1 state a single-state level and a level with 2 states a double-state level.

In its simplest form, our construction creates an automaton with consecutive double-state levels and an initial single-state level for c . At each step of the construction, the double-state levels are created by computing both the ceiling and floor of an integer divided by k or the floor and ceiling of two consecutive integers divided by k . The number of times we need to divide c by k and apply floor and ceiling to get to the 0 state is $|(c)_k|$. Therefore, in the simple form, there are $2|(c)_k| + 1$ states. The automaton M_7 for base 3 depicted in Figure 2.3 is an example of the simple form.

However, for some values of k and c , some double-state levels in the simple form collapse to single-state levels, causing the number of states to be lower than the simple form. We study these cases, how they are reflected in Equation 2.2, and why they are the only possible such cases.

Case 1: A positive power of k divides c .

If k divides some integer c' , then $\lfloor c'/k \rfloor = \lceil c'/k \rceil$. As a result, in this case, instead of the last level being single-state, the last $\nu_k(c) + 1$ levels are single-state. So Case 1 is covered by the $-\nu_k(c)$ in Equation 2.2. The automaton M_6 in base 3 depicted in Figure 2.4 is an example of this case.

Case 2: The word $(c)_k$ starts with 1, or $k = 2$ and $(c)_2$ starts with 10.

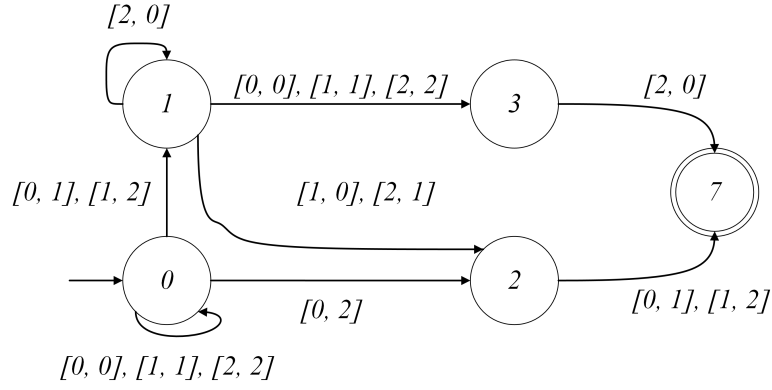


Figure 2.3: An automaton with msd-first base-3 input accepting $x \times y$ if and only if $[x]_3 + 7 = [y]_3$.

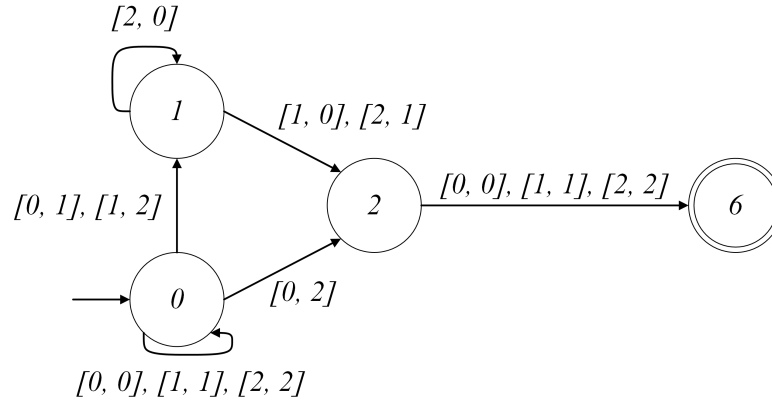


Figure 2.4: An automaton with msd-first base-3 input accepting $x \times y$ if and only if $[x]_3 + 6 = [y]_3$.

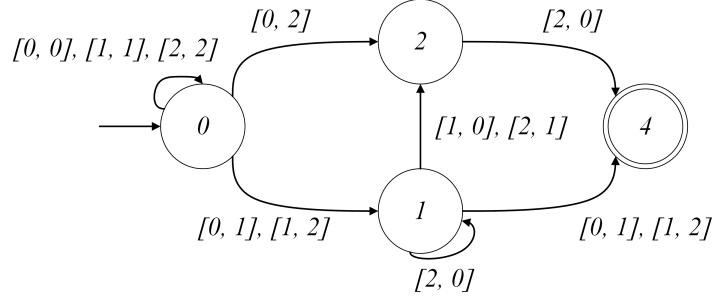


Figure 2.5: An automaton with msd-first base-3 input accepting $x \times y$ if and only if $[x]_3 + 4 = [y]_3$.

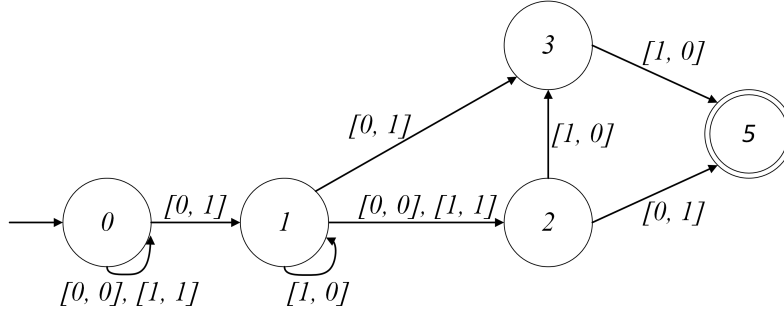


Figure 2.6: An automaton with msd-first base-2 input accepting $x \times y$ if and only if $[x]_2 + 5 = [y]_2$.

The shortest path from state 0 to state c indicated by the transitions is $0^{|(c)_k|} \times (c)_k$. If $(c)_k$ starts with 1, the shortest path starts with $[0, 1]$. As a result, the state 1 is included in the automaton, but the states 0 and 1 cannot be on the same level; otherwise, the shortest path to c does not start with $[0, 1]$. The automaton M_4 for base 3 depicted in Figure 2.5 is an example.

Similarly, if $k = 2$ and $(c)_k$ starts with 10, the shortest path starts with $[0, 1]$ followed by $[0, 0]$. As a result, the states 0, 1, 2 are included in the automaton and not only 0 and 1 cannot be on the same level, but also 1 and 2 cannot be on the same level. The automaton M_5 for base 2 depicted in Figure 2.6 is an example.

If $k = 2$, the word $(c)_2$ starts with 10, and also c is a power of 2, the state reduced in the second level is already taken into account in the term $-\nu_k(c)$ described in Case 1. Therefore, Case 2 is covered by the terms $-\lfloor (c)_k \text{ starts with } 1 \rfloor - \lfloor k =$

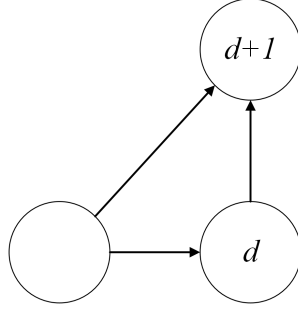


Figure 2.7: A double-state followed by a single-state in an automaton accepting $x \times y$ if and only if $[x]_k + c = [y]_k$.

2, $(c)_k$ starts with 10, and c is not a power of 2] in Equation 2.2, while ensuring absence of overlap with Case 1.

It remains to show that besides cases 1 and 2, no additional reductions in the number of states compared to the simple form is possible.

Consider a double-state level consisting of states d and $d + 1$ following a single-state level d' (Figure 2.7). In this case, d transitions to $d + 1$ (proof by contradiction). We have $d + 1 = kd + b - a$ for some $a, b \in [0, k - 1]$, $d \neq 0$, and $k \geq 2$. The only possible solutions to this equation are the following:

- $d = 1, b = 0, a = k - 2$,
- $d = 1, b = 1, a = k - 1$,
- $d = 2, b = 0, a = 2k - 3, k = 2$,

which are covered in Case 2.

If some positive integer c' is not divisible by k , then the floor and ceiling of c'/k are different numbers, except for the special case already discussed in Case 2. So Case 1 covers all the other single-state levels not counted in Case 2.

Therefore, Equation 2.2 introduced in the theorem correctly computes the number of states in the minimal automaton M_c . $\square$

2.3.2 Multiplication

In addition to providing an msd-first automaton construction for recognizing the relation $[x]_k + [y]_k = [z]_k$, Waxweiler [47, Proposition 4.4.3] provided one for recognizing $n[x]_k = [y]_k$. Later, Charlier et al. [15] provided a construction for recognizing $n[x]_k + c = [y]_k$. Furthermore, the automaton for $n[x]_k + c = [y]_k$ with msd-first base-2 input (resp., lsd-first base-2 input) can be created by the methods explained by Wolper and Boigelot [48] (resp., Boudet and Comon [10]). The following theorem considers the automaton accepting $n[x]_k + c = [y]_k$ in the case $c < n$ with both lsd- and msd-first base k input.

Theorem 9. *Let $n \geq 1$, $0 \leq c < n$ be fixed constants. For both lsd- and msd-first input, there is an n -state automaton accepting $x \times y$ where $n[x]_k + c = [y]_k$.*

Proof. For the msd-first input, let x, y be two words of the same length in Σ_k^* . We define $D_k(x, y) = [y]_k - n[x]_k$. The basic idea is to define a DFA $M_{n,c,k} = (Q, \Sigma, \delta, q_0, A)$ such that $\delta(q_0, x \times y) = D_k(x, y)$, but some additional consideration is needed, since D_k could be arbitrarily large in absolute value. This would seem to require infinitely many states.

However, it is easy to see that

- (a) If $D_k(x, y) \leq -1$, then $D_k(xa, yb) \leq -1$ for $a, b \in \Sigma_k$.
- (b) If $D_k(x, y) \geq n$, then $D_k(xa, yb) \geq n$ for $a, b \in \Sigma_k$.

We can therefore take our set of states Q to be the set of all possible differences; namely, $Q = \{0, 1, \dots, n-1\}$. Thus, if an input to the automaton $x \times y$ ever causes $D_k(x, y)$ to fall outside the closed interval $[0, n-1]$, then no right extension of this input could satisfy the acceptance criteria. So in this case we can go to the (unique) dead state.

The transition rule for the automaton is now easy to compute. Suppose $d = D_k(x, y)$. After reading $xa \times yb$, where $a, b \in \Sigma_k$ are single letters, the new difference is

$$[yb]_k - n[xa]_k = k[y]_k + b - n(k[x]_k + a) = kd + b - na.$$

This gives the transition rule of the automaton: $\delta(d, [a, b]) = kd + b - na$, provided $kd + b - na$ lies in the interval $[0, n-1]$; otherwise the automaton transitions to the dead state. A state d is accepting if and only if $d = c$.

For the lsd-first input, each state corresponds to a carry in the range $[0, c]$. The initial state is c to account for the $+c$. By reading $[a, b]$, the transition function computes the multiplication by n and addition by the carry and the automaton transitions to the new

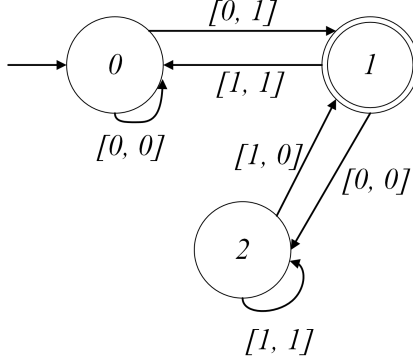


Figure 2.8: An automaton with msd-first base-2 input accepting $x \times y$ if and only if $3[x]_2 + 1 = [y]_2$.

carry state. Since $c < n$, a simple proof by induction shows that the maximum carry possible is $c \leq n - 1$ and the range $[0, c]$ for states is sufficient.

So for lsd-first format we construct the automaton $M = (Q, \Sigma_k^2, \delta, c, A)$ where

$$\begin{aligned} Q &= \{r \in \mathbb{N} : r \in [0, c]\}, \\ \delta(r, [a, b]) &= (na + r - b)/k, \\ A &= \{r \in Q : r = 0\}. \end{aligned}$$

This automaton has $c + 1 \leq n$ states. □

Example 10. *The automaton with msd-first input created by the construction in Theorem 9 for $k = 2, n = 3, c = 1$ is depicted in Figure 2.8.*

2.4 Linear Subsequences of Automatic Sequences

We now turn to examining the state complexity of transformations on automatic sequences.

Zantema and Bosma [51, Theorem 16] showed that for lsd-first input, if $(h(i))_{i \geq 0}$ is a k -automatic sequence generated by a DFAO of m states, then $(h(i + 1))_{i \geq 0}$ is generated by a DFAO of at most $2m$ states. Furthermore, in the same theorem they showed that for msd-first input, the sequence $(h(i + 1))_{i \geq 0}$ is generated by a DFAO of at most m^2 states and there is an example where this bound is tight. Additionally, they [51, Theorem 17]

showed that the same bounds hold for $(h(i-1))_{i \geq 0}$ where $h(0)$ is any chosen letter from the sequence alphabet.

However, they did not show that their upper bound of $2m$ states for $(h(i+1))_{i \geq 0}$ in the lsd-first case is tight. We suspect it is not. In the following result, we show there are examples attaining the very slightly weaker bound of $2m-1$ states.

Theorem 11. *For lsd-first input, for each $m \geq 2$, there exists an m -state DFAO M generating the sequence $(h(i))_{i \geq 0}$ such that at least $2m-1$ states are required in the DFAO M' generating $(h(i+1))_{i \geq 0}$.*

Proof. We use an adaptation of the Myhill-Nerode theorem [24, §3.4] for DFAOs. The adaptation for DFAOs is also used by Zantema and Bosma [51] and the state equivalence for DFAOs is defined by van Spaendonck [46].

Let $k = 2$, $m \geq 3$, and define the lsd-first DFAO M generating $(h(i))_{i \geq 0}$ depicted in Figure 2.9 as follows.

$$\begin{aligned} Q &= \{0, \dots, m-1\}, \\ \delta(q, a) &= \begin{cases} (q+1) \bmod m, & \text{if } a = 1; \\ q+1, & \text{if } a = 0 \text{ and } q \in \{0, \dots, m-3\}; \\ q, & \text{if } a = 0 \text{ and } q \in \{m-2, m-1\}; \end{cases} \\ q_0 &= 0, \\ \Delta &= \Sigma_2, \\ \tau(q) &= \begin{cases} 1, & \text{if } q = m-1; \\ 0, & \text{otherwise.} \end{cases} \end{aligned}$$

Let M' be the lsd-first DFAO generating the sequence $(h(i+1))_{i \geq 0}$. We show that there are $2m-1$ words w_i such that for all w_i, w_j where $i \neq j$, there exists w_{ij} such that the output of M' on input $w_i w_{ij}$ is different from the output of M' on input $w_j w_{ij}$. In this case, we say w_i and w_j are *distinguishable*.

The $2m-1$ pairwise distinguishable words are the following.

- 0^i where $1 \leq i \leq m-2$,
- 1^i where $0 \leq i \leq m-2$,
- $0^{m-2}1$,

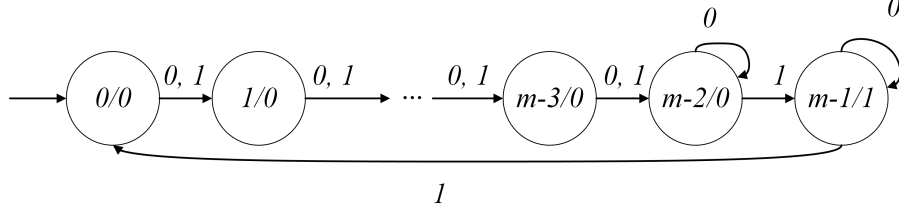


Figure 2.9: An automaton with base-2 lsd-first input generating a 2-automatic sequence $(h(i))_{i \geq 0}$.

- $0^{m-2}11$.

We need to show each pair of words is distinguishable. There are eight cases to consider.

Case 1: 0^i and 0^j where $1 \leq i < j \leq m-2$. Concatenate $0^{m-j-2}1$. The new words are $0^{i+m-j-2}1$ and $0^{m-2}1$. The output of the automaton M' on these words is then the same as the output of the automaton M on inputs $10^{i-j+m-3}1$ and $10^{m-3}1$, which are 0 and 1. Therefore, the two words are distinguishable.

Proving the rest of the cases follows the same logic.

Case 2: 1^i and 1^j where $0 \leq i < j \leq m-2$. Concatenate $1^{m-j-2}0$.

Case 3: $0^{m-2}1$ and $0^{m-2}11$. Concatenate λ .

Case 4: 0^i and 1^j where $1 \leq i \leq m-2$ and $0 \leq j \leq m-2$. Concatenate $1^{2m-i}0$.

Case 5: 0^i and $0^{m-2}1$ where $1 \leq i \leq m-2$. Concatenate λ .

Case 6: 0^i and $0^{m-2}11$ where $1 \leq i \leq m-2$. Concatenate 1^{m-i-1} .

Case 7: 1^i and $0^{m-2}1$ where $0 \leq i \leq m-2$. Concatenate 1^{m-i-1} .

Case 8: 1^i and $0^{m-2}11$ where $0 \leq i \leq m-2$. Concatenate 1^{m-i-2} .

Therefore, the DFAO M' for $(h(i+1))_{i \geq 0}$ requires at least $2m-1$ states. $\square$

Theorem 12. Suppose $(h(i))_{i \geq 0}$ is an automatic sequence generated by a DFAO of m states with msd-first input. Then there is a DFAO of $O(m^2)$ states with msd-first input generating the two-dimensional automatic sequence $(h(i+j))_{i,j \geq 0}$.

Proof. Let the DFAO generating the sequence $(h(i))_{i \geq 0}$ be $M = (Q, \Sigma_k, \delta, q_0, \Delta, \tau)$. We want to design a DFAO M' for $(h(i+j))_{i,j \geq 0}$. The basic idea is to perform $i+j$ while reading the representation of i and j in parallel; however, upon reading a letter pair $[a, b]$ and before reading the next letter pair, we do not know what the letter corresponding to $[a, b]$ in $i+j$ is. For example, $[00]_3 + [01]_3 = [01]_3$ but $[01]_3 + [02]_3 = [10]_3$; the first letter pairs read are the same but the first letters in the addition results are not. So each state in the automaton M' includes two states from M : one of them corresponds to the case where a carry is not added to $a+b$, and one to the case where a carry is added.

More formally, we construct the automaton $M' = (Q', \Sigma_k^2, \delta', q'_0, \Delta, \tau')$ as follows.

$$\begin{aligned} Q' &= Q^2, \\ \delta'([q, p], [a, b]) &= \begin{cases} [\delta(q, a+b), \delta(q, a+b+1)], & \text{if } a+b \leq k-2; \\ [\delta(q, a+b), \delta(p, a+b+1-k)], & \text{if } a+b = k-1; \\ [\delta(p, a+b-k), \delta(p, a+b+1-k)], & \text{if } a+b \geq k; \end{cases} \\ q'_0 &= [\delta(q_0, 0), \delta(q_0, 1)], \\ \tau'([q, p]) &= \tau(q). \end{aligned}$$

The automaton is constructed so that if $[q, p]$ is a state in the automaton reachable by $x \times y$, then $[q, p] = [h'([x]_k + [y]_k), h'([x]_k + [y]_k + 1)]$. So the reachable states are length-2 subsequences of the interior sequence $(h'(i))_{i \geq 0}$. Therefore, based on Theorem 4, there are $O(m^2)$ reachable states. $\square$

We now turn to linear subsequences of automatic sequences. More specifically, we consider the following problem. Let $(h(i))_{i \geq 0}$ be a k -automatic sequence generated by a DFAO of m states. Let $n \geq 1$ and $c \geq 0$ be integer constants. How many states are needed to generate $(h(ni+c))_{i \geq 0}$?

Zantema and Bosma [51] studied the specific case where $n = k$ and $0 \leq c < k$. They showed that for this case and for both msd-first and lsd-first input, the subsequence $(h(ki+c))_{i \geq 0}$ is generated by a DFAO of at most m states.

In [3, Proof of Theorem 6.8.1], it is proved that if $(h(i))_{i \geq 0}$ is a k -automatic sequence generated by an m -state DFAO with lsd-first input, then for $n \geq 1, c \geq 0$ the linear subsequence $(h(ni+c))_{i \geq 0}$ is generated by an lsd-first DFAO with $m(n+c)$ states. (The theorem is phrased in terms of the k -kernel, but the number of elements of the k -kernel of a sequence is the same as the number of states in a minimal lsd-first DFAO generating the sequence.)

In the special case that $0 \leq c < n$, this lsd-first bound was improved slightly to at most mn states by Zantema and Bosma [51]. Furthermore, they provided examples where the same bound does not hold for msd-first input. They stated obtaining a bound in the msd-first case as an open question. We address this in the following theorem, which provides a close and novel connection between state complexity and subword complexity.

Theorem 13. *Let $n \geq 1$, $c \geq 0$ be fixed integer constants. Let $\mathbf{h} = (h(i))_{i \geq 0}$ be an automatic sequence generated by a DFAO of m states with msd-first input, and let $\mathbf{h}' = (h'(i))_{i \geq 0}$ be the interior sequence of $\mathbf{h}$.*

- (a) *If $c < n$, then there is a DFAO of at most $\rho_{\mathbf{h}'}(n)$ states generating $(h(ni + c))_{i \geq 0}$ with msd-first input.*
- (b) *If $c \geq n$, then there is a DFAO of at most $\rho_{\mathbf{h}'}(c + 1)$ states generating $(h(ni + c))_{i \geq 0}$ with msd-first input.*

Proof. Let $(h(i))_{i \geq 0}$ be generated by a DFAO $M = (Q, \Sigma_k, \delta, q_0, \Delta, \tau)$.

- (a) Suppose $c < n$. We want to create a DFAO generating $(h(ni + c))_{i \geq 0}$. The idea is that we construct M' such that on input x it reaches a state given by the following subword of the interior sequence $(h'(i))_{i \geq 0}$:

$$[h'(n[x]_k), h'(n[x]_k + 1), \dots, h'(n[x]_k + n - 1)].$$

Then, since $c \leq n - 1$, we only need to obtain $h'(n[x]_k + c)$ by mapping the subword $[r_0, \dots, r_{n-1}]$ to r_c .

Let $S_n(h')$ be the set of length- n subwords of the sequence $(h'(i))_{i \geq 0}$. We define $M' = (Q', \Sigma_k, \delta', q'_0, \Delta, \tau')$ as follows:

$$\begin{aligned} Q' &= \{[r_0, \dots, r_{n-1}] \in S_n(h')\}, \\ \delta'([r_0, \dots, r_{n-1}], a) &= [s_0, \dots, s_{n-1}] \\ &\quad \text{where } s_d = \delta(r_j, b), \ j = \lfloor (na + d)/k \rfloor, \ b = (na + d) \bmod k, \\ q'_0 &= [h'(0), h'(1), \dots, h'(n - 1)], \\ \tau'([r_0, \dots, r_{n-1}]) &= \tau(r_c). \end{aligned}$$

The states of M' are the length- n subwords of the interior sequence $(h'(i))_{i \geq 0}$. Note that not all states are necessarily reachable from the initial state.

First, let us see that

$$s_d = \delta(r_j, b), \quad j = \lfloor (na + d)/k \rfloor, \quad b = (na + d) \bmod k$$

in the transition function is well-defined. We have $0 \leq a \leq k - 1$, and the definition of b implies that $0 \leq b < k$. We need to verify that $0 \leq j \leq n - 1$. We do this as follows: $j = \lfloor (na + d)/k \rfloor$ implies that $j \leq (n(k - 1) + d)/k$. Since $d < n$, we see this implies that $kj < nk$. Hence $j < n$, as desired.

Now let us prove by induction on input length $|x|$ that

$$\delta'(q'_0, x) = [h'(n[x]_k), \dots, h'(n[x]_k + n - 1)]. \quad (2.3)$$

The base case is $x = \lambda$. Here the result follows by the definition of q'_0 .

Otherwise, assume Equation (2.3) is true for $|x|$; we prove it for $|x| + 1$. Write $y = xa$. So $[y]_k = k[x]_k + a$. Then $\delta'(q'_0, y) = \delta'(\delta'(q'_0, x), a)$. By induction $\delta'(q'_0, x) = [r_0, r_1, \dots, r_{n-1}]$ where $r_j = h'(n[x]_k + j)$. By the definition of δ' we have $\delta'(q'_0, y) = [s_0, s_1, \dots, s_{n-1}]$ where

$$\begin{aligned} s_d &= \delta(r_j, b) = \delta(h'(n[x]_k + j), b) = \delta(q_0, k(n[x]_k + j) + b) \\ &= \delta(q_0, kn[x]_k + na + d) = h'(kn[x]_k + na + d) = h'(n(k[x]_k + a) + d) \\ &= h'(n[y]_k + d), \end{aligned}$$

as desired.

- (b) Suppose $c \geq n$. The proof proceeds exactly as in (a), except now the states are the length- $(c + 1)$ (instead of length- n) subwords of $(h'(i))_{i \geq 0}$. The idea is that on input x , the DFAO M' reaches a state given by the subword

$$[h'(n[x]_k), h'(n[x]_k + 1), \dots, h'(n[x]_k + c)]. \quad (2.4)$$

Similarly, the states are length- $(c + 1)$ subwords of the interior sequence $(h'(i))_{i \geq 0}$.

It is now not difficult to check, by exactly the same sort of calculation as in (a), that this is enough to construct the necessary subword giving the desired image, which will always be the last component of the state.

□

Corollary 14. *Let $n \geq 1$, $c \geq 0$ be fixed integer constants. Let $(h(i))_{i \geq 0}$ be an automatic sequence generated by a DFAO of m states with msd -first input.*

- (a) If $c < n$, then there is a DFAO of at most knm^2 states generating the automatic sequence $(h(ni + c))_{i \geq 0}$ with msd-first input.
- (b) If $c \geq n$, then there is a DFAO of at most $k(c+1)m^2$ states generating the automatic sequence $(h(ni + c))_{i \geq 0}$ with msd-first input.

Proof.

- (a) From Theorem 4, we know $\rho_{\mathbf{h}'}(n) \leq knm^2$.
- (b) Similarly, we know $\rho_{\mathbf{h}'}(c+1) \leq k(c+1)m^2$.

□

We do not know if the bounds in Corollary 14 are tight. This seems worthy of further study, so we state it as an open problem.

Problem 1. Find a class of automatic sequences $(h_m(i))_{i \geq 0}$, generated by a DFAO of m states with msd-first input, such that there is a constant C so the linear subsequences $(h_m(ni))_{i \geq 0}$ have at least the state complexity Cnm^2 for infinitely many n .

2.4.1 Connection with Subword Complexity

We now consider an interesting application of this connection between state complexity and subword complexity. Given an automatic sequence $\mathbf{h} = (h(i))_{i \geq 0}$ we would like to obtain an upper bound on the subword complexity of its linear subsequence $(h(ni))_{i \geq 0}$.

Theorem 15. Let $\mathbf{h} = (h(i))_{i \geq 0}$ be a k -automatic sequence and $\mathbf{h}'$ its interior sequence. Let $\mathbf{h}_n$ denote the linear subsequence $(h(ni))_{i \geq 0}$ and similarly for $\mathbf{h}'_n$. Then $\rho_{\mathbf{h}_n}(\ell) \leq k\ell\rho_{\mathbf{h}'}(2n)$ for all $\ell, n \geq 1$.

Proof. Clearly $\rho_{\mathbf{h}_n}(\ell) \leq \rho_{\mathbf{h}'_n}(\ell)$. Let $M = (Q, \Sigma_k, \delta, q_0, \Delta, \tau)$ be an automaton generating $\mathbf{h}'_n$ built by the construction in Theorem 13, and let $\hat{\mathbf{M}}$ be the sequence generated by the intrinsic DFAO $\hat{M} = (Q, \Sigma_k, \delta, q_0, Q, \iota)$ where $\iota : Q \rightarrow Q$ is the identity map. By Theorem 4, we have $\rho_{\mathbf{h}'_n}(\ell) \leq k\ell\rho_{\hat{\mathbf{M}}}(2)$. However, each length-2 subword of $\hat{\mathbf{M}}$ corresponds, from the construction of Theorem 13, to a distinct length- $2n$ subword of $\mathbf{h}'$. Thus $\rho_{\hat{\mathbf{M}}}(2) \leq \rho_{\mathbf{h}'}(2n)$. Putting this all together gives

$$\rho_{\mathbf{h}_n}(\ell) \leq \rho_{\mathbf{h}'_n}(\ell) \leq k\ell\rho_{\hat{\mathbf{M}}}(2) \leq k\ell\rho_{\mathbf{h}'}(2n),$$

as desired. □

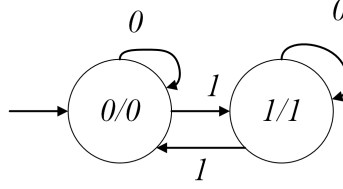


Figure 2.10: An automaton with base-2 input generating the Thue-Morse sequence.

Remark 16. A superficially similar notion of complexity was studied by Konieczny and Müllner [29]. However, the particular notion of complexity they studied is the number of length- ℓ subwords of all of the subsequences $(h(ni + c))_{i \geq 0}$ for integers $n \geq 1$ and $c \geq 0$; this notion was first introduced by Avgustinovich et al. [6]. In contrast, the complexity we discuss in this chapter is the number of length- ℓ subwords of a single subsequence $(h(ni))_{i \geq 0}$ with fixed n .

2.4.2 Thue-Morse Sequence

We now apply some of the previous theorems to the famous Thue-Morse sequence $\mathbf{t} = (t(i))_{i \geq 0}$ generated by the DFAO in Figure 2.10 [3, Chapter 5]. The Thue-Morse sequence is defined as the fixed point of the morphism $0 \rightarrow 01, 1 \rightarrow 10$ [2].

First we apply Theorem 15 to the Thue-Morse sequence.

Corollary 17. *Let $\mathbf{t}_n$ denote the linear subsequence $(t(ni))_{i \geq 0}$. We have $\rho_{\mathbf{t}_n}(\ell) \leq \frac{40}{3}\ell n$ for $\ell, n \geq 1$.*

Proof. Results from Avgustinovich [5] imply that $\rho_{\mathbf{t}}(n) \leq \frac{10}{3}n$. So by Theorem 15, we have $\rho_{\mathbf{t}_n}(\ell) \leq 2 \cdot \ell \cdot \frac{10}{3} \cdot 2n = \frac{40}{3}\ell n$. $\square$

Now we apply Theorem 13 to the Thue-Morse sequence $(t(i))_{i \geq 0}$.

Example 18. *If we apply Theorem 13 to the Thue-Morse sequence with $n = 1, c = 1$, we get the DFAO in Figure 2.11 generating $(t(i + 1))_{i \geq 0}$.*

Theorem 19. *For all integers $n \geq 1$, the number of states in the minimal DFAO generating $(t(ni))_{i \geq 0}$ with msd-first input is $\rho_{\mathbf{t}}(n/\nu_2(n))$.*

In order to prove Theorem 19, we first need Lemma 20 and Theorem 21.

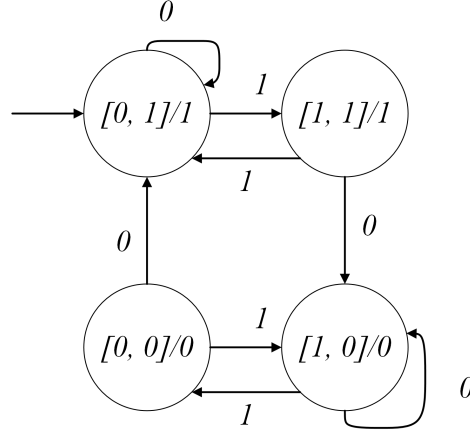


Figure 2.11: An automaton with msd-first base-2 input generating $(t(i+1))_{i \geq 0}$.

Lemma 20. *For an odd integer n , all length- n subwords of the Thue-Morse sequence $\mathbf{t}$ are reachable states in the automaton generating $(t(ni))_{i \geq 0}$ with msd-first input built by the construction provided in Theorem 13.*

Proof. We use results by Blanchet-Sadri et al. [8, Lemma 12] by setting $i = 0$ in their lemma. As a result, for each subword of the Thue-Morse sequence and odd integer n , there exists integer i' such that the subword occurs at index ni' of the Thue-Morse sequence; in other words, the state corresponding to the subword is reachable by input $(i')_2$. $\square$

Theorem 21. *For all odd integers $n \geq 1$, the construction provided in Theorem 13 is optimal for $(t(ni))_{i \geq 0}$.*

Proof. Consider the automaton created by the construction for $(t(ni))_{i \geq 0}$. On input x the automaton reaches the state $[t(n[x]), \dots, t(n[x] + n - 1)]$ where the output is $t(n[x])$. Furthermore, by Lemma 20 we know all length- n subwords of the Thue-Morse sequence are reachable states in the automaton.

Now suppose we have two distinct states in this automaton reachable by x and y :

$$[t(n[x]), \dots, t(n[x] + n - 1)] \neq [t(n[y]), \dots, t(n[y] + n - 1)].$$

We need to show they are distinguishable.

If $t(n[x]) \neq t(n[y])$, the distinguishability is trivial. If $t(n[x]) = t(n[y])$, there is $0 < j < n$ such that $t(n[x] + j) \neq t(n[y] + j)$. We need to find z such that $t(n[xz]) \neq t(n[yz])$; in other words, we want $t(n[x0^{|z|}] + n[z]) \neq t(n[y0^{|z|}] + n[z])$.

For any word z , since $t(n[x]) = t(n[y])$, we have $t(n[x0^{|z|}]) = t(n[y0^{|z|}])$. So, if there is word z such that $\lfloor n[z]/2^{|z|} \rfloor = j$, the number of ones in $(n[x0^{|z|}] + n[z])_2$ is the sum of the number of ones in $(n[x] + j)_2$ and $(n[z] \bmod 2^{|z|})_2$; a similar result holds for y instead of x . Since $t(n[x] + j) \neq t(n[y] + j)$, we have $t(n[xz]) \neq t(n[yz])$. By iterating different z , we get $0 \leq \lfloor n[z]/2^{|z|} \rfloor < n$. Therefore, such z always exists. $\square$

Proof of Theorem 19. We know $t(2i) = t(i)$ and the automaton generating $(t(2ni))_{i \geq 0}$ is the same as the automaton generating $(t(ni))_{i \geq 0}$. Furthermore, Theorem 21 implies that for all odd n , the number of states in the minimal automaton generating $(t(ni))_{i \geq 0}$ is $\rho_{\mathbf{t}}(n)$.

Therefore, in order to obtain the number of states in the minimal automaton generating $(t(ni))_{i \geq 0}$, we remove powers of 2 from n to get an odd integer $n' = n/\nu_2(n)$. The automaton generating $(t(n'i))_{i \geq 0}$ is the same as the automaton generating $(t(ni))_{i \geq 0}$ and the number of states in the minimal automaton generating $(t(n'i))_{i \geq 0}$ is $\rho_{\mathbf{t}}(n') = \rho_{\mathbf{t}}(n/\nu_2(n))$. $\square$

Let $\text{sc}(h(ni))$ denote the number of states in a minimal automaton generating the sequence $(h(ni))_{i \geq 0}$.

Corollary 22. *For all integers $r, p \geq 1$, we have*

$$\begin{cases} \text{sc}(t(2ri)) = \text{sc}(t(ri)), \\ \text{sc}(t((2^p + r)i)) = 3 \cdot 2^p + 4(r - 1), & \text{if } 1 < r < 2^{p-1} \text{ and } r \bmod 2 = 1; \\ \text{sc}(t((2^p + 2^{p-1} + r)i)) = 5 \cdot 2^p + 2(r - 1), & \text{if } 1 < r < 2^{p-1} \text{ and } r \bmod 2 = 1. \end{cases}$$

Proof. We know $t(2i) = t(i)$ and the automaton generating $(t(2ri))_{i \geq 0}$ is the same as the automaton generating $(t(ri))_{i \geq 0}$. Therefore $\text{sc}(t(2ri)) = \text{sc}(t(ri))$.

The subword complexity of the Thue-Morse sequence was studied by Brlek [11], de Luca and Varricchio [31], and Avgustinovich [5]. Brlek [11] proved that for $n \geq 3$ where $n = 2^p + r' + 1$ and $0 < r' \leq 2^p$ we have

$$\rho_{\mathbf{t}}(n) = \begin{cases} 6 \cdot 2^{p-1} + 4r', & \text{if } 0 < r' \leq 2^{p-1}; \\ 8 \cdot 2^{p-1} + 2r', & \text{if } 2^{p-1} < r' \leq 2^p. \end{cases}$$

By Theorem 19, from the $0 < r' \leq 2^{p-1}$ case we get $\text{sc}(t((2^p + r)i)) = 3 \cdot 2^p + 4(r - 1)$. From the $2^{p-1} < r' \leq 2^p$ case we get $\text{sc}(t((2^p + 2^{p-1} + r)i)) = 5 \cdot 2^p + 2(r - 1)$. $\square$

Remark 23. At first glance it might appear that Charlier et al. [14, 15] already proved results similar to those in Corollary 22. However, this is not the case. The DFAO we considered takes $(i)_2$ as input and outputs $t(ni)$. The DFA they considered takes x as input and decides whether $[x]_k = ni + r$ where $t(i) = 0$, k is a power of 2, and $0 \leq r < n$.

Next we apply Theorem 13 to the Thue-Morse sequence $(t(i))_{i \geq 0}$ and $n = 1$.

Theorem 24. *For all integers $c \geq 1$, the number of states in the minimal automaton generating $(t(i + c))_{i \geq 0}$ with msd-first input is at most $\frac{10}{3}c$.*

Proof. Consider the automaton generating $(t(i + c))_{i \geq 0}$ built by the construction in Theorem 13. On input x the automaton reaches the state $[t([x]_k), \dots, t([x]_k + c)]$ where the output is $t([x] + c)$. So the number of states in the minimal automaton generating $(t(i + c))_{i \geq 0}$ is at most $\rho_t(c + 1)$.

We use the formula by Brlek [11] previously mentioned in the proof of Corollary 22. Let $c = 2^p + r'$. If $0 < r' \leq 2^{p-1}$, then $r' \leq \frac{1}{3}c$ and we have $\rho_t(c + 1) = 6 \cdot 2^{p-1} + 4r' = 3c + r' \leq \frac{10}{3}c$. If $2^{p-1} < r' \leq 2^p$, then $r' > \frac{1}{3}c$ and we have $\rho_t(c + 1) = 8 \cdot 2^{p-1} + 2r' = 4c - 2r' < \frac{10}{3}c$. $\square$

Unlike the case of Theorem 21, where the construction in Theorem 13 gives the minimal automaton, in the case of Theorem 24, the construction in Theorem 13 does not necessarily give the minimal automaton, so the upper bound in Theorem 24 is not tight. The next theorem gives a weak lower bound on the state complexity of $(t(i + c))_{i \geq 0}$.

Theorem 25. *The minimal DFAO computing $(t(i + c))_{i \geq 0}$ with msd-first input has more than $c^{0.694}$ states for infinitely many c .*

Proof. The idea is to consider the case $c = 2^n$ for $n \geq 1$. In this case we show, using the adaptation of the Myhill-Nerode theorem for DFAOs, that the minimal DFAO for $(t(i + c))_{i \geq 0}$ has at least F_{n+1} states. We let S_n be the set of all binary words of length n that start with 0 and have no two consecutive 1's. As is well-known, this set has cardinality F_{n+1} . We now show the members of this set are pairwise inequivalent under the adaptation of the Myhill-Nerode theorem.

Let $x, y \in S_n$ with $x \neq y$. We need to find a word z such that $t([xz]_2 + 2^n) \neq t([yz]_2 + 2^n)$. If $t([x]_2 + 2^n) \neq t([y]_2 + 2^n)$, we can choose $z = \lambda$, the empty word.

Otherwise, assume $t([x]_2 + 2^n) = t([y]_2 + 2^n)$. Since x, y are distinct, there must be some position p (numbered from left to right, starting with index 1) at which $x[p] \neq y[p]$ where $w[i]$ is the i -th letter of w . Without loss of generality, assume $x[p] = 0$ and $y[p] = 1$.

Choose $z = 0^p$. Let x' denote the binary representation of $[xz]_2 + 2^n$ and y' denote the binary representation of $[yz]_2 + 2^n$. Letting $|u|_1$ denote the number of 1's in a word u , we clearly have $|x'|_1 = |x|_1 + 1$, since a new 1 arises from the addition of 2^n . However, $|y'|_1 = |y|_1$, since adding 2^n causes the 1 at position p to become 0 and causes the 0 at position $p-1$ to become 1, while no other bits change (since y has no two consecutive 1's). It follows that $t([xz]_2 + 2^n) \neq t([yz]_2 + 2^n)$, as desired. Hence all the elements of S_n are pairwise inequivalent, and so the minimal DFAO for $(t(i+c))_{i \geq 0}$ has F_{n+1} states.

Finally, from the estimate $F_{n+1} \geq \frac{1}{2}\alpha^n$ (which is easily proved by induction), where $\alpha = (1 + \sqrt{5})/2 \doteq 1.61803$ is the golden ratio, we see that if $c = 2^n$, then the minimal DFAO has at least

$$F_{n+1} \geq \frac{1}{2}\alpha^n = \frac{1}{2}c^{\frac{\log \alpha}{\log 2}} \geq c^{0.694}$$

states, for n large enough. (Note that $(\log \alpha)/(\log 2) \doteq 0.69424$.) □

The exact number of states in the minimal automaton generating $(t(i+c))_{i \geq 0}$ for small values of c is given as sequence [A382296](#) of the OEIS [25]. We state the following as an open problem.

Problem 2. *What is a good formula for the exact number of states in the minimal automaton generating $(t(i+c))_{i \geq 0}$ with msd-first input, as a function of c ?*

2.4.3 Sequences Whose Shifts Have Small State Complexity

A simple counting argument shows that if an automatic sequence $(h(i))_{i \geq 0}$ is not ultimately periodic, then the state complexity of the shifted sequence $(h(i+c))_{i \geq 0}$ cannot be $o((\log c)/(\log \log c))$. We do not know any aperiodic example, automatic or not, that achieves state complexity $O((\log c)/(\log \log c))$ for all c -shifts, but we can get almost that close.

Let the infinite word $\mathbf{p}_2 = (P_2(i))_{i \geq 0}$ be defined as follows.

$$P_2(i) = \begin{cases} 1, & \text{if } i \text{ is a power of 2;} \\ 0, & \text{otherwise.} \end{cases}$$

Theorem 26. *The number of states in the minimal automaton generating $(P_2(i+c))_{i \geq 0}$ with base-2 msd-first input is $O(\log c)$.*

Proof. We create an automaton $M = (Q, \Sigma, \delta, q_0, \Delta, \tau)$ where $\Sigma = \Delta = \{0, 1\}$ with $O(\log c)$ states generating $(P_2(i+c))_{i \geq 0}$. In order to create the automaton, we need to consider two possibilities for i where $P_2(i+c) = 1$. Let c' be $2^s - c$ where 2^s is the smallest power of 2 greater than c . For now let us ignore the case where c is a power of 2.

Case 1: $i+c = 2^j$ where $|(i)_2| \leq |(c)_2|$. In this case, the automaton reads $(c')_2$ in a series of states.

Case 2: $i+c = 2^j$ where $|(i)_2| > |(c)_2|$. In this case, the automaton reads one or more 1s and then reads $(c')_2$ padded with leading zeros so the length is the same as $(c)_2$.

Now we provide a construction for M . Let $\delta(q_0, 0) = q_0$. We first address Case 1. Let $(c') = c''_0 \cdots c''_l$. For $p \in [0, l]$, let q''_p be a state in the automaton corresponding to $c''_0 \cdots c''_p$ read and for $p \in [0, l-1]$, let $\delta(q''_p, c''_{p+1}) = q''_{p+1}$ and $\delta(q_0, 1) = q''_0$. Furthermore, let $\tau(q''_l) = 1$.

Next, we address Case 2. Let $c'_0 \cdots c'_{|(c)_2|-1}$ be $(c')_2$ padded with leading 0s so the length is $|(c)_2|$. For $p \in [0, |(c)_2|-1]$, let q'_p be a state corresponding to $c'_0 \cdots c'_p$ read and for $p \in [0, |(c)_2|-2]$, let $\delta(q'_p, c'_{p+1}) = q'_{p+1}$. Let q_1 be the state corresponding to reading the initial 1s in Case 2. So $\delta(q_1, 1) = q_1$ and $\delta(q_1, c'_0) = q'_0$. We know $c'_0 = 0$ if c is not a power of 2. Furthermore, let $\tau(q'_{|(c)_2|-1}) = 1$.

Now we need to define transitions so the automaton is deterministic and accepts $1^j c'_0 \cdots c'_{|(c)_2|-1}$ for all $j \geq 1$. If there are no 0s in $c''_0 \cdots c''_l$, let $p'' = l$; otherwise, let p'' be the first index starting from 0 where $c''_{p''+1} = 0$. We set $\delta(q''_{p''}, 1) = q_1$; this handles $j \geq p'' + 2$. For $p \in [0, p''-1]$, we set $\delta(q''_p, 0) = q'_0$; this handles $1 \leq j' \leq p''$. Let p' be the first index starting from 0 where $c''_0 \cdots c''_l$ differs from $1^{p''+1} c'_0 \cdots c'_{|(c)_2|-1}$, which can potentially be $l+1$. We set $\delta(q''_{p'-1}, c'_{p'-p''-1}) = q'_{p'-p''-1}$; this handles $j = p'' + 1$.

So if c is not a power of 2, the DFAO $M = (Q, \Sigma_2, \delta, q_0, \Delta, \tau)$ can be formally defined

as follows.

$$\begin{aligned}
Q &= \{q_0, q_1, q_0'', \dots, q_l'', q_0', \dots, q_{|(c)_2|-1}'\}, \\
\delta(q, a) &= \begin{cases} q_0, & \text{if } q = q_0, a = 0; \\ q_0'', & \text{if } q = q_0, a = 1; \\ q_1, & \text{if } q = q_1, a = 1; \\ q_0', & \text{if } q = q_1, a = c'_0; \\ q_1, & \text{if } q = q_0'', a = 1; \\ q_{p'-p''-1}', & \text{if } q = q_{p'-1}'', a = c'_{p'-p''-1}; \\ q_0', & \text{if } q = q_p'', a = 0 \text{ for } p \in [0, p'' - 1]; \\ q_{p+1}'', & \text{if } q = q_p'', a = c''_{p+1} \text{ for } p \in [0, l - 1]; \\ q_{p+1}', & \text{if } q = q_p', a = c'_{p+1} \text{ for } p \in [0, |(c)_2| - 2]; \end{cases} \\
\Delta &= \{0, 1\}, \\
\tau(q) &= \begin{cases} 1, & \text{if } q = q_l'' \text{ or } q = q_{|(c)_2|-1}'; \\ 0, & \text{otherwise.} \end{cases}
\end{aligned}$$

The correctness of M is explained above. The number of states in M is $2 + (l + 1) + (c)_2 = O(\log c)$.

If c is a power of 2, then $c = c'$ and it is easy to see that the same idea works with less complications because $|(c)_2| = |(c')_2|$. We just need to modify the automaton defined above and set $q_i'' = q_i'$ for $i \in [0, l]$, $q_1 = q_0''$, $\tau(q_0) = 1$ and omit the transitions involving p' and p'' . The number of states will remain $O(\log c)$. $\square$

Example 27. *If we create the automaton in the proof of Theorem 26 for $c = 10$, we get the DFAO in Figure 2.12 generating $(P_2(i + 10))_{i \geq 0}$. Note that in the figure, the state q_j is represented by qj .*

2.5 Construction by Büchi Arithmetic

We now re-examine the previous relations and operations in light of their implementation in an interpretation of Büchi arithmetic (as, for example, in the system **Walnut** [36, 42]) to find the time required for constructing their automata. In what follows we treat DFAO's computing automatic sequences as more or less a primitive object, without requiring an

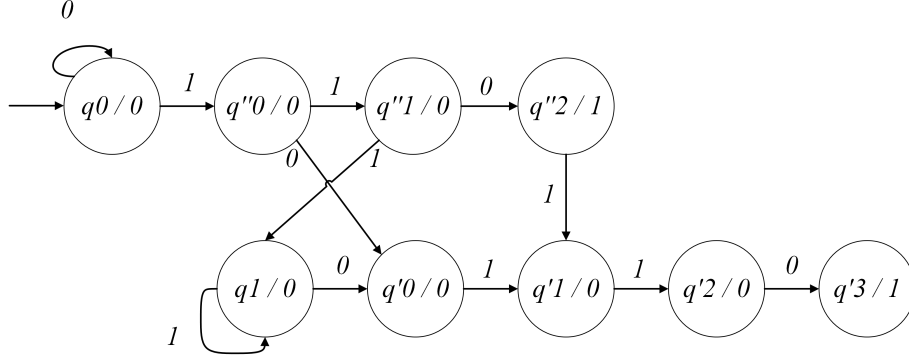


Figure 2.12: An automaton with msd-first base-2 input generating $(P_2(i + 10))_{i \geq 0}$.

explicit expression in terms of $\langle \mathbb{N}, +, V_k \rangle$. Here the total number of transitions of an automaton M as a measure for the computational complexity of constructing M .

In particular, in its current version, **Walnut** does not use the algorithms for creating the automata we discussed in the previous sections of this chapter. Instead, more general techniques, suitable for any addable numeration system, are used. This means that the size of intermediate automata may differ from the minimal automata we constructed, and the running time can similarly be larger.

Let the msd-first 2-state automaton for recognizing addition from Theorem 5 be M_{add} . We use $M_{\text{add}}(x, y, z)$ to show the output of M_{add} on input $x \times y \times z$, which is 1 if and only if $[x]_k + [y]_k = [z]_k$; we use a similar notation to refer to the output of other automata on a specific input. Recall that M_{add} states correspond to the difference $[z]_k - ([x]_k + [y]_k)$.

In **Walnut** each DFA or DFAO created is minimized. If n is the number of states in an automaton (and the number of transitions is $O(n)$), the minimization steps in the implementation can use an algorithm running in $O(n \log n)$ time by Hopcroft [23] or Valmari [45]. The algorithm by Hopcroft can be applied to DFAOs with slight modifications as described by van Spaendonck [46]. Here we sometimes only consider the effects of a partial minimization to facilitate analysis in the following theorems; however, the upper bound on state complexity and runtime complexity is correct regardless and the runtime complexity computed takes into account the computational cost of a full minimization.

We start with an analysis of how an automaton for a statement like $[x]_k = c$ is created, where c is a fixed natural number constant. Recall that in **Walnut**, constants are entered as numbers written in decimal notation (base 10). There are well-known efficient algorithms for converting a number from base 10 to base k for an integer $k \geq 2$; see, for example, [28,

§4.4], but these are not used by **Walnut**. Instead, a more general technique, suitable for any numeration system, is used.

Lemma 28. *The minimal automaton for recognizing the relation $[x]_k = c$ with msd-first input has $\lfloor \log_k c \rfloor + 2$ states.*

Proof. Such an automaton consists of a linear chain of nodes labeled with the base- k representation of c , which has length $\lfloor \log_k c \rfloor + 1$. $\square$

Theorem 29. *Let $c \geq 0$ be a fixed constant. The automaton recognizing the relation $[x]_k = c$ with msd-first input can be computed in $O((\log^2 c)(\log \log c))$ time.*

Proof. We recursively create the automaton $M_{=c}$.

The base case is $c = 0$ and the automaton has 1 state.

If c is even, we recursively compute a minimal DFA $M_{=(c/2)}$ recognizing $[y]_k = c/2$ on input y and then use the first-order expression

$$\exists y \ M_{=(c/2)}(y) \wedge M_{\text{add}}(y, y, x)$$

which is translated into an automaton. We know the automaton for M_{add} has 2 states and by Lemma 28, the automaton $M_{=(c/2)}$ has $\lfloor \log_k(c/2) \rfloor + 2$ states. First we apply the product construction to $M_{=(c/2)}$ and M_{add} such that the $M_{=(c/2)}$ part processes y and the M_{add} part processes $y \times y \times x$; the resulting automaton has $2(\lfloor \log_k(c/2) \rfloor + 2)$ states. Each state in the product automaton is of the form $[q, p]$ where on input $x \times y$, q is a state from the minimal automaton $M_{=(c/2)}$ representing the number of letters from $(c/2)_k$ matched with y by Lemma 28 and $p \in \{0, 1\}$ is a state from M_{add} equal to $[x]_k - 2[y]_k$. Then we apply the $\exists y$ quantifier by removing the component corresponding to y from the transitions and we get an NFA. In the NFA there are no two same transitions from a single state leading to different states. So after subset construction the resulting DFA is the same as the NFA. Then we do minimization and the resulting DFA is the automaton from Lemma 28 with $O(\log c)$ states.

If c is odd, we recursively compute a DFA $M_{=(c-1)}$ recognizing $[y]_k = c - 1$ on input y and then use the first-order expression

$$\exists y \ M_{=(c-1)}(y) \wedge M_{\text{add}}(y, 1, x)$$

which is translated into an automaton. Following a construction similar to the one described for the case where c is even, we get the DFA from Lemma 28 for constant c with $O(\log c)$ states.

Taking minimization steps into account, we get the following recursive formula for the time complexity:

$$T(c) = \begin{cases} O((\log c)(\log \log c)) + T(c/2), & \text{if } c \bmod 2 = 0; \\ O((\log c)(\log \log c)) + T(c-1), & \text{if } c \bmod 2 = 1. \end{cases}$$

So the total time is $O((\log^2 c)(\log \log c))$. $\square$

Theorem 30. *Let $c \geq 0$ be a fixed constant. The automaton recognizing the relation $[x]_k + c = [z]_k$ with msd-first input can be computed in $O((\log^2 c)(\log \log c))$ time.*

Proof. Consider the $O(\log c)$ state automaton $M_{=c}$ from Theorem 29 recognizing $[y]_k = c$ on input y . We have $M_{=c}(y) = 1$ if and only if $[y]_k = c$.

We can use the following expression to create the automaton M_c recognizing $[x]_k + c = [z]_k$ on input $x \times z$.

$$\exists y \ M_{\text{add}}(x, y, z) \wedge M_{=c}(y).$$

First, we apply the $\wedge$ by product construction for M_{add} and $M_{=c}$ such that the M_{add} part processes $x \times y \times z$ and the $M_{=c}$ part processes y . The resulting automaton M'_c has $O(\log c)$ states. Let $\text{Pre}_k(c, i)$ denote the first i letters in $(c)_k$. On input $x \times y \times z$ where y is $0^*y'$ and y' is a prefix of $(c)_k$, the automaton M'_c leads to a state $[q, p]$ where $q \in \{0, 1\}$ is from M_{add} and $p = |y'|$ is from $M_{=c}$ indicating how many letters from y have been matched with $(c)_k$. Therefore, we have $q = [z]_k - ([x]_k + [y]_k)$ and $[\text{Pre}_k(c, p)]_k = [y]_k$. So $[z]_k - [x]_k = q + [\text{Pre}_k(c, p)]_k$ and to each state $[q, p]$ reachable on $x \times y \times z$ in M'_c a difference $d = [z]_k - [x]_k$ can be attributed.

Then we apply the $\exists y$ quantifier by removing the component corresponding to y from the transitions and we get an NFA. In the NFA there are no two same transitions from a single state leading to different states. So after subset construction the resulting DFA is the same as the NFA. Therefore, there are $O(\log c)$ states in the DFA and to each state of the DFA reachable on $x \times z$ a difference $d = [z]_k - [x]_k$ can be attributed.

From the proof of Theorem 6, we know the automaton recognizing $[x]_k + c = [z]_k$ on input $x \times z$ only needs to keep track of a difference $[z]_k - [x]_k$ in the range $[0, c]$. So after some minimization, the resulting automaton $M_c = (Q, \Sigma_k^2, \delta, q_0, A)$ can be formally defined as follows:

$$\begin{aligned} Q &= \{d \in \mathbb{N} : d \in [0, c]\}, \\ \delta(d', [a, e]) &= kd' + e - a, \\ q_0 &= 0, \\ A &= \{c\}. \end{aligned}$$

Note that not all $d \in [0, c]$ are present in the minimized automaton.

Considering the time spent on creating $M_{=c}$ and the minimization steps, it takes $O((\log^2 c)(\log \log c))$ time to create the automaton M_c recognizing $[x]_k + c = [z]_k$ on input $x \times z$. $\square$

We continue by constructing the automaton M_n recognizing $n[x]_k = [z]_k$ on input $x \times z$ where n is a fixed constant. We have $M_n(x, z) = 1$ if and only if $n[x]_k = [z]_k$. There are several possible recursions for creating this automaton; for example, we could use the following:

- $\exists y \ M_{n-1}(x, y) \wedge M_{\text{add}}(x, y, z)$.
- $\exists y_1, y_2 \ M_{\lfloor n/2 \rfloor}(x, y_1) \wedge M_{\lceil n/2 \rceil}(x, y_2) \wedge M_{\text{add}}(y_1, y_2, z)$.

However, there is a third recursive implementation that is more efficient than the two above, and we describe it now. The base case is $n = 1$. Otherwise, either n is even or n is odd.

If n is even, we recursively compute a DFA $M_{n/2}$ recognizing $[y]_k = (n/2)[x]_k$ on input $x \times y$ and then use the first-order expression

$$\exists y \ M_{n/2}(x, y) \wedge M_{\text{add}}(y, y, z),$$

which is translated into an automaton by a direct product construction for the $\wedge$, and projection of the second coordinate to collapse transitions on y .

If n is odd, we recursively compute a DFA $M_{n-1}(x, y)$ recognizing $[y]_k = (n-1)[x]_k$ and use the expression

$$\exists y \ M_{n-1}(x, y) \wedge M_{\text{add}}(x, y, z),$$

which is similarly translated into an automaton.

In both cases, the translation can potentially generate a nondeterministic automaton (by the $\exists y$ projection). In this case, the subset construction is used to determinize the resulting automaton. Thus, at least in principle, this particular translation mechanism could take exponential time. However, we show that in this case, the subset construction does not cause an exponential blowup in the number of states.

Theorem 31. *Let $n \geq 1$ be a fixed constant. The automaton for recognizing $[y]_k = n[x]_k$ on msd-first input can be computed in $O(n \log^2 n)$ time.*

Proof. First by induction on n we prove that we can recursively create DFA M_n with msd-first input recognizing $[y]_k = n[x]_k$ with at most n states.

Base case: $n = 1$.

In this case, a simple 1-state automaton recognizes $[y]_k = [x]_k$. The single state corresponds to $[y]_k - [x]_k = 0$.

For the induction step, assume the result is true for all $n' < n$; we prove it for n . There are two cases.

Case 1: n is even.

By the induction hypothesis, we have a DFA $M_{n/2}$ with $n/2$ states corresponding to differences $[y]_k - (n/2)[x]_k$ in the range $[0, (n/2) - 1]$. In this case, we use the formula

$$\exists y \ M_{n/2}(x, y) \wedge M_{\text{add}}(y, y, z).$$

First, we create an automaton M'_n that is the product construction of $M_{n/2}$ and M_{add} . So this automaton has $2(n/2) = n$ states. Consider an input $x \times y \times z$ for M'_n leading to some state $[q, p]$ where q is from $M_{n/2}$ and p is from M_{add} . By the construction of M'_n , we know $q = [y]_k - (n/2)[x]_k$ and $p = [z]_k - ([y]_k + [y]_k)$. So we have $2q + p = [z]_k - n[x]_k$. Since $p \in \{0, 1\}$ and $q \in \{0, \dots, n/2 - 1\}$, all differences in $\{0, \dots, n - 1\}$ are present in M'_n and they are only represented by a single state. Therefore, the automaton M'_n keeps track of the difference $[z]_k - n[x]_k$ on input $x \times y \times z$. The difference $[z]_k - n[x]_k$ that the automaton keeps track of is independent of y .

Then we apply the $\exists y$ quantifier by removing the component corresponding to y from the transitions and we get an NFA. In the NFA there are no two same transitions from a single state leading to different states. So after subset construction the resulting DFA is the same as the NFA and we get the n -state automaton $M_n = (Q, \Sigma_k^2, \delta, q_0, A)$ as follows.

$$\begin{aligned} Q &= \{d \in \mathbb{N} : d \in [0, n - 1]\}, \\ \delta(d, [a, e]) &= kd + e - na, \\ q_0 &= 0, \\ A &= \{q_0\}. \end{aligned}$$

Projecting away the y component from transitions could have caused nondeterminism, but this is not the case.

Case 2: n is odd.

By the induction hypothesis, we have a DFA M_{n-1} with $n - 1$ states corresponding to $[y]_k - (n - 1)[x]_k$ in the range $[0, n - 2]$. In this case, we use the formula

$$\exists y \ M_{n-1}(x, y) \wedge M_{\text{add}}(x, y, z).$$

First, we create an automaton M'_n from the usual product construction for M_{n-1} and M_{add} . So this automaton has $2(n - 1)$ states. Consider an input $x \times y \times z$ for M'_n leading to some state $[q, p]$ where q is from M_{n-1} and p is from M_{add} . By the construction of M'_n , we know $q = [y]_k - (n - 1)[x]_k$ and $p = [z]_k - ([x]_k + [y]_k)$. So we have $q + p = [z]_k - n[x]_k$. Therefore, to each state in the automaton M'_n reachable on $x \times y \times z$, a difference $d = [z]_k - n[x]_k$ can be attributed. Furthermore, since $p \in \{0, 1\}$ and $q \in \{0, \dots, n - 2\}$, all differences in $\{0, \dots, n - 1\}$ are present in M'_n . Note that the difference 0 corresponds to exactly one pair—namely $[0, 0]$ —and the difference $n - 1$ corresponds to only one pair—namely $[n - 2, 1]$. All the other differences d (those in $\{1, 2, \dots, n - 2\}$) are represented by two states—namely, $[d - 1, 1]$ and $[d, 0]$.

Next, we project away the y component. The resulting automaton is an NFA, and we use the subset construction to create a DFA. Consider a state from M_n reachable on $x \times z$. This state consists of some pairs $[q, p]$ from the NFA such that $[z]_k - n[x]_k = q + p$. As mentioned above, if $[z]_k - n[x]_k$ is 0 or $n - 1$, then there is only one such pair; otherwise, there are two such pairs. So the DFA has $O(n)$ states and the determinization did not cause a blow up in the number of states.

From the proof of Theorem 9, we know the automaton recognizing $n[x]_k = [z]_k$ on input $x \times z$ only needs to keep track of the difference $[z]_k - n[x]_k$ in the range $[0, n - 1]$. So after some minimization, the DFA $M_n = (Q, \Sigma_k^2, \delta, q_0, A)$ can be formally defined as follows.

$$\begin{aligned} Q &= \{d \in \mathbb{N} : d \in [0, n - 1]\}, \\ \delta(d, [a, e]) &= kd + e - na, \\ q_0 &= 0, \\ A &= \{q_0\}. \end{aligned}$$

Let $T(n)$ be the time used by the algorithm above to create M_n . We now analyze $T(n)$ based on the information above.

In Case 1, the number of transitions traversed for creating M_n is as follows: $O(1)$ transitions for M_{add} , $O(n)$ transitions for the product construction of $M_{n/2}$ and M_{add} , $O(n)$ transitions for projecting away the y component, $O(n)$ transitions for the determinized automaton, giving a total of $O(n)$ transitions in addition to the number of transitions traversed to create $M_{n/2}$.

In Case 2, the number of transitions traversed for creating M_n is as follows: $O(1)$ transitions for M_{add} , $O(n)$ transitions for the product construction of M_{n-1} and M_{add} , $O(n)$ transitions for projecting away the y component, $O(n)$ transitions for the determinized automaton, giving a total of $O(n)$ transitions in addition to the number of transitions to create M_{n-1} .

Putting everything together and taking minimization steps into account, we get the following recursive formula:

$$T(n) = \begin{cases} O(n \log n) + T(n/2), & \text{if } n \bmod 2 = 0; \\ O(n \log n) + T(n-1), & \text{if } n \bmod 2 = 1. \end{cases}$$

So we have $T(n) = O(n \log^2 n)$. □

Theorem 32. *Let $n \geq 1$, $0 \leq c < n$ be fixed constants. The automaton recognizing the relation $n[x]_k + c = [z]_k$ with msd-first input can be computed in $O(n \log(n) \log(n \log(n)))$ time.*

Proof. Now we want to create an automaton $M_{n,c}$ recognizing $n[x]_k + c = [z]_k$.

Let M_n be the automaton from Theorem 31 for constant n and let M_c be the automaton from Theorem 30 for constant c . We have $M_n(x, y) = 1$ if and only if $n[x]_k = [y]_k$ and $M_c(y, z) = 1$ if and only if $[y]_k + c = [z]_k$. So we can use the following expression to create $M_{n,c}$:

$$\exists y \ M_n(x, y) \wedge M_c(y, z).$$

We first apply the product construction to M_n and M_c . The resulting automaton $M'_{n,c}$ has $O(n \log c) = O(n \log n)$ states.

Consider an input $x \times y \times z$ for $M'_{n,c}$ leading to some state $[q, p]$ where q is from M_n and p is from M_c . By the construction of $M'_{n,c}$, we know that $q = [y]_k - n[x]_k$ and $p = [z]_k - [y]_k$. So we have $q + p = [z]_k - n[x]_k$. Therefore, to each state in the automaton $M'_{n,c}$ reachable on $x \times y \times z$ a difference $d = [z]_k - n[x]_k$ can be attributed. Furthermore, since $q \in \{0, \dots, n-1\}$ and $p \in \{0, \dots, c\}$, the attributed differences are in the range $[0, n + c - 1]$. If we choose a difference $d \in [0, n + c - 1]$, then we can exactly tell what combinations of $[q, p]$ correspond to d (if any) and no $[q, p]$ corresponds to more than one d .

Consider all the states $[0, d], [1, d-1], \dots, [d, 0]$ corresponding to the difference d , assuming all these states exist in $M'_{n,c}$. If one of these states $[q, d-q]$ is reached on input $x \times y \times z$, then the state $[q+i, d-q-i]$ is reached on input $x \times ([y]_k + i) \times z$. Recall that

we established we create automata so that they have the correct output regardless of the number of leading zeros read.

Next, we project away the y component from $M'_{n,c}$. The resulting NFA is determinized by subset construction to get a DFA. Consider a state from the DFA reachable on $x \times z$. This state consists of some $[q, p]$ from the NFA such that $q + p = [z]_k - n[x]_k$. Based on the discussion above, there are $d = O(n+c)$ states in the DFA. So the DFA has $O(n+c) = O(n)$ states.

From the proof of Theorem 9, we know the automaton recognizing $n[x]_k + c = [z]_k$ on input $x \times z$ for $0 \leq c < n$ only needs to keep track of a difference $[z]_k - n[x]_k$ in the range $[0, n-1]$. So after some minimization, the DFA $M_{n,c} = (Q, \Sigma_k^2, \delta, q_0, A)$ can be formally defined as follows.

$$\begin{aligned} Q &= \{d \in \mathbb{N} : d \in [0, n-1]\}, \\ \delta(d, [a, e]) &= kd + e - na, \\ q_0 &= 0, \\ A &= \{c\}. \end{aligned}$$

and $M_{n,c}$ has $O(n)$ states. Taking into account the minimization steps and the time spent on creating M_n and M_c based on the proof of Theorems 31 and 30 and considering $c = O(n)$, the time spent on creating $M_{n,c}$ is $O(n \log(n) \log(n \log(n)))$. $\square$

Theorem 33. *Let $n \geq 1$, $0 \leq c < n$ be fixed constants. For msd-first input, given an m -state DFAO generating $(h(i))_{i \geq 0}$, we can compute the DFAO generating $(h(ni + c))_{i \geq 0}$ in $O(n \log n \log(n \log n) + nm^2 \log(nm^2))$ time.*

Proof. We create the DFAO for $(h(ni + c))_{i \geq 0}$. We are given a DFAO $M_h = (Q_h, \Sigma_k, \delta_h, q_{0,h}, \Delta_h, \tau_h)$ for $(h(i))_{i \geq 0}$ with m states and we have $M_h(y) = h([y]_k)$. Let $M_{n,c}$ be the automaton from the proof of Theorem 32 where $M_{n,c}(x, y)$ is 1 if and only if $n[x]_k + c = [y]_k$.

We first use the product construction for $M_{n,c}$ and M_h such that the $M_{n,c}$ part processes the input $x \times y$ and the M_h part processes the input y to get the automaton M' . Consider a state $[q, p]$ reachable on $x \times y$ in M' where q is from $M_{n,c}$ and p is from M_h . We have $q = [y]_k - n[x]_k \in [0, n-1]$ and $p = \delta_h(q_{0,h}, y)$. This automaton has nm states.

Next, we project the y component away and use subset construction to construct a DFAO. On input x the automaton reaches a state consisting of some pairs $[q, p]$ where $q \in [0, n-1]$ and $p = \delta_h(q_{0,h}, (n[x]_k + q)_k)$. The q are all values in the range $[0, n-1]$

and the p values are length- n subwords of $(h'(i))_{i \geq 0}$. Furthermore, the output of the state is $\tau_h(p)$ where p is from the state $[c, p]$ of M' . So based on Theorem 4, there are $O(nm^2)$ states in the automaton.

Taking into account the minimization steps and the time spent on creating $M_{n,c}$ based on the proof of Theorem 32, creating M given M_h takes $O(n \log n \log(n \log n) + nm^2 \log(nm^2))$ time. $\square$

Theorem 34. *For msd-first input, given the m -state DFAO generating $(h(i))_{i \geq 0}$, the DFAO generating $(h(i+j))_{i,j \geq 0}$ is computed in $O(m^2 \log m)$ time.*

Proof. Let $M_h = (Q_h, \Sigma_k, \delta_h, q_{0,h}, \Delta_h, \tau_h)$ be the m -state DFAO generating the sequence $(h(i))_{i \geq 0}$; that is, $M_h(z) = h([z]_k)$.

We first apply the product construction to M_{add} and M_h such that the M_{add} part works on input $x \times y \times z$ and the M_h part works on input z to get the automaton M' . Consider a state $[q, p]$ reachable on $x \times y \times z$ in M' where q is from M_{add} and p is from M_h . We have $q = [z]_k - ([x]_k + [y]_k)$ and $p = \delta_h(q_{0,h}, z)$. This automaton has $2m$ states.

Next, we project the z component away and use the subset construction to construct a DFAO. On input $x \times y$, the DFAO reaches a state consisting of some $[q, p]$ where $q \in \{0, 1\}$ and $p = \delta_h(q_{0,h}, (q + [x]_k + [y]_k)_k)$. The p values are length-2 subwords of $(h'(i))_{i \geq 0}$. So the DFAO has $O(m^2)$ states.

Taking minimization steps into account, creating M given M_h takes $O(m^2 \log m)$ time. $\square$

2.6 Conclusion

In this chapter we studied various automata recognizing basic relations or carrying out operations on automatic sequences where the automata have input in base k . Furthermore, we answered an open question of Zantema and Bosma [51] in Theorem 13 and Corollary 14.

In Chapter 3, we address similar topics with input in Fibonacci numeration system. In Section 4.2, we introduce more open problems related to the current chapter.

Chapter 3

Automata With Input in Fibonacci Representation

This chapter includes content taken nearly verbatim or verbatim from the arXiv paper [\[34\]](#) co-authored with Narad Rampersad and Jeffrey Shallit.

3.1 Introduction

In this chapter, we discuss the state complexity of specific arithmetic relations (such as addition and multiplication) when the inputs are given in Fibonacci representation (also known as Zeckendorf representation). We also study the state complexity of shifts and linear subsequences of Fibonacci-automatic sequences, and the computational complexity of forming the automata using an interpretation of Büchi arithmetic. In the previous chapter we studied the same questions for k -automatic sequences, but the case of Fibonacci representation presents many new and different challenges.

In Section [3.2](#), we provide the necessary background for the rest of the chapter. In Section [3.3](#), we construct automata of $O(\log c)$ states recognizing $Y = X + c$ and $Y = X - c$ where X, Y are inputs, and $c \geq 0$ is a constant. We construct an automaton of $O(n^2)$ states recognizing the relation $Y = nX + c$ where $n \geq 1$ is a natural number, $0 \leq c < n$, and the numbers X and Y are inputs. In Section [3.4](#), we show that if $(h(i))_{i \geq 0}$ is a fixed Fibonacci-automatic sequence generated by an m -state automaton, then the shifted subsequence $(h(i + c))_{i \geq 0}$ can be generated by a DFAO of $O(m^2 c^2)$ states, the linear subsequence $(t(i + c))_{i \geq 0}$ of the Thue-Morse sequence can be generated by a DFAO of $O(c)$

states, and the linear subsequence $(h(ni + c))_{i \geq 0}$ for $n \geq 1$ and $0 \leq c < n$ can be generated by a DFAO of $O(m^2 n^4)$ states. As a consequence, we improve a bound in a recent paper by Bosma and Don [9] concerning the size of morphisms for linear subsequences of the Fibonacci word. In Section 3.5, we show how the automaton generating $(h(ni + c))_{i \geq 0}$ for fixed $n \geq 1, 0 \leq c < n$ is created in polynomial time using an interpretation of Büchi arithmetic.

3.2 Background

3.2.1 Finite Automata

A DFAO is called a *Fibonacci-DFAO* if its input(s) are in Fibonacci representation (see Section 3.2.2). A Fibonacci-DFAO $M = (Q, \Sigma, \Delta, \delta, q_0, \tau)$ generates a *Fibonacci-automatic sequence* $(h(i))_{i \geq 0}$ in the following way: the input is an msd-first representation x of i in the Fibonacci numeration system, and the output $h(i)$ is $\tau(\delta(q_0, x))$. We assume that such a DFAO does not reach any state on an input containing an invalid Fibonacci representation.

We will need a lemma about the subword complexity (also called “factor complexity”) of an infinite word $\mathbf{x}$ generated by a Fibonacci-automatic sequence. The subword complexity of $\mathbf{x}$ is a function $\rho_{\mathbf{x}}(n)$ counting the number of distinct subwords of length n appearing in $\mathbf{x}$.

Lemma 35. *We have $\rho_{\mathbf{x}}(n) = O(n)$. Let $\mathbf{x}$ be Fibonacci-automatic sequence generated by an m -state DFAO with msd-first input. Then $\rho_{\mathbf{x}}(n) = O(nm^2)$ for all $n \geq 1$.*

Proof. See [3, Theorem 10.3.1]. The proof is stated there for base- k , but the same proof works for Fibonacci-automatic sequences. $\square$

We also need the unfamiliar notion of unambiguous finite automaton with output (UFAO), which is a blend of the notion of unambiguous finite automaton (UFA) with that of the DFAO. The UFAO is a nondeterministic machine where there is at most one output for every input. The output associated with an input is the output associated with the unique accepting state reached. The usual subset construction can be used to turn a UFAO into a DFAO, but at the risk of exponential blow-up in the number of states. UFA’s are discussed in, among many other places, Jirásek et al. [26].

The integer	The Fibonacci representation
0	λ
1	1
2	10
3	100
4	101
5	1000
6	1001
7	1010
8	10000
9	10001
10	10010

Table 3.1: The canonical Fibonacci representation of integers 0 to 10.

3.2.2 Fibonacci Representation

We let $\mathbb{N} = \{0, 1, 2, \dots\}$. Define the Fibonacci numbers, as usual, by $F_0 = 0$, $F_1 = 1$, and $F_n = F_{n-1} + F_{n-2}$ for $n \geq 0$.

Fibonacci representation was introduced by Lekkerkerker [30] and Zeckendorf [52], although it can be found in an earlier, much more general form in a paper of Ostrowski [38]. The Fibonacci representation of $n \in \mathbb{N}$ writes n as a sum of distinct Fibonacci numbers F_i for $i \geq 2$; that is, $n = \sum_{2 \leq i \leq t} e_i F_i$ for $e_i \in \{0, 1\}$. To ensure uniqueness of the representation, no two consecutive Fibonacci numbers can be used; that is, in a valid representation we have $e_i \cdot e_{i+1} = 0$ for $1 \leq i \leq t$. A representation can be associated with the binary word of its coefficients $e_t \cdots e_2$; note that representations are written with the most significant bit at the left. We need a way to convert an arbitrary binary word $x = e_t \cdots e_2$ to the integer it represents. For this we write $[x] = \sum_{1 \leq i \leq t} e_i F_i$. For example, we have $[01001] = 6$. The canonical Fibonacci representation of an integer i is the word x , without leading zeros, such that $[x] = i$. For this we use the notation (i) . Table 3.1 shows the canonical Fibonacci representation of integers 0 to 10. We say a word z is a *valid right extension* of a word x if x is a prefix of z and z contains no occurrence of 11.

For the rest of this chapter, we assume that the inputs to automata are in Fibonacci representation and given most significant digit first format, unless otherwise stated. The input alphabet is $\Sigma_2 = \{0, 1\}$ and it represents the digits of the representation of some integer i .

The Fibonacci word $\mathbf{f} = f_0 f_1 f_2 \cdots = 01001010 \cdots$ is an infinite binary word that can

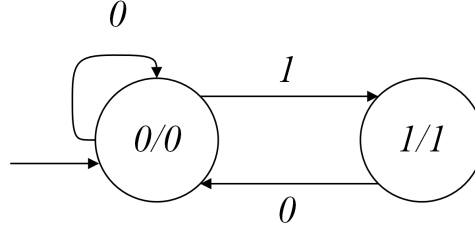


Figure 3.1: An automaton generating the Fibonacci word $\mathbf{f}$.

be defined in a number of equivalent ways [7]. The simplest for us is that f_i is the last bit of the Fibonacci representation of i . It is also defined as the fixed point of the morphism $0 \rightarrow 01, 1 \rightarrow 0$. It can be generated by a DFAO of 2 states, as illustrated in Figure 3.1.

We now state some facts about Fibonacci representation. The first is easy and the second is mentioned by Reble [39], and the third is a consequence of the Fibonacci representation being given by the greedy algorithm. Let $\alpha = (1 + \sqrt{5})/2$ and $\beta = (1 - \sqrt{5})/2$, the zeros of the polynomial $X^2 - X + 1$.

Lemma 36.

- (a) Let $x \in \Sigma_2^*$. Then $[x00] = [x] + [x0]$.
- (b) Let x be a valid Fibonacci representation. Then $-\beta^2 < [x0] - \alpha[x] < -\beta$.
- (c) Let x, y be valid Fibonacci representations. Then $x < y$ in lexicographic order iff $[x] < [y]$.

Next we state a lemma that we will use in Section 3.4.

Lemma 37. Suppose x is a word, $a \in \{0, 1\}$ and xa is a valid Fibonacci representation. Then $[x] = \lfloor ([xa] + 2)/\alpha \rfloor - 1$.

Proof.

Case 1: $a = 0$. Then by Lemma 36 (b), we have

$$\alpha[x] - \beta^2 < [xa] < \alpha[x] - \beta.$$

Add 2 and divide by α to get

$$[x] + 1 < \frac{[xa] + 2}{\alpha} < [x] + \alpha < [x] + 2.$$

The result now follows.

Case 2: $a = 1$. Then, since xa is a valid Fibonacci representation, we must have $x = x'0$. Following the argument in Reble [39], let $x' = e_t \cdots e_2$ so that $[x'] = \sum_{2 \leq i \leq t} e_i F_i$. Then

$$\begin{aligned} [x'00] &= \sum_{2 \leq i \leq t} e_i F_{i+2} \\ &= \sum_{2 \leq i \leq t} e_i (\alpha F_{i+1} + \beta^{i+1}) \\ &= \alpha \sum_{2 \leq i \leq t} e_i F_{i+1} + \sum_{2 \leq i \leq t} \beta^{i+1} \\ &= \alpha[x'0] + \sum_{2 \leq i \leq t} \beta^{i+1}. \end{aligned}$$

Hence

$$-\beta^2 = \beta^3 + \beta^5 + \beta^7 + \cdots < [x'00] - \alpha[x'0] < \beta^4 + \beta^6 + \beta^8 + \cdots = \sqrt{5} - 2.$$

Now, adding $3 + \alpha[x'0]$ to both sides, we get $\alpha[x'0] + \alpha^2 < [x'01] + 2 < \alpha[x'0] + \sqrt{5} + 1$, or in other words, $\alpha[x] + \alpha^2 < [xa] + 2 < \alpha[x] + \sqrt{5} + 1$. Dividing through by α gives $[x] + 1 < [x] + \alpha < \frac{[xa] + 2}{\alpha} < [x] + 2$, which proves the result for this case. $\square$

We also need the following lemma.

Lemma 38. *Suppose x, y are two words of the same length, and $a, a', b, b' \in \Sigma_2$ and $D(x, y) = [y] - n[x]$ for some $n \geq 1$. Then*

$$D(xa'a, yb'b) = D(x, y) + D(xa', yb') + b - na + b' - na'.$$

Proof. Lemma 36 (a) gives us that $[x00] = [x] + [x0]$. Thus $[xa'a] = [x] + [x0] + 2a' + a$ and $[yb'b] = [y] + [y0] + 2b' + b$. Hence

$$\begin{aligned} D(xa'a, yb'b) &= [yb'b] - n[xa'a] \\ &= [y] + [y0] + 2b' + b - n([x] + [x0] + 2a' + a) \\ &= ([y] - n[x]) + ([yb'] - n[xa']) + b' - na' + b - na \\ &= D(x, y) + D(xa', yb') + b' - na' + b - na. \end{aligned}$$

□

3.3 Recognizing Relations

In this section we study the state complexity of recognizing some simple relations; for example the state complexity of the automaton accepting $x \times y$ if and only if $[x] + c = [y]$.

3.3.1 Addition

Theorem 39. *For all integers $c \geq 0$, there exists an msd-first Fibonacci automaton with at most $O(\log c)$ states accepting $x \times y$ where $[x] + c = [y]$.*

Proof. We assume $c \geq 1$, since the case $c = 0$ is trivial. We design a Fibonacci automaton M_c that reads input $x \times y$ in parallel and accepts if and only if $[y] = [x] + c$. We modify the same approach we used previously for the case of base- k representation [35]. Namely, we first find a range of values I containing c such that if $[y] - [x]$ ever falls outside I , then $[y'] - [x']$ stays outside I for all right extensions x' of x and y' of y .

Suppose $[y] < [x]$. By Lemma 36 (c) we see that $y < x$ under the lexicographic order. Hence, if y' is a right-extension of y and x' is a right-extension of x , we have $y' < x'$ under the lexicographic order, and so $[y'] < [x']$.

Now suppose $[y] > [x] + c$. We have

$$\begin{aligned} [yb] - [xa] &= [y0] - [x0] + b - a > (\alpha[y] - \beta^2) - (\alpha[x] - \beta) - 1 \\ &= \alpha([y] - [x]) - \beta^2 + \beta - 1 \geq \alpha(c + 1) - \beta^2 + \beta - 1 = \alpha c - \beta^2 > c. \end{aligned}$$

Therefore, we can take the range I to be $[0, c]$.

Our automaton $M_c = (Q, \Sigma_2, q_0, \delta, A)$ is constructed as follows.

$$\begin{aligned} Q &\subseteq \{[a, b, d', d] : a, b \in \{0, 1\} \text{ and } d, d' \in I\}, \\ q_0 &= [0, 0, 0, 0], \\ \delta([a', b', d'', d'], [a, b]) &= [a, b, d', d] \text{ where } a, b, a', b \in \{0, 1\}, aa', bb' \neq 11, \text{ and} \\ &\quad d = d' + d'' + b' - a' + b - a, \\ A &= \{[a, b, d', d] : a, b \in \{0, 1\} \text{ and } d = c\}. \end{aligned}$$

The correctness of the transition rule now follows from Lemma 38 (with $n = 1$) and an easy induction on the length of the input.

Our next task is to define Q more precisely. Define $d' = [y] - [x]$, $d = [yb] - [xa]$, $\varepsilon_x = [x0] - \alpha[x]$, and $\varepsilon_y = [y0] - \alpha[y]$. Thus

$$\begin{aligned} d &= [yb] - [xa] = [y0] - [x0] + b - a \\ &= \alpha[y] + \varepsilon_y - \alpha[x] - \varepsilon_x + b - a = \alpha d' + \varepsilon_y - \varepsilon_x + b - a. \end{aligned}$$

Rewriting, we get

$$d' - d/\alpha = \frac{\varepsilon_x - \varepsilon_y + a - b}{\alpha}. \quad (3.1)$$

There are now three cases to consider. By Lemma 36 (b) we know $-\beta^2 < \varepsilon_x, \varepsilon_y < -\beta$. Hence the cases are as follows.

Case 1: $a = b$. Then $d/\alpha - 1 < d' < d/\alpha + 1$.

Case 2: $a = 0, b = 1$. Then $d/\alpha - 2 < d' < d/\alpha$.

Case 3: $a = 1, b = 0$. Then $d/\alpha < d' < d/\alpha + 2$.

This gives eight possible forms of states corresponding to a particular d :

$$\begin{aligned} &\{[0, 0, \lfloor d/\alpha \rfloor, d], [0, 0, \lceil d/\alpha \rceil, d], [1, 1, \lfloor d/\alpha \rfloor, d], [1, 1, \lceil d/\alpha \rceil, d], \\ &[0, 1, \lfloor d/\alpha \rfloor, d], [0, 1, \lfloor d/\alpha \rfloor - 1, d], [1, 0, \lceil d/\alpha \rceil, d], [1, 0, \lceil d/\alpha \rceil + 1, d]\}. \end{aligned}$$

Note that not all 8 possibilities are necessarily accessible.

Starting from a final state, of the form $[a, b, d', c]$, we determine the possible predecessors of this state, and the possible predecessors of those, etc. Note that by Equation (3.1) if $[y] - [x] \geq 4$, then for any right extension of x, y the difference only increases. We group the predecessors together in levels. The states of the form $[a, b, d', c]$ are at level 0, and the predecessors of these are at level 1, and so forth. For examples, Figure 3.2 shows part of an automaton constructed for $c = 40$ including level 0 and part of level 1. We now have to bound the number of states in each level, and the number of levels.

States in level i are of the form $[a, b, d', d]$ and $d \in [D_i, D_i + r_i]$ for certain D_i, r_i . Note that $D_0 = c$ and $r_0 = 0$. Since the smallest possible d' in a level i state is $\lfloor D_i/\alpha \rfloor - 1$ and the largest possible is $\lceil (D_i + r_i)/\alpha \rceil + 1$, it follows that $r_{i+1} < r_i/\alpha + 4$. Starting with $r_0 = 0$, using this inequality four times shows that $r_i \leq 8$ for all i . Thus there are at most $8 + 1 = 9$ possible values of d among all states of any given level.

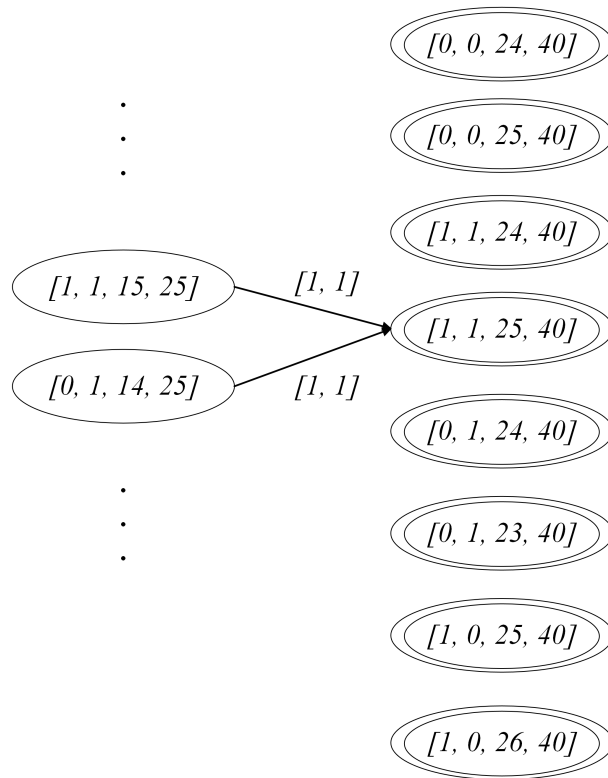


Figure 3.2: Part of an automaton accepting $x \times y$ if and only if $[x] + 40 = [y]$.

To estimate the number of possible levels, note that Equation (3.1) gives $d' < (d+2)/\alpha$ and thus $D_{i+1} + r_{i+1} < (D_i + 10)/\alpha < D_i/1.1$ provided $D_i \geq 22$. Thus there are at most $\log_{1.1} c = O(\log c)$ levels until $D_i \leq 22$. When the largest d in a level is smaller than 22, we cease our computation of levels and simply include all states $[a, b, d', d]$ with $d \leq 22$ (a constant number).

This shows that there are only $O(\log c)$ co-accessible states and we are done. $\square$

A similar proof can be provided for $[x] - c = [y]$.

Theorem 40. *For all $c \geq 0$, there exists an automaton with at most $O(\log c)$ states accepting $x \times y$ where $[x] - c = [y]$.*

Proof. Consider the automaton provided for the proof of Theorem 39. Here the only difference is that we keep track of $[x] - [y]$ instead of $[y] - [x]$. So we only need to change the transition function to the following:

$$\delta([a', b', d, d'], [a, b]) = [a, b, g, d], \text{ if } g \in I \text{ where } g = d + d' + a - b + a' - b' \text{ and } aa', bb' \neq 11.$$

$\square$

3.3.2 Multiplication

Let $n \geq 1$ be a natural number and $0 \leq c < n$. We provide an upper bound on the number of states required for an automaton to recognize the relation $Y = nX + c$.

We let $L_{n,c}$ be the language of all words over $\Sigma_2 \times \Sigma_2$ where the first component represents some integer $[x]$ and the second component represents $n[x] + c$. More precisely,

$$L_{n,c} := \{x \times y \in (\Sigma_2 \times \Sigma_2)^* : [y] = n[x] + c\}.$$

For example, Figure 3.3 depicts a minimal DFA recognizing $L_{2,0}$ computed by Walnut. Here the state numbered 0 is the initial state (denoted by the headless arrow) and the accepting states 0, 3, 6, and 8 are indicated by double circles.

Theorem 41. *There is a DFA of $O(n^2)$ states that accepts $L_{n,c}$.*

A result related to Theorem 41 was proved by Charlier et al. [16]. They proved that $2n^2$ states are necessary and sufficient for a DFA to accept multiples of the constant n with

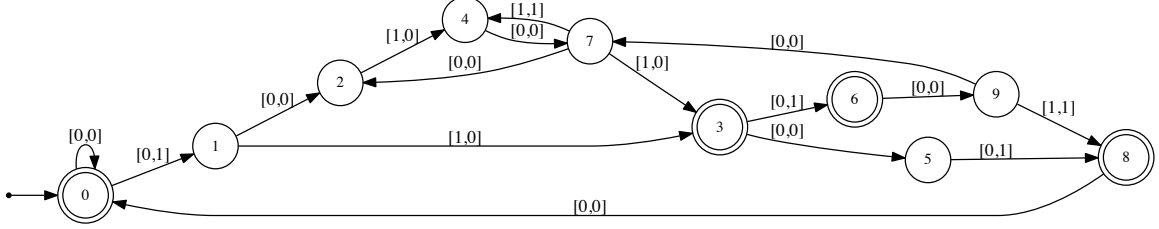


Figure 3.3: An automaton with input in Fibonacci representation accepting $x \times y$ if and only if $2[x] = [y]$.

input in the Fibonacci numeration system. However, despite the close relationship of their result and ours, we see no direct way to get our result from theirs.

For input in Fibonacci representation, we use the same strategy outlined for recognizing this relation with input in base k from [35]. Namely, we use an automaton $M_{n,c}$ to keep track of the difference function $D(x, y) := [y] - n[x]$. Here there are complications that arise from four things:

- In base k , reading a single letter a from the input changes the integer read from n to $kn + a$. But the formula is more involved for Fibonacci representation.
- It is no longer sufficient to maintain just a single d , the current difference. We also need to maintain another related quantity, d' , the previous difference.
- We have to ensure that the inputs are valid Fibonacci representations. This means we have to store, in the state, the last letter read for both input words.
- It is no longer so simple to determine the interval I_n for d and d' , outside of which no right extension would be accepted.

As we mentioned, the idea is that on input $xa' \times yb'$, the automaton should keep track of $d = D(xa', yb')$. However, considering Lemma 38, this would not be sufficient; we also need to keep track of $d' = D(x, y)$. Moreover, we need to determine the interval I_n so that if d ever falls outside this interval, we know that no right extension of the input can ever lie in L_n . Thus states are 4-tuples of the form $[a', b', d, d'] \in \Sigma_2 \times \Sigma_2 \times I_n \times I_n$.

Assuming the existence of I_n , the definition of the transition function now follows immediately from Lemma 38. Namely, $M_{n,c} = (Q, \Sigma_2, q_0, \delta, A)$ is defined as follows:

$$\begin{aligned}
Q &= \{[a', b', d, d'] : d, d' \in I_n \text{ and } a', b' \in \Sigma_2\}, \\
\Sigma &= \Sigma_2 \times \Sigma_2, \\
q_0 &= [0, 0, 0, 0], \\
\delta([a', b', d, d'], [a, b]) &= [a, b, g, d], \text{ if } g \in I_n \text{ where } g = d + d' + b - na + b' - na' \\
&\quad \text{and } aa', bb' \neq 11; \\
A &= \{[a', b', d, d'] : d = c\}.
\end{aligned}$$

So task that remains is to determine the interval I_n . We show that we can take I_n to be the closed interval $[-n, 2n - 1]$.

Lemma 42. *Suppose $D(x, y) \leq -n - 1$. Then $D(xa, yb) \leq -n - 1$.*

Proof. We have

$$\begin{aligned}
D(xa, yb) &= [yb] - n[xa] \\
&= [y0] - n[x0] + b - na \\
&< (\alpha[y] - \beta) - n(\alpha[x] - \beta^2) + 1 \\
&= \alpha D(x, y) - \beta + n\beta^2 + 1 \\
&\leq \alpha(-n - 1) - \beta + n\beta^2 + 1 \\
&= -n(\alpha - \beta^2) + 1 - \alpha - \beta \\
&= -n(\alpha - \beta^2) \\
&< -n \\
&\leq -n - 1.
\end{aligned}$$

□

Lemma 43. *Let $a, b \in \Sigma_2$, and suppose $D(x, y) \geq 2n$. Then $D(xa, yb) \geq n$.*

Proof. We have

$$\begin{aligned}
D(xa, yb) &= [yb] - n[xa] \\
&= ([y0] + b) - n([x0] + a) \\
&\geq [y0] - n[x0] - n \\
&> (\alpha[y] - \beta^2) - n(\alpha[x] - \beta) - n \\
&= \alpha([y] - n[x]) - \beta^2 + n\beta - n \\
&= \alpha D(x, y) - \beta^2 + n\beta - n \\
&\geq 2n\alpha - \beta^2 + n\beta - n \\
&= (2\alpha + \beta - 1)n - \beta^2 \\
&> n - 1.
\end{aligned}$$

□

Lemma 44. *Let $a, b, a', b' \in \Sigma_2$ such that either $a = 0$ or $a' = 0$. Suppose $D(x, y) \geq n$ and $D(xa', yb') \geq n$. Then $D(xa'a, yb'b) \geq n$.*

Proof. From Lemma 38 we have

$$\begin{aligned}
D(xa'a, yb'b) &= D(x, y) + D(xa', yb') + b - na + b' - na' \\
&\geq 2n - n(a + a') \\
&\geq n.
\end{aligned}$$

Note that we know that at least one of a, a' equals 0, for otherwise, we do not have a valid Fibonacci representation. □

Theorem 45. *We can take $I_n = [-n, 2n - 1]$.*

Proof. Lemma 42 shows that if $D(x, y) \leq -n - 1$, then the same inequality holds for every right extension of x and y .

Lemmas 43 and 44 together show that if $D(x, y) \geq 2n$, then $D(x', y') \geq n$ holds for every right extension x' of x and y' of y . □

Now a simple induction on $|x| = |y|$ shows that

$$\delta(q_0, xa' \times yb') = [a', b', D(xa', yb'), D(x, y)],$$

provided that no prefix $x' \times y'$ of $xa' \times yb'$ has $D(x', y') \notin I_n$.

We can now prove Theorem 41.

Proof of Theorem 41. The automaton $M_{n,c}$ we have constructed consists of states of the form $[a', b', d, d']$ where $d, d' \in I_n$ and $a', b' \in \Sigma_2$. This gives a total of $2 \cdot 2 \cdot 3n \cdot 3n$ states, which is $O(n^2)$. $\square$

Remark 46. We point out that the DFA $M_{n,c}$ in our construction is not, in general, minimal. However, we can construct our automaton in quadratic time and then apply a well-known minimization algorithm for DFAs by Hopcroft [24] or Valmari [45] to get the minimal automaton recognizing $Y = nX + c$ in $O(n^2 \log n)$ time.

The number of states in the minimal DFA recognizing the language $L_{n,0}$ forms sequence [A372846](#) in the On-Line Encyclopedia of Integer Sequences (OEIS) [25]. The values for $1 \leq n \leq 10$ (not counting the dead state), as computed by the free software **Walnut**, are 2, 10, 23, 40, 59, 85, 114, 146, 181, 224. Numerical evidence suggests that this quantity is roughly $2n^2 + \alpha^2 n$, but we do not know how to prove this. A better understanding of the reachable states of $M_{n,c}$ might help reduce the bound.

3.4 Linear Subsequences

A sequence $(h(i))_{i \geq 0}$ over a finite alphabet is said to be *Fibonacci-automatic* if it is generated by a DFAO M_h reading an integer input i in Fibonacci representation, and returning the value of $h(i)$ as the output associated with the last state reached. More precisely, if $M_h = (Q_h, \Sigma_2, \Delta_h, q_{0,h}, \delta_h, \tau_h)$, then $h(i) = \tau_h(\delta_h(q_{0,h}, x))$ for all valid representations $x \in \Sigma_2^*$ with $[x] = i$. We assume that there are no transitions corresponding to two consecutive inputs of 1; in other words, if there is a state q reachable on 1 from some other state, then $\delta_h(q, 1)$ is undefined. For more information about such sequences, see [37].

Theorem 47. *Let $(h(i))_{i \geq 0}$ be a Fibonacci-automatic sequence generated by a DFAO of m states. For a constant $c \geq 0$, there is a DFAO of $O(m^2 c^2)$ states generating $(h(i + c))_{i \geq 0}$.*

Proof. Let $(h'(i))_{i \geq 0}$ be the interior sequence of $(h(i))_{i \geq 0}$ generated by $M = (Q, \Sigma_2, q_0, \delta, \Delta, \tau)$. We use a method similar to the one used for base- k in [35] and create an automaton M' that on input x reaches a state of the form $[[h'([x]), a'_0], \dots, [h'([x] + c), a'_c]]$ where a'_i is the last letter in $([x] + i)$ and after reading the letter a transitions to the next state $[[h'([xa]), a_0], \dots, [h'([xa] + c), a_c]]$.

The automaton $M' = (Q', \Sigma_2, q'_0, \delta', \Delta, \tau')$ is constructed as follows. The states in Q' are of the form $[[p_0, a_0], \dots, [p_c, a_c]]$ where $p_i \in Q$ and $a_i \in \Sigma_2$. The initial state q'_0 is $[[h'(0), 0], \dots, [h'(c), a_c]]$ where a_c is the last letter of (c) . Furthermore, we have $\tau'([p_0, a_0], \dots, [p_c, a_c]) = \tau(p_c)$.

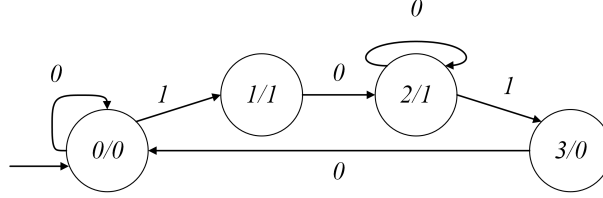


Figure 3.4: An automaton with input in Fibonacci representation generating the Thue-Morse sequence.

The transition function δ' is more complicated and requires explanation. If the input to δ' are $[[p_0, a_0], \dots, [p_c, a_c]]$ and a , it computes the output as follows. First, it creates a list. Then it goes through each $[p_i, a_i]$ in order; if $a_i = 0$, then $[\delta(p_i, 0), 0]$ and $[\delta(p_i, 1), 1]$ are added to the list; otherwise only $[\delta(p_i, 0), 0]$ is added to the list. Finally, if $a = 0$, the function outputs the 0-th to c -th element; otherwise it outputs the 1-st to $(c + 1)$ -th element.

Now let us find the number of states in M' . Since $p_0, \dots, p_c$ form a length- $(c + 1)$ subsequence of $(h'(i))_{i \geq 0}$, there are $O(m^2 c)$ possibilities. Furthermore, the $a_0, \dots, a_c$ form a length- $(c + 1)$ subsequence of the infinite Fibonacci word and there are $O(c)$ possibilities for them. So the automaton M' for $(h(i + c))_{i \geq 0}$ has $O(m^2 c^2)$ states. $\square$

Theorem 48. *Let $(t(i))_{i \geq 0}$ be the Fibonacci-Thue-Morse sequence; in other words, $t(i)$ is the number of 1s in the Fibonacci representation of i modulo 2. For a constant $c \geq 0$, the number of states in the minimal DFAO generating $(t(i + c))_{i \geq 0}$ is $O(c)$.*

Proof. Let us create the automaton M_c for $(t(i + c))_{i \geq 0}$ based on the construction provided in Theorem 47. On input x , the automaton M_c reaches a state of the form $[[t'([x]), a'_0], \dots, [t'([x] + c), a'_c]]$ where a'_j is the last letter of $([x] + j)$. Consider the automaton generating $(t(i))_{i \geq 0}$ from [41] shown in Figure 3.4. We can see that $t'([x] + j)$ determines a'_j . So from Lemma 35, we conclude that there are $O(c)$ states in M_c . $\square$

We suspect that the number of states in a minimal automaton generating $(t(i + c))_{i \geq 0}$ has a linear lower bound too. The following is left as an open problem.

Problem 3. *Prove the number of states in a minimal automaton generating $(t(i + c))_{i \geq 0}$ is $\Theta(c)$.*

In [33], we study the state complexity of generating $(f(i+c))_{i \geq 0}$ where $(f(i))_{i \geq 0}$ is the Fibonacci word $\mathbf{f}$ and we prove it is $O(\log c)$.

Theorem 49. *Let $n \geq 1$ and $0 \leq c < n$, and let $(h(i))_{i \geq 0}$ be a Fibonacci-automatic sequence generated by a DFAO of m states. There is a DFAO of $O(m^2 n^4)$ states recognizing the linear subsequence $(h(ni+c))_{i \geq 0}$.*

Let $(h(n))_{n \geq 0}$ be an arbitrary Fibonacci-automatic sequence computed by a DFAO

$$M_h = (Q_h, \Sigma_2, \Delta, \delta_h, q_{0,h}, \tau_h),$$

where Δ is a finite alphabet and τ_h is the output function associated with M_h . Fix n and c . We now form a UFAO $M_u = (Q_u, \Sigma_2, \Delta, \delta_u, q_{0,u}, A_u, \tau_u)$ as follows:

$$\begin{aligned} Q &= \{[a', b', d, d', q'] : a', b' \in \Sigma_2 \text{ and } d, d' \in I_n \text{ and } q' \in Q_h\}, \\ \delta([a', b', d, d', q'], a) &= \{[a, b, g, d, q] : b \in \Sigma_2, g = d + d' + b' - na' + b - na, d \in I_n, \\ &\quad aa', bb' \neq 11, \text{ and } q = \delta_h(q', b)\}, \\ q_{0,u} &= \{[0, b', 0, i, p] : 0 \leq i \leq 2n-1, b' = \mathbf{f}[i], p = \delta_h(q_{0,h}, (i))\}, \\ A_u &= \{[a', b', d, d', q'] : d = c\}, \\ \tau_u([a', b', d, d', q']) &= \tau_h(q'). \end{aligned}$$

Conceptually, M_u is formed by taking a Cartesian product of $M_{n,c}$ with M_h that takes an input x , “guesses” y , and simulates $M_{n,c}$ on $x \times y$ and M_h on y . The output of the UFAO is the unique output of M_h on y corresponding to $d = c$.

The initial state $q_{0,u}$ requires some explanation. It corresponds to all inputs x such that $[x] = 0$ (in other words, $x = 0^j$ for some j) and all possible y such that $[y] - n[x] \in I_n$. This is required because we want the DFAO we construct to return the correct result no matter how many leading zeros are prefixed to the input.

Next, we use the subset construction on M_u , obtaining a DFAO M . On input x this DFAO computes $h(n[x] + c)$. The states of M are subsets of states of M_u . We first prove a lemma that characterizes the subsets of states that can appear in a state of M .

Lemma 50. *For each state p in M there exist $a \in \Sigma_2$ and integers S, T such that $[S, S+T] \subseteq [-n, 2n-1]$ and every state of p is of the form $[a, b_i, d_i, d'_i, s_i]$ with $0 \leq i \leq T$ and $d_i = S + i$.*

In other words, the third components of the states of p are distinct, and consist of all integers in the range $[S, S+T]$ exactly once.

Proof. Consider a state p reachable via the input xa . The subset construction generates those states of M_u reachable via the input $xa \times yb$ where $r = |y| = |x|$. The corresponding integers $[yb]$ are therefore precisely those in the interval $[0, F_{r+2} - 1]$. The corresponding d_i are of the form $[yb] - n[xa]$. However, not all of these result in valid states because the d'_i, d_i could fall outside the interval $[-n, 2n - 1]$. Thus the range of permissible d_i is an intersection of intervals, and hence itself is a subinterval of $[-n, 2n - 1]$. $\square$

Proof of Theorem 49. Consider an input xa to M . By Lemma 50 we know that on this input M reaches a set of states corresponding to inputs $xa \times y_i b_i$ in M_u , where $d_i = [y_i b_i] - n[xa] = S + i$ for $0 \leq i \leq T$, some word y_i and letter b_i . Let $[y_0 b_0] = Y$. Then $[y_i b_i] = Y + i$ for $0 \leq i \leq T$. Noting that $d'_i = [y_i] - n[x]$, define $e_i = d'_{i+1} - d'_i$ for $0 \leq i < T$. Let g be the function that maps t to the integer obtained by dropping the last bit in the Fibonacci representation of t ; Lemma 37 shows that $g(t) = \lfloor (t+2)/\alpha \rfloor - 1$. Then $d'_i = g(Y + i) - n[x]$. The triples (b_i, e_i, s_i) are given by $(\mathbf{f}[Y + i], g(Y + i + 1) - g(Y + i), h'(Y + i))$ for $0 \leq i \leq T$, and this formula completely specifies *all* the b_i and d'_i and s_i , once d'_0 is chosen. Namely, $d'_i = d'_0 + \sum_{0 \leq j < i} e_j$.

On input x the automaton generating $(g(j+1) - g(j))_{j \geq 0}$ only needs to keep track of the last two letters read; if any of them is 1 then $g([x]) = 1$, otherwise $g([x]) = 0$. Now think of the triples (b_i, e_i, s_i) as a subsequence of the infinite sequence of triples $\mathbf{x} := (\mathbf{f}[j], g(j+1) - g(j), h'(j))_{j \geq 0}$, which is an automatic sequence because $(\mathbf{f}[j])_{j \geq 0}$, $(g(j+1) - g(j))_{j \geq 0}$, and the interior sequence $(h'(j))_{j \geq 0}$ are automatic sequences. Furthermore, the sequence $\mathbf{x}$ is generated by a DFAO of $O(m)$ states. The finite sequence $(\mathbf{f}[Y + i], g(Y + i + 1) - g(Y + i), h'(Y + i))_{0 \leq i \leq T}$ is then completely specified by saying which factor of length $T + 1$ of $\mathbf{x}$ it is, and by Lemma 35, there are $O(m^2 n)$ of them.

Summarizing, there are

- (a) $O(n)$ choices for $d_0 \in [-n, 2n - 1]$,
- (b) $O(n)$ choices for $d'_0 \in [-n, 2n - 1]$,
- (c) $O(n)$ choices for T ,
- (d) $O(m^2 n)$ choices for which particular subword of the automatic sequence $\mathbf{x}$ gives $(b_i, d'_{i+1} - d'_i, s_i)$ for $0 \leq i \leq T$,
- (e) $O(1)$ choices for a .

Therefore there are $O(m^2 n^4)$ possible states. $\square$

Example 51. Let us find the minimal DFAO for $\mathbf{f}[2i]$. We can use the free software Walnut for this, using the commands

```
def f2 "?msd_fib F[2*i]=@1":
combine F2 f2=1:
```

This gives us the DFAO in Figure 3.5.

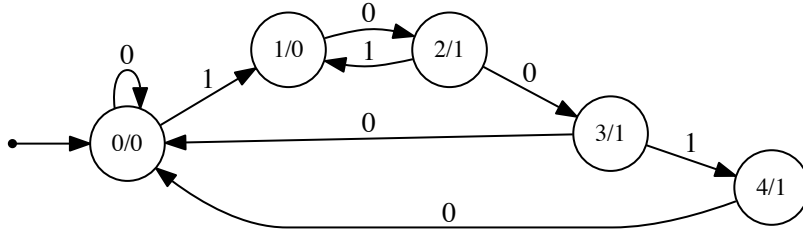


Figure 3.5: An automaton with input in Fibonacci representation generating $\mathbf{f}[2i]$.

Note that Walnut's output is a minimized automaton. So Figure 3.5 is the minimized version of the automaton built by the construction provided in this chapter.

Remark 52. We point out that the automaton we have constructed for $(h(ni + c))_{i \geq 0}$ is not minimal. However, we can find the minimal automaton in polynomial time by a minimization algorithm. The minimization algorithm running in $O(n \log n)$ time by Hopcroft [23] can be adapted for DFAOs as described by van Spaendonck [46].

For the special case where $c = 0$ and $h(i) = \mathbf{f}[i]$, the infinite Fibonacci word, the smallest DFAO for the linear subsequence $\mathbf{f}[ni]$ has a number of states given by OEIS [25] sequence A385021. The first 10 terms, corresponding to $1 \leq n \leq 10$, are 2, 5, 10, 17, 27, 36, 52, 65, 78, 103.

3.4.1 Application to a Problem of Bosma and Don

As is well-known, a k -automatic sequence can also be generated as the image (under a coding) of the fixed point $\gamma^\omega(0)$ of a prolongable morphism γ that maps each letter of

some alphabet Σ to a word of Σ^* of length k [3]. Here “prolongable” means that the image of the letter 0 begins with 0. In fact, the appropriate morphism can be deduced immediately from the automaton, as follows: the letters of the alphabet are the states q of the automaton; the value $\gamma(q)$ is given by the states reached on inputs $0, 1, \dots, k-1$. The coding is defined by the output associated with each state. Roughly the same thing is true for Fibonacci-automatic sequences, except now the values of $\gamma(q)$ are given by the outputs on 0 and 1 (if the latter is possible by the rules of the numeration system).

Example 53. *From the automaton in Figure 3.5 we can read off the following representation for the subsequence $(\mathbf{f}[2i])_{i \geq 0}$ in terms of a morphism γ and coding ρ :*

$$\begin{array}{ll} \gamma(0) = 01, & \tau(0) = 0, \\ \gamma(1) = 2, & \tau(1) = 0, \\ \gamma(2) = 31, & \tau(2) = 1, \\ \gamma(3) = 04, & \tau(3) = 1, \\ \gamma(4) = 0, & \tau(4) = 1. \end{array}$$

When we iterate the morphism γ we get the sequence $01231040 \dots$ and after applying the coding τ we get $00110000 \dots$.

Bosma and Don [9] studied the linear subsequences of the Fibonacci word $\mathbf{f}$. They were mostly concerned with the size of the smallest morphism generating the sequence $(\mathbf{f}[ni + c])_{i \geq 0}$, and they were able to prove an exponential upper bound on the size of this morphism. Here the size of the morphism is the total number of symbols required to write it down, that is, $\sum_{a \in \Sigma} |\gamma(a)|$.

As a consequence of our results, we can reduce the size of the needed morphism from exponential to polynomial.

Corollary 54. *Let $n \geq 1$ and $0 \leq c < n$, and let $(h(i))_{i \geq 0}$ be a Fibonacci-automatic sequence. There is a morphism of size $O(n^4)$ generating the subsequence $(h(ni + c))_{i \geq 0}$.*

Proof. As we saw in Theorem 49, the resulting automaton has $O(m^2 n^4)$ states. Hence the corresponding morphism has size $O(m^2 n^4)$. $\square$

3.5 Construction by Büchi Arithmetic

Similar to what had been previously discussed for the case of base- k in Chapter 2, given an automatic sequence, we can compute the automaton for a linear subsequence using the free

software `Walnut` [36, 42]. `Walnut` uses an interpretation of Büchi arithmetic and implements a decision procedure due to Büchi [13, 12] and takes a first-order logical statement and translates it into the corresponding automaton. We use the total number of transitions of an automaton M as a proxy for the computational complexity of constructing M .

As mentioned in Section 2.5, in the current version of `Walnut`, the algorithms for creating the automata we discussed in the previous sections of this chapter are not used. Instead, more general techniques, suitable for any addable numeration system, are used. This means that the size of intermediate automata may differ from the minimal automata we constructed, and the running time can similarly be larger.

Furthermore, the minimization steps in the implementation for an n -state automaton can use an algorithm running in $O(n \log n)$ time by Hopcroft [23] or Valmari [45]. The algorithm by Hopcroft can be slightly modified to be applied to DFAOs as described by van Spaendonck [46]. Note that in an n -state DFA, the number of transitions is $O(n)$. The minimization is applied to each DFA or DFAO created. In this chapter, we sometimes only apply partial minimizations and ignore the potential effects of fully minimizing automata in order to be able to analyze the runtime complexity and obtain an upper bound; the upper bound on state complexity and runtime complexity will remain correct regardless and the runtime complexity obtained reflects the computational cost of a complete minimization.

Consider the automaton M_{add} from Mousavi et al. [37] that accepts input $x \times y \times z$ if $[x] + [y] = [z]$. On input $x \times y \times z$ this automaton leads to the state denoted by the sequence $([x0^n] + [y0^n] - [z0^n])_{n \geq 0}$. We modify this automaton to keep track of some extra information. Consider the automaton M'_{add} created from M_{add} that on input $xa \times yb \times ze$ leads to the state denoted by $[s, s', e]$ where $s = ([xa0^n] + [yb0^n] - [ze0^n])_{n \geq 0}$ and $s' = ([x0^n] + [y0^n] - [z0^n])_{n \geq 0}$.

If we have $M_{\text{add}} = (Q, \Sigma_2^3, \delta, q_0, A)$, we create $M'_{\text{add}} = (Q', \Sigma_2^3, \delta', q'_0, A')$ as follows.

$$\begin{aligned} Q' &= Q \times Q \times \Sigma_2, \\ \delta'([s, s', e], [a, b, e]) &= [\delta(s, [a, b, e]), s, e], \\ q'_0 &= [q_0, q_0, 0], \\ A' &= \{[s, s', e] : s \in A\}. \end{aligned}$$

We use $M'_{\text{add}}(x, y, z)[i]$ to show the i -th component of the state M'_{add} reaches on input $x \times y \times z$ for $i \in \{0, 1, 2\}$; for example, $M'_{\text{add}}(x, y, z)[0]$ is $([x0^n] + [y0^n] - [z0^n])_{n \geq 0}$. The automaton M'_{add} has $O(1)$ states.

Lemma 55. *The minimal automaton for recognizing the relation $[x]_k = c$ has $O(\log c)$ states.*

Proof. Such an automaton consists of a linear chain of nodes labeled with the Fibonacci representation of c , which has length $O(\log c)$. $\square$

Theorem 56. *Let $c \geq 0$ be a fixed constant. The automaton recognizing the relation $[x] = c$ can be computed in $O((\log^2 c)(\log \log c))$ time.*

Proof. We recursively create the automaton $M_{=c}$.

The base case is $c = 0$ and the automaton has 1 state.

If c is even, we recursively compute a minimal DFA $M_{=(c/2)}$ recognizing $[y] = c/2$ on input y and then use the first-order expression

$$\exists y \ M_{=(c/2)}(y) \wedge M_{\text{add}}(y, y, x)$$

which is translated into an automaton. We know the automaton for M_{add} has $O(1)$ states and by Lemma 55, the automaton $M_{=(c/2)}$ has $O(\log c)$ states. First we apply the product construction to $M_{=(c/2)}$ and M_{add} such that the $M_{=(c/2)}$ part processes y and the M_{add} part processes $y \times y \times x$; the resulting automaton has $O(\log c)$ states. Each state in the product automaton is of the form $[q, p]$ where on input $x \times y$, q is a state from the minimal automaton $M_{=(c/2)}$ representing the number of letters from $(c/2)$ matched with y by Lemma 55 and p is a state from M_{add} such that $p(0) = 2[y] - [x]$. Then we apply the $\exists y$ quantifier by removing the component corresponding to y from the transitions and we get an NFA. In the NFA there are no two same transitions from a single state leading to different states. So after subset construction the resulting DFA is the same as the NFA. Then we do minimization and the resulting DFA is the automaton from Lemma 55 with $O(\log c)$ states.

If c is odd, we recursively compute a DFA $M_{=(c-1)}$ recognizing $[y] = c - 1$ on input y and then use the first-order expression

$$\exists y \ M_{=(c-1)}(y) \wedge M_{\text{add}}(y, 1, x)$$

which is translated into an automaton. Following a construction similar to the one described for the case where c is even, we get the DFA from Lemma 55 for constant c with $O(\log c)$ states.

Taking minimization steps into account, we get the following recursive formula for the time complexity:

$$T(c) = \begin{cases} O((\log c)(\log \log c)) + T(c/2), & \text{if } c \bmod 2 = 0; \\ O((\log c)(\log \log c)) + T(c - 1), & \text{if } c \bmod 2 = 1. \end{cases}$$

So the total time is $O((\log^2 c)(\log \log c))$. $\square$

Theorem 57. *Let $c \geq 0$ be a fixed constant. The automaton recognizing the relation $[x] + c = [z]$ can be computed in $O((\log^2 c)(\log \log c))$ time.*

Proof. Assume we have modified M'_{add} again so that on input $xa \times yb \times ze$, it keep track of a in addition to e and its states are of the form $[s, s', a, e]$. We can use the following expression to create the automaton M_c recognizing $[x] + c = [z]$ on input $x \times z$. Moreover, we have automaton $M_{=c}$ recognizing $[y] = c$ from Theorem 56.

$$\exists y M'_{\text{add}}(x, y, z) \wedge M_{=c}(y).$$

First, we apply the product construction to M'_{add} and $M_{=c}$ such that the M'_{add} part processes $x \times y \times z$ and the $M_{=c}$ part processes y . The resulting automaton M'_c has $O(\log c)$ states. Let $\text{Pre}(c, i)$ denote the first i letters in (c) . On input $xa \times yb \times ze$ where yb is $0^*y'b$ and $y'b$ is a prefix of (c) the automaton M'_c leads to a state $[q, p]$ where q is from M'_{add} and $p = |y'b|$ is from $M_{=c}$ indicating how many letters from y have been matched with (c) . Therefore, we have $q = [s, s', a, e]$ where $s(0) = [xa] + [yb] - [ze]$, $s'(0) = [x] + [y] - [z]$, and $[\text{Pre}(c, p)] = [yb]$. So $[ze] - [xa] = [\text{Pre}(c, p)] - s(0)$ and $[z] - [x] = [\text{Pre}(c, p - 1)] - s'(0)$. So to each state $[q, p]$ in M'_c a tuple $[d, d', a, e]$ can be attributed where $d = [ze] - [xa]$ and $d' = [z] - [x]$ and two states reachable by the same xa, ze cannot have different $[d, d', a, e]$ attributed to them.

Then we remove the y component from transitions and determinize the automaton to obtain a DFA. Consider a (non-dead) state reachable on input $xa \times ze$ in the DFA. This state consists of some $[q, p]$ such that $[ze] - [xa] = [\text{Pre}(c, p)] - s(0)$ and $[z] - [x] = [\text{Pre}(c, p - 1)] - s'(0)$. In the NFA there are no two same transitions from a single state leading to different states. So after subset construction the resulting DFA is the same as the NFA.

From the proof of Theorem 39, we know the automaton recognizing $[x] + c = [z]$ on input $x \times z$ only needs to keep track of a difference $[z] - [x]$ in range $[0, c]$ with $O(\log c)$ states. So after some minimization, the automaton $M_c = (Q, \Sigma_2, q_0, \delta, A)$ can be formally defined as follows. Let $I_c = [0, c]$.

$$\begin{aligned} Q &= \{[a, b, d, d'] : d, d' \in I_c\}, \\ q_0 &= [0, 0, 0, 0], \\ \delta([a', b', d, d'], [a, b]) &= [a, b, g, d] \text{ where } g = d + d' + b' - a' + b - a, \\ A &= \{[a, b, d, d'] : d = c\}. \end{aligned}$$

Note that not all states in the formal definition above are necessarily reachable.

Considering the time spent on creating $M_{=c}$ and minimization steps, it takes $O((\log^2 c)(\log \log c))$ time to create the automaton M_c recognizing $[x] + c = [z]$ on input $x \times z$. $\square$

We continue with the automaton recognizing the relation $[z] = n[x]$ where n is a fixed constant. There are a few possible recursions for implementing this automaton.

- $\exists y M_{n-1}(x, y) \wedge M_{\text{add}}(x, y, z)$.
- $\exists y_1, y_2 M_{\lfloor n/2 \rfloor}(x, y_1) \wedge M_{\lfloor n/2 \rfloor}(x, y_2) \wedge M_{\text{add}}(y_1, y_2, z)$.

However, a third recursive implementation that follows is more efficient than the two above.

If n is even, we recursively compute a DFA $M_{n/2}(x, y)$ recognizing $[y] = (n/2)[x]$ and then use the first-order expression

$$\exists y M_{n/2}(x, y) \wedge M'_{\text{add}}(y, y, z),$$

which is translated into an automaton by a direct product construction for the $\wedge$, and projection of the 2-nd coordinate to remove transitions on y .

If n is odd, we recursively compute a DFA $M_{n-1}(x, y)$ recognizing $[y]_k = (n-1)[x]_k$ and use the expression

$$\exists y M_{n-1}(x, y) \wedge M'_{\text{add}}(x, y, z).$$

In both cases, the translation could conceivably generate a nondeterministic automaton (by the projection) and determinizing it could take exponential time. However, we show that this is not the case.

Theorem 58. *Let $n \geq 1$ be a fixed constant. An automaton for recognizing $[y] = n[x]$ on input $x \times y$ can be computed in $O(n^6 \log^2 n)$ time.*

Proof. We first prove by induction on n that we can recursively create the automaton M_n with states of the form $[a, b, d, d']$ such that on input $xa \times yb$ the automaton reaches a state where $d = [yb] - n[xa]$ and $d' = [y] - [x]$, and $d, d' \in I_n = [-n, 2n - 1]$.

Base case: $n = 1$.

In this case, a simple 1-state automaton recognizes $[y] = [x]$. The single state corresponds to $[a, b, d, d'] = [0, 0, 0, 0]$.

For the induction step, assume the result is true for all $n' < n$; we prove it for n . There are two cases.

Case 1: $n \bmod 2 = 0$.

By the induction hypothesis, we have constructed a DFA $M_{n/2}$ with $O(n^2)$ states corresponding to all possible $[a, b, d, d']$ where $a, b \in \Sigma_2$ and $d, d' \in I_{n/2} = [-(n/2), 2(n/2) - 1]$. In this case, we use the formula

$$\exists y M_{n/2}(x, y) \wedge M'_{\text{add}}(y, y, z).$$

First we create an automaton M'_n that is the product construction of $M_{n/2}$ and M'_{add} . So this automaton has $O(n^2)$ states. Consider an input $xa \times yb \times ze$ for M'_n leading to some state $[q, p]$ where q is from $M_{n/2}$ and p is from M'_{add} . By the construction of M'_n , we know $q = [a, b, d, d']$ where $d = [yb] - (n/2)[xa]$, $d' = [y] - (n/2)[x]$ and $p = [s, s', e]$ where $s = (2[yb0^n] - [ze0^n])_{n \geq 0}$, $s' = (2[y0^n] - [z0^n])_{n \geq 0}$. So we have $2d - s(0) = [ze] - n[xa]$ and $2d' - s'(0) = [z] - n[x]$. Therefore, for each state reachable by $xa \times yb \times ze$ in M'_n a tuple $[a, e, \bar{d}, \bar{d}']$ can be attributed where $\bar{d} = [ze] - n[xa]$ and $\bar{d}' = [z] - n[x]$. Furthermore, two input with the same xa, ze cannot lead to states with different $[a, e, \bar{d}, \bar{d}']$ attributed to them.

Next, we project away the y component. The resulting automaton is an NFA, and we use the subset construction to create a DFA. Consider a state from the DFA reachable on $xa \times ze$. This state consists of some pairs $[q, p]$ from the NFA such that $q = [a, b, d, d']$, $p = [s, s', e]$ and we can create $[a, e, 2d - s(0), 2d' - s'(0)]$ from $[q, p]$ where $[ze] - n[xa] = 2d - s(0)$ and $[z] - n[x] = 2d' - s'(0)$.

In each state of the DFA, using an argument similar to the one used in the proof of Theorem 49, there are $O(n^4)$ possibilities for the $q = [a, b, d, d']$ parts. All $[q, p]$ in this state should have the same $2d - s(0)$ and $2d' - s'(0)$ ($O(n^2)$ possibilities). If we look at the transition table from Hamoon et al. [37] and consider all $\delta_{\text{add}}(s', [b, b, e]) = s$ where δ_{add} is the transition function of M_{add} , then for each b the $(s'(0), s(0))$ pairs obtained from transitions are unique. So considering we have already determined the b, d, d' of each $[q, p]$ and we have fixed the $2d - s(0)$ and $2d' - s'(0)$, the s, s' are now determined as well ($O(1)$ possibilities). So there are $O(n^6)$ states in the DFA.

From the proof of Theorem 41, we know the automaton recognizing $n[x] = [z]$ on input $x \times z$ only needs to keep track of a difference $[z] - n[x]$ in range $I_n = [-n, 2n - 1]$. So after some minimization, the DFA $M_n = (Q, \Sigma_k^2, \delta, q_0, A)$ can be formally defined as follows.

$$\begin{aligned} Q &= \{[a, e, d, d'] : a, e \in \Sigma_2, d, d' \in I_n\}, \\ q_0 &= [0, 0, 0, 0], \\ \delta([a', e', d, d'], [a, e]) &= [a, e, g, d] \text{ where } g = d + d' + e' - na' + e - na, \\ A &= \{[a, e, d, d'] : d = 0\}. \end{aligned}$$

Case 2: $n \bmod 2 = 1$.

By the induction hypothesis, we have a DFA M_{n-1} with $O(n^2)$ states corresponding to all possible $[a, b, d, d']$ where $a, b \in \Sigma_2$ and $d, d' \in I_{n-1} = [-(n-1), 2(n-1)-1]$. In this case, we use the formula

$$\exists y M_{n-1}(x, y) \wedge M'_{\text{add}}(x, y, z)$$

to create an automaton M'_n from the usual product construction for M_{n-1} and M'_{add} . So this automaton has $O(n^2)$ states. Consider an input $xa \times yb \times ze$ for M'_n leading to some state $[q, p]$ where q is from M_{n-1} and p is from M'_{add} . By the construction of M'_n , we know $q = [a, b, d, d']$ where $d = [yb] - (n-1)[xa]$, $d' = [y] - (n-1)[x]$ and $p = [s, s', e]$ where $s = ([xa0^n] + [yb0^n] - [ze0^n])_{n \geq 0}$, $s' = ([x0^n] + [y0^n] - [z0^n])_{n \geq 0}$. So we have $d - s(0) = [ze] - n[xa]$ and $d' - s'(0) = [z] - n[x]$. Therefore, for each state reachable by $xa \times yb \times ze$ in M'_n a tuple $[a, e, \bar{d}, \bar{d}']$ can be attributed where $\bar{d} = [ze] - n[xa]$ and $\bar{d}' = [z] - n[x]$. Furthermore, two input with the same xa, ze cannot lead to states with different $[a, e, \bar{d}, \bar{d}']$ attributed to them.

Next, we project away the y component. The resulting automaton is an NFA, and we use the subset construction to create a DFA. Consider a state from the DFA reachable on $xa \times ze$. This state consists of some pairs $[q, p]$ from the NFA such that $q = [a, b, d, d']$, $p = [s, s', e]$ and we can create $[a, e, d - s(0), d' - s'(0)]$ from $[q, p]$ where $[ze] - n[xa] = d - s(0)$ and $[z] - n[x] = d' - s'(0)$.

In each state of the DFA, using an argument similar to the one used in the proof of Theorem 49, there are $O(n^4)$ possibilities for the $q = [a, b, d, d']$ parts and there are $O(1)$ possibilities for e in p . All $[q, p]$ in this state should have the same $d - s(0)$ and $d' - s'(0)$ ($O(n^2)$ possibilities). If we look at the transition table from Hamoon et al. [37] and consider all $\delta_{\text{add}}(s', [a, b, e]) = s$ where δ_{add} is the transition function of M_{add} , then for each a, b, e the $(s'(0), s(0))$ pairs obtained from transitions are unique. So considering we have already determined the a, b, e, d, d' of each $[q, p]$ and we have fixed the $d - s(0)$ and $d' - s'(0)$, the s, s' are now determined as well ($O(1)$ possibilities). So there are $O(n^6)$ states in the DFA.

From the proof of Theorem 41, we know the automaton recognizing $n[x] = [z]$ on input $x \times z$ only needs to keep track of a difference $[z] - n[x]$ in the range $I_n = [-n, 2n-1]$. So after some minimization, the DFA $M_n = (Q, \Sigma_k^2, \delta, q_0, A)$ can be formally defined as follows:

$$\begin{aligned} Q &= \{[a, e, d, d'] : a, e \in \Sigma_2, d, d' \in I_n\}, \\ q_0 &= [0, 0, 0, 0], \\ \delta([a', e', d, d'], [a, e]) &= [a, e, g, d] \text{ where } g = d + d' + e' - na' + e - na, \\ A &= \{[a, e, d, d'] : d = 0\}. \end{aligned}$$

Next we analyze the time complexity of creating the above automaton. We use $T(n)$ to show the time required to recursively create M_n . We use the number of transitions traversed as a measure of the time required.

In Case 1, the number of transitions traversed to obtain M_n is: $O(1)$ transitions for M'_{add} , $O(n^2)$ transitions for the product construction of $M_{n/2}$ and M'_{add} , $O(n^2)$ transitions for projecting away the y component, $O(n^6)$ transitions for the determinized automaton, in total $O(n^6)$ transitions in addition to the number of transitions traversed for creating $M_{n/2}$ and the minimizations.

In Case 2, the number of transitions traversed to obtain M_n is: $O(n^2)$ transitions for M_{n-1} , $O(1)$ transitions for M'_{add} , $O(n^2)$ transitions for the product construction of M_{n-1} and M'_{add} , $O(n^2)$ for projecting away the y component, $O(n^6)$ for the determinized automaton, in total $O(n^6)$ transitions in addition to the number of transitions traversed for creating M_{n-1} and the minimizations.

Putting everything together, we get the following recursive formula:

$$\begin{cases} T(n) = O(n^6 \log n) + T(n/2), \\ T(n) = O(n^6 \log n) + T(n-1). \end{cases}$$

So we have $T(n) = O(n^6 \log^2 n)$. □

Theorem 59. *Let $n \geq 1$, $0 \leq c < n$ be fixed constants. The automaton recognizing $n[x] + c = [z]$ on input $x \times z$ can be computed in $O(n^6 \log^2 n)$ time.*

Proof. Let M_c be the automaton from Theorem 57 and let M_n be the automaton from Theorem 58. We say $M_c(y, z)$ is 1 if and only if $[y] + c = [z]$. So we can use the following expression to create $M_{n,c}$:

$$\exists y M_n(x, y) \wedge M_c(y, z).$$

We first apply product construction to M_n and M_c . The resulting automaton $M'_{n,c}$ has $O(n^2 \log c)$ states.

Consider an input $xa \times yb \times ze$ for $M'_{n,c}$ leading to some state $[q, p]$ where q is from M_n and p is from M_c . By the construction of $M'_{n,c}$, we know that $q = [a, b, d_q, d'_q]$ where $d_q = [yb] - n[xa]$, $d'_q = [y] - n[x]$ and $p = [b, e, d_p, d'_p]$ where $d_p = [ze] - [yb]$, $d'_p = [z] - [y]$. So we have $d_q + d_p = [ze] - n[xa]$ and $d'_q + d'_p = [z] - n[x]$. Therefore, to each state in the automaton $M'_{n,c}$ a tuple $[a, e, d, d']$ can be attributed where $d = [ze] - n[xa]$ and $d' = [z] - n[x]$ and no two states reachable by the same xa, ze can have different $[a, e, d, d']$ attributed to them.

Next, we project away the y component from $M'_{n,c}$. The resulting NFA needs to be determinized by subset construction to get a DFA. Consider a state from the DFA reachable on $xa \times ze$. This state consists of some $[q, p]$ from the NFA such that $q = [a, b, d_q, d'_q]$, $p = [b, e, d_p, d'_p]$ and $d_q + d_p = [ze] - n[xa]$, $d'_q + d'_p = [z] - n[x]$. In each state of the DFA, using an argument similar to the one used in the proof of Theorem 49, there are $O(n^4)$ possibilities for the $q = [a, b, d_q, d'_q]$ parts. Then we fix e, d, d' for this state of DFA with $O(n^2)$ possibilities. Since $d_q + d_p = d$ and $d'_q + d'_p = d'$, we can now determine d_p, d'_p as well. So the DFA has $O(n^6)$ states.

From the proof of Theorem 41, we know the automaton recognizing $n[x] + c = [z]$ on input $x \times z$ only needs to keep track of a difference $[z] - n[x]$ in range $I_n = [-n, 2n - 1]$. So after some minimization, the DFA $M_{n,c} = (Q, \Sigma_k^2, \delta, q_0, A)$ can be formally defined as follows.

$$\begin{aligned} Q &= \{[a, e, d, d'] : a, e \in \Sigma_2, d, d' \in I_n\}, \\ q_0 &= [0, 0, 0, 0], \\ \delta([a', e', d, d'], [a, e]) &= [a, e, g, d] \text{ where } g = d + d' + e' - na' + e - na, \\ A &= \{[a, e, d, d'] : d = c\}. \end{aligned}$$

Taking into account the minimization steps and the time spent on creating M_n , the time spent on creating $M_{n,c}$ is $O(n^6 \log^2 n)$. $\square$

Theorem 60. *Let $n \geq 1$, $0 \leq c < n$ be fixed constants. Given an m -state DFAO generating $(h(i))_{i \geq 0}$, we can compute the DFAO for $(h(ni + c))_{i \geq 0}$ in $O(n^6 \log^2 n + m^2 n^4 \log(m^2 n^4))$ time.*

Proof. We are given a DFAO $M_h = (Q_h, \Sigma_2, \delta_h, q_{0,h}, \Delta_h, \tau_h)$ for $(h(i))_{i \geq 0}$ with m states and we want to create an automaton for $(h(ni + c))_{i \geq 0}$. We show the output of M_h on y by $M_h(y)$ and we have $M_h(y) = h([y])$. Furthermore, we have $M_{n,c}$ from above such that $M_{n,c}(x, y)$ is 1 if and only if $n[x] + c = [y]$.

We first use the product construction for $M_{n,c}$ on input $x \times y$ and M_h on input y to get the automaton M' . Consider a state $[q, p]$ reachable on $xa \times yb$ in M' where q is from $M_{n,c}$ and p is from M_h . We have $q = [a, b, d, d']$ where $d = [yb] - n[xa]$, $d' = [y] - n[x]$ and $p = \delta(q_{0,h}, yb)$. This automaton has $O(mn^2)$ states.

Next, we project the y component away and use the subset construction to obtain a DFAO from the non-deterministic automaton. On input xa the automaton reaches a state consisting of some $[q, p]$ where $q = [a, b, d, d']$ such that there is yb where $d = [yb] - n[xa]$, $d' = [y] - n[x]$, $d, d' \in I_n$ and $p = \delta(q_{0,h}, yb) \in Q_h$. In each state of the DFAO, using

an argument similar to the one used in the proof of Theorem 49, there are $O(m^2n^4)$ possibilities. Therefore, the DFAO has $O(m^2n^4)$ states.

Taking the time required for constructing $M_{n,c}$ and minimization steps into account, creating M given $M_{n,c}$ takes $O(n^6 \log^2 n + m^2n^4 \log(m^2n^4))$ time. $\square$

3.6 Conclusion

In the previous chapter, we studied topics similar to this chapter, but with base- k input instead of input in Fibonacci representation. We can attempt to study these topics in other numeration systems and solve similar problems. In Section 4.2, we introduce another open problem related to the current chapter.

Chapter 4

Conclusion

This chapter includes content taken nearly verbatim or verbatim from the arXiv paper [35] co-authored with Narad Rampersad and Jeffrey Shallit and the arXiv paper [34] co-authored with Narad Rampersad and Jeffrey Shallit.

4.1 Conclusion

In this thesis, we studied the state complexity of specific languages and the state complexity of specific automatic sequences. Furthermore, we studied the runtime complexity of creating automata for specific languages or automatic sequences using an interpretation of Büchi arithmetic. In Chapter 2 we focused on automata with input in base k for some $k \geq 2$ and in Chapter 3 we focused on automata with input in Fibonacci representation. In Section 4.2 we introduce more open problems.

We also want to note that an anonymous reviewer of the Developments in Language Theory (DLT) 2026 international conference has mentioned an alternative approach for studying the state complexity of recognizing relations that include multiplication and addition by a fixed integer; this alternative approach is based on the works of Frougny [19] and Grabner and Steiner [21].

4.2 More Open Problems

In Chapter 2 we introduced Problem 1 and 2. We first introduce more open problems related to Chapter 2.

Let $(h(i))_{i \geq 0}$ be a k -automatic sequence generated by a DFAO M of m states. Recall that $M(x)$ is the output of M on input word x . Let M' be an automaton such that on input $x \times y \times z$ the output is 1 if $h([x]) \cdots h([x] + [z] - 1) = h([y]) \cdots h([y] + [z] - 1)$, and 0 otherwise. In other words, the output is 1 if and only if the length- $[z]$ subwords of the sequence $(h(i))_{i \geq 0}$ starting at positions $[x]$ and $[y]$ are the same. We can use either of the following first-order logical expressions to compute M' by an interpretation of Büchi arithmetic similar to what we did in Section 2.5.

$$\begin{aligned} \forall u, [u] < [z] &\implies M([x] + [u])_k = M([y] + [u])_k, \\ \forall u, v, ([u] \geq [x] \wedge [u] < [x] + [z] \wedge [u] + [y] = [v] + [x]) &\implies M(u) = M(v). \end{aligned}$$

The state complexity and the time complexity of algorithmically creating M' can be studied similar to the other operations in Chapter 2. The automaton for $[u] \geq [x] \wedge [u] < [x] + [z] \wedge [u] + [y] = [v] + [x]$ has 9 states. So by a simple analysis, for the first one, we can prove an upper bound of $2^{O(m^4)}$ states and for the second 2^{9m^2} . However, both of these bounds are likely to be quite weak. We state the following questions.

Problem 4. *What is the best lower and upper bound on the state complexity of M' in terms of m ?*

Problem 5. *How much time is required to compute M' by an interpretation of Büchi arithmetic?*

In Chapter 3 we introduced Problem 3. Here we introduce another open problem related to that chapter.

Problem 6. *What is the state complexity of $(r(i + c))_{i \geq 0}$ where r is the Tribonacci word?*

References

- [1] Boris Alexeev. Minimal DFA for testing divisibility. *Journal of Computer and System Sciences*, 69(2):235–243, 2004.
- [2] Jean-Paul Allouche and Jeffrey Shallit. The ubiquitous Prouhet-Thue-Morse sequence. In C. Ding, T. Hellese, and H. Niederreiter, editors, *Sequences and Their Applications*, Discrete Mathematics and Theoretical Computer Science, pages 1–16. Springer, 1999.
- [3] Jean-Paul Allouche and Jeffrey Shallit. *Automatic Sequences: Theory, Applications, Generalizations*. Cambridge University Press, 2003.
- [4] Nicolas Amat, Pierre Ganty, and Alessio Mansutti. How big is the automaton? Certified lower bounds on the size of Presburger DFAs. In *2025 40th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 2670–2681. IEEE, 2025. Patapsco available at <https://patapsco.software.imdea.org/>, Accessed 26 March 2026.
- [5] S. V. Avgustinovich. The number of distinct subwords of fixed length in the Morse-Hedlund sequence. *Sibirskii Zhurnal Issledovaniya Operatsii*, 1(2):3–7, 1994. In Russian. English translation in Alekseï D. Korshunov, editor, *Discrete Analysis and Operations Research*, volume 355 of *Mathematics and Its Applications*, pages 1–5. Kluwer Academic Publishers, 1996.
- [6] S. V. Avgustinovich, D. G. Fon-Der-Flaass, and A. E. Frid. Arithmetical complexity of infinite words. In Masami Ito and Teruo Imaoka, editors, *Proceedings of the International Conference Words, Languages & Combinatorics III*, pages 51–62. World Scientific, 2003.
- [7] Jean Berstel. Fibonacci words — a survey. In G. Rozenberg and A. Salomaa, editors, *The Book of L*, pages 13–27. Springer, 1986.

- [8] F. Blanchet-Sadri, James D. Currie, Narad Rampersad, and Nathan Fox. Abelian complexity of fixed point of morphism $0 \mapsto 012$, $1 \mapsto 02$, $2 \mapsto 1$. *The Electronic Journal of Combinatorics*, 14:#A11, 2014.
- [9] Wieb Bosma and Henk Don. Constructing morphisms for arithmetic subsequences of Fibonacci. In Venanzio Capretta, Robbert Krebbers, and Freek Wiedijk, editors, *Logics and Type Systems in Theory and Practice*, volume 14560 of *Lecture Notes in Computer Science*, pages 100–110. Springer, 2024.
- [10] Alexandre Boudet and Hubert Comon. Diophantine equations, Presburger arithmetic and finite automata. In Hélène Kirchner, editor, *Trees in Algebra and Programming—CAAP ’96*, volume 1059 of *Lecture Notes in Computer Science*, pages 30–43. Springer, 1996.
- [11] Srećko Brlek. Enumeration of factors in the Thue-Morse word. *Discrete Applied Mathematics*, 24(1-3):83–96, 1989.
- [12] Véronique Bruyère, Georges Hansel, Christian Michaux, and Roger Villemaire. Logic and p -recognizable sets of integers. *Bulletin of the Belgian Mathematical Society - Simon Stevin*, 1(2):191–238, 1994.
- [13] J. Richard Büchi. Weak second-order arithmetic and finite automata. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 6:66–92, 1960. Reprinted in Saunders Mac Lane and Dirk Siefkes, editors, *The Collected Works of J. Richard Büchi*, pages 398–424, Springer-Verlag, 1990.
- [14] Émilie Charlier, Célia Cisternino, and Adeline Massuir. State complexity of the multiples of the Thue-Morse set. In Jérôme Leroux and Jean-Francois Raskin, editors, *Proceedings Tenth International Symposium on Games, Automata, Logics, and Formal Verification*, volume 305 of *Electronic Proceedings in Theoretical Computer Science*, pages 34–49. Open Publishing Association, 2019.
- [15] Émilie Charlier, Célia Cisternino, and Adeline Massuir. Minimal automaton for multiplying and translating the Thue-Morse set. *The Electronic Journal of Combinatorics*, 28(3):#P3.12, 2021.
- [16] Émilie Charlier, Narad Rampersad, Michel Rigo, and Laurent Waxweiler. The minimal automaton recognizing $m\mathbb{N}$ in a linear numeration system. *INTEGERS: The Electronic Journal of Combinatorial Number Theory*, 11B:#A4, 2011.

- [17] Alan Cobham. On the base-dependence of sets of numbers recognizable by finite automata. *Mathematical Systems Theory*, 3(2):186–192, 1969.
- [18] Alan Cobham. Uniform tag sequences. *Mathematical Systems Theory*, 6(1-2):164–192, 1972.
- [19] Christiane Frougny. Numeration Systems. In M. Lothaire, editor, *Algebraic Combinatorics on Words*, volume 90 of *Encyclopedia of Mathematics and its Applications*, pages 230–268. Cambridge University Press, 2002.
- [20] Yuan Gao, Nelma Moreira, Rogério Reis, and Sheng Yu. A survey on operational state complexity. *Journal of Automata, Languages and Combinatorics*, 21(4):251–310, 2016.
- [21] Peter J. Grabner and Wolfgang Steiner. Redundancy of minimal weight expansions in Pisot bases. *Theoretical Computer Science*, 412(45):6303–6315, 2011.
- [22] Bernard R. Hodgson. Décidabilité par automate fini. *Annales des sciences mathématiques du Québec*, 7(1):39–57, 1983.
- [23] John Hopcroft. An $n \log n$ algorithm for minimizing states in a finite automaton. In Zvi Kohavi and Azaria Paz, editors, *Theory of Machines and Computations*, pages 189–196. Academic Press, 1971.
- [24] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley Series in Computer Science. Addison-Wesley Publishing Company, 1979.
- [25] OEIS Foundation Inc. The On-Line Encyclopedia of Integer Sequences. <https://oeis.org/>. Accessed 27 March 2026.
- [26] Jozef Jirásek, Jr., Galina Jirásková, and Juraj Šebej. Operations on unambiguous finite automata. *International Journal of Foundations of Computer Science*, 29(05):861–876, 2018.
- [27] Mazen Khodier, Luke Schaeffer, and Jeffrey Shallit. Self-verifying predicates in Büchi arithmetic. In Giuseppa Castiglione and Sabrina Mantaci, editors, *Implementation and Application of Automata*, volume 15981 of *Lecture Notes in Computer Science*, pages 237–251. Springer, 2026.
- [28] Donald E. Knuth. *Seminumerical Algorithms*, volume 2 of *The Art of Computer Programming*. Addison-Wesley, 3rd edition, 1997.

- [29] Jakub Konieczny and Clemens Müllner. Arithmetical subword complexity of automatic sequences, 2024. arXiv:2309.03180v2 [math.NT].
- [30] C. G. Lekkerkerker. Voorstelling van natuurlijke getallen door een som van getallen van Fibonacci. *Simon Stevin*, 29:190–195, 1951/52.
- [31] Aldo de Luca and Stefano Varricchio. Some combinatorial properties of the Thue–Morse sequence and a problem in semigroups. *Theoretical Computer Science*, 63(3):333–348, 1989.
- [32] John C. Martin. *Introduction to Languages and the Theory of Computation*. McGraw-Hill, 4th edition, 2011.
- [33] Delaram Moradi, Pierre Popoli, Jeffrey Shallit, and Ingrid Vukusic. State complexity of shifts of the Fibonacci word. arXiv:2603.18858 [cs.FL], 2026.
- [34] Delaram Moradi, Narad Rampersad, and Jeffrey Shallit. Complexity of linear subsequences of Fibonacci-automatic sequences, 2026. arXiv:2603.21645v1 [cs.FL].
- [35] Delaram Moradi, Narad Rampersad, and Jeffrey Shallit. Complexity of linear subsequences of k -automatic sequences, 2026. arXiv:2512.10017v5 [cs.FL].
- [36] Hamoon Mousavi. Automatic theorem proving in Walnut, 2021. arXiv:1603.06017v2 [cs.FL], Walnut available at <https://walnut-theorem-prover.github.io/>, Accessed 26 March 2026.
- [37] Hamoon Mousavi, Luke Schaeffer, and Jeffrey Shallit. Decision algorithms for Fibonacci-automatic words, I: Basic results. *RAIRO Theoretical Informatics and Applications*, 50(1):39–66, 2016.
- [38] Alexander Ostrowski. Bemerkungen zur Theorie der diophantischen Approximationen. *Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg*, 1(1):77–98, 1922. Reprinted in volume 3 of Collected Mathematical Papers, pages 57–80. Birkhäuser Verlag, 1984.
- [39] Don Reble. Zeckendorf vs. Wythoff representations: Comments on A007895, 2008. Available at <https://oeis.org/A007895>.
- [40] Jeffrey Shallit. *A Second Course in Formal Languages and Automata Theory*. Cambridge University Press, 2009.

- [41] Jeffrey Shallit. Subword complexity of the Fibonacci–Thue–Morse sequence: The proof of Dekking’s conjecture. *Indagationes Mathematicae*, 32(3):729–735, 2021.
- [42] Jeffrey Shallit. *The Logical Approach to Automatic Sequences: Exploring Combinatorics on Words with Walnut*, volume 482 of *London Mathematical Society Lecture Note Series*. Cambridge University Press, 2023.
- [43] Michael Sipser. *Introduction to the Theory of Computation*. Cengage Learning, 3 edition, 2013.
- [44] Klaus Sutner. Divisibility and state complexity. *The Mathematica Journal*, 11(3):430–445, 2010.
- [45] Antti Valmari. Fast brief practical DFA minimization. *Information Processing Letters*, 112(6):213–217, 2012.
- [46] Flip van Spaendonck. Automatic Sequences: The Effect of Local Changes on Complexity. Master’s thesis, Radboud University, Netherlands, 2020.
- [47] Laurent Waxweiler. *Caractère reconnaissable d’ensembles de polynômes à coefficients dans un corps fini*. Docteur en sciences dissertation, Université de Liège, Belgium, 2009.
- [48] Pierre Wolper and Bernard Boigelot. On the Construction of Automata from Linear Arithmetic Constraints. In Susanne Graf and Michael Schwartzbach, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 1785 of *Lecture Notes in Computer Science*, pages 1–19. Springer, 2000.
- [49] Sheng Yu, Qingyu Zhuang, and Kai Salomaa. The state complexities of some basic operations on regular languages. *Theoretical Computer Science*, 125(2):315–328, 1994.
- [50] Hans Zantema. Complexity of automatic sequences. In Alberto Leporati, Carlos Martı́n-Vide, Dana Shapira, and Claudio Zandron, editors, *Language and Automata Theory and Applications*, volume 12038 of *Lecture Notes in Computer Science*, pages 260–271. Springer, 2020.
- [51] Hans Zantema and Wieb Bosma. Complexity of automatic sequences. *Information and Computation*, 288:104710, 2022.
- [52] E. Zeckendorf. Représentation des nombres naturels par une somme de nombres de Fibonacci ou de nombres de Lucas. *Bulletin de la Société Royale des Sciences de Liège*, 41(3-4):179–182, 1972.