Theoretical and Practical Aspects Related to the Avoidability of Patterns in Words

Kamellia Reshadi

Dissertation
zur Erlangung des akademischen Grades
Doktor der Ingenieurwissenschaften
(Dr.-Ing.)
der Technischen Fakultät
der Christian-Albrechts-Universität zu Kiel
eingereicht im Jahr 2019

Kiel Computer Science Series (KCSS) 2019/3 dated 2019-06-21

URN:NBN urn:nbn:de:gbv:8-diss-257685
ISSN 2193-6781 (print version)
ISSN 2194-6639 (electronic version)

Electronic version, updates, errata available via https://www.informatik.uni-kiel.de/kcss

The author can be contacted via kamelliareshadi@gmail.com

Published by the Department of Computer Science, Kiel University

Dependable Systems Group

Please cite as:

▻ Kamellia Reshadi. *Theoretical and Practical Aspects Related to the Avoidability of Patterns in Words* Number 2019/3 in Kiel Computer Science Series. Department of Computer Science, 2019. Dissertation, Faculty of Engineering, Kiel University.

```
@book{reshadi_theoretical_19,
  author    = {Kamellia Reshadi},
  title     = {Theoretical and Practical Aspects Related to the  Avoidability of
  Patterns in Words},
  publisher = {Department of Computer Science, Kiel University},
  year      = {2019},
  number    = {2019/3},
  series    = {Kiel Computer Science Series},
  note      = {Dissertation, Faculty of Engineering,
               Kiel University.}
}
```

About this Series

The Kiel Computer Science Series (KCSS) covers dissertations, habilitation theses, lecture notes, textbooks, surveys, collections, handbooks, etc. written at the Department of Computer Science at Kiel University. It was initiated in 2011 to support authors in the dissemination of their work in electronic and printed form, without restricting their rights to their work. The series provides a unified appearance and aims at high-quality typography. The KCSS is an open access series; all series titles are electronically available free of charge at the department's website. In addition, authors are encouraged to make printed copies available at a reasonable price, typically with a print-on-demand service.

Please visit `http://www.informatik.uni-kiel.de/kcss` for more information, for instructions how to publish in the KCSS, and for access to all existing publications.

1. Gutachter:	Prof. Dr. Dirk Nowotka Christian-Albrechts-Universität Kiel
2. Gutachter:	Prof. Dr. Tero Harju University of Turku Turku

Datum der mündlichen Prüfung: 21.June 2019

Zusammenfassung

Diese Arbeit beschäftigt sich mit wiederholenden Strukturen in Wörtern. Genauer gesagt, wird das vorkommen und das Fehlen solcher Wiederholungen in Wörtern untersucht.

Im ersten und größten Teil dieser Arbeit wird die Vermeidbarkeit unärer Muster mit Permutationen untersucht. Am Anfang zeigen wir, dass für gegebene $i, j, k \geqslant 0$, das Muster $x\pi^i(x)\pi^j(x)\pi^k(x)$ über einem Alphabet der Größe 2, 3 und 4, vermeidbar ist. Im nächsten Schritt beweisen wir, dass es ein Muster gibt, welches vermeidbar ist in Σ_m, für $m \geqslant 6$. Später versuchen wir stärkere Resultate zu finden indem wir ein Intervall σ finden, welches für ist $i, j, k \geqslant 0$, alle Muster $x\pi^i(x)\pi^j(x)\pi^k(x)$ vermeidbar über Alphabete mit $m > \sigma$ Buchstaben und vermeidbar für Alphabete mit höchstens $\sigma - 1$ Symbolen sind.

Der zweite Teil der Arbeit beschäftigt sich mit dem Modellieren und Lösen einiger Vermeidbarkeitsprobleme als sog. Constraint Satisfaction Probleme mithilfe des Werkzeugs MiniZinc. Für das Lösen von den vorher erwähnten Vermeidungsproblemen, ist es erforderlich, ein sehr langes Wort, welches kein gegebenes Muster enthält, mit einen Computerprogramm zu berechnen. Dadurch kamen wir auf die Idee, SAT-Solver einzusetzen.Die Darstellung der problembasierten SAT-Solver schien ein optimierter und effektiver Ansatz zu sein, um die bekannten Vermeidbarkeitsprobleme zu lösen.

Der letzte Teil beschäftigt sich mit Varianten klassischer Vermeidbarkeitsprobleme aus der Wortkombinatorik. In Anbetracht der Konkatenation von i Verschiedene Faktoren von dem Wort w, $\mathtt{pexp}_i(w)$ ist das Supremum von der Anzahl der Vorkommen jener i Faktoren, so dass ein Faktor in w erzeugt wird, und $\mathrm{RT}_i(k)$ ist dann das infimum von $\mathtt{pexp}_i(w)$ über allen Wörtern $w \in \Sigma_k^\omega$. Indem wir unendliche ternäre Wörter prüfen, die einige Eigenschaften erfüllen, zeigen wir, dass $\mathrm{RT}_i(3) = \frac{3i}{2} + \frac{1}{4}$ wenn i gerade ist und $\mathrm{RT}_i(3) = \frac{3i}{2} + \frac{1}{6}$ wenn i ungerade und $i \geqslant 3$ ist.

Die Zusammenfassung wurde nicht vom Kandidaten ins Deutsche übersetzt.

Abstract

This thesis concerns repetitive structures in words. More precisely, it contributes to studying appearance and absence of such repetitions in words.

In the first and major part of this thesis, we study avoidability of unary patterns with permutations. More precisely, we want to see whether there exists an infinite word such that these structures do not appear on it. To begin with, we show that given $i, j, k \geqslant 0$, the pattern $x\pi^i(x)\pi^j(x)\pi^k(x)$ is avoidable over an alphabet of size 2, 3 and 4, then we prove that there exists a pattern which is unavoidable in Σ_m, for $m \geqslant 6$. Later, we try to find stronger results by finding an interval σ such that given $i, j, k \geqslant 0$, all such patterns are unavoidable over alphabets with $m > \sigma$ letters, and avoidable for alphabets with at most $\sigma - 1$ symbols.

The second part of this thesis deals with modeling and solving several avoidability problems as constraint satisfaction problems, using the framework of MiniZinc. Solving avoidability problems like the one mentioned in the past paragraph required, the construction, via a computer program, of a very long word that does not contain any word that matches a given pattern. This gave us the idea of using SAT solvers. Representing the problem-based SAT solvers seemed to be a standardised, and usually very optimised approach to formulate and solve the well-known avoidability problems like avoidability of formulas with reversal and avoidability of patterns in the abelian sense too.

The final part is concerned with a variation on a classical avoidance problem from combinatorics on words. Considering the concatenation of i different factors of the word w, $\mathtt{pexp}_i(w)$ is the supremum of powers that can be constructed by concatenation of such factors, and $\mathrm{RT}_i(k)$ is then the infimum of $\mathtt{pexp}_i(w)$ over all words $w \in \Sigma_k^\omega$. Again, by checking infinite ternary words that satisfies some properties, we show that $\mathrm{RT}_i(3) = \frac{3i}{2} + \frac{1}{4}$ if i is even and $\mathrm{RT}_i(3) = \frac{3i}{2} + \frac{1}{6}$ if i is odd and $i \geqslant 3$.

Acknowledgements

First of all, I would like to thank my family, for their endless love and supports from long distance. Especially, I would like to thank my mother, for her neverending kindness, supports during all situations, especially the hard ones, and encouragements to be a strong woman like her. I owe her everything.

Next, I want to thank my supervisor Dirk Nowotka for his support, and for providing me with the necessary funding to pursuing my research interests. I would like to thank Florin Manea too, for all the helpful discussions and advice during my PhD study.

Furthermore, I would like to thank the members of my examining committee, Tero Harju, Klaus Jansen and Henning Schnoor for their efforts.

Thanks to my colleagues from the Dependable Systems Group in Kiel, who supported me, that is Philipp Sieweck, Thorsten Ehlers, Mitja Kulczynski, Danny Poulsen, Florin Manea, Joel Day, Pamela Fleischmann, Yannik Potdevin, Frederik Hamester, Karoliina Lehtinen, and Max Friese.

And finally, I would like to thank all my fantastic friends that I met in Kiel which made my life during this period more delightful.

Contents

List of Tables

Chapter 1

Introduction

Combinatorics on words is one area of algorithmic and discrete mathematics that has attracted significant attention from scholars in more recent times. This established area of research involves examining the arithmetic, geometrical, and combinatorial elements of infinite or finite discrete sequences that are referred to as words. These words consist of symbols, which are known as letters, that have been extracted from a finite set, the alphabet. This type of research has long-established roots and has been performed by renowned scholars such as Gauß[Gau00], Bernoulli [Ber71]. Berstel and Perrin [BP07] published a comprehensive survey that provides an in-depth overview of the historical development of word-related studies. Studies of this nature can be traced back over one hundred years to the formative work that was performed by Axel Thue, a Norwegian mathematician. Thue examined the use of two symbols that did not contain three consecutive identical blocks to construct an infinite sequence. This concept is known as avoiding cubes, i.e., three consecutive blocks. He also demonstrated how to derive a sequence using three symbols where every two consecutive blocks of the same length are different. Thue's work represented an early exploration of the area of the avoidability. Since Thue's early explorations, a range of extended forms of availability have emerged. One such extension is the study of Abelian patterns [Cur05; Ros17]. According to this model, the way in which a variable occurs in a given pattern should not be mapped to the same word but to an Abelian item, that is, a jumbled version of words. A pattern now matches a word if there is a way to map the variables of the pattern in the Abelian way such that the word is obtained. Another concept that is of relevance within the current study is unary patterns in which there are functional dependencies between variables. That is, there are patterns $x\pi^i(x)\pi^j(x)\ldots$ in which x is

a word variable as per the original setting, while π is a function variable that can be replaced by functions acting on words. It is possible to obtain the instances of these patterns over an alphabet Σ by exchanging x for a concrete word and substituting π by a function mapping x to words over Σ. The existing literature [GMM+13; GMN13; GMN14] outlines algorithms that can be employed to ascertain whether a word contains a factor that matches a given unary pattern that has functional dependencies. Specifically, numerous scholars ([MMN12a; Res19; BO15; RR16]) have demonstrated how a significant number of technical lemmas can be obtained using computer software. In the majority of studies, the software that was employed used backtracking or performed intense explorations of long words within an alphabet that could be expressed in the form of morphic images of some recognized infinite words in which there were no factors that matched the pattern.

Thesis outline

This thesis contributes to the study of avoiding repetitive special structures in words and provides solutions to some of the problems that have remained.

In Chapter 2, we study the avoidability of patterns of size four with functional dependencies between the blocks, i.e., a pattern similar to $x\pi^i(x)$ $\pi^j(x)\pi^k(x)$. This work improves on a result from [CMN15]. Here, we show that all patterns $\pi^{i_1}(x)\dots\pi^{i_n}(x)$ with $n \geqslant 4$ under morphic or antimorphic (anti-/morphic, for short) permutations are avoidable in alphabets of size 2, 3, and 4, but at least one pattern of this form becomes unavoidable in Σ_6 when π is replaced by a morphic permutation.

In Chapter 3, we continue the work of Florin Manea, Mike Müller, and Dirk Nowotka [MMN12a]. They characterize the (un)avoidability of cubic patterns or patterns of size three with permutations, which is denoted by $x\pi^i(x)\pi^j(x)$ (where x is a word variable and π is a functional variable with values that are in the set of all morphic permutations of the respective alphabets). We attempt to generalize this by achieving similar results for patterns of size four. To that end, we show that for the positive integers i, j, and k, the sizes of the alphabets over which a pattern $x\pi^i(x)\pi^j(x)\pi^k(x)$

is avoidable are intervals of integers. We also show how to compute a good approximation of this interval. For this work, we used computer programs in order to construct infinite k-uniform words (obtained using backtracking) in many lemmas.

In Chapter 4, we show how several relevant avoidability problems can be modeled and consequently solved in a uniform way as constraint satisfaction problems using the framework of MiniZinc. Solving avoidability problems in the area of combinatorics on words often requires, in an initial step, the construction of a very long word that does not contain any word that matches a given pattern using a computer program. It is well known that this is a computationally difficult task. We faced this issue while generating infinite avoidable words that would be defined in Chapter 4. At times, our computer programs needed to be run on the server for months to generate such words. Ultimately, despite it being rather straightforward for all such tasks to be formalized as constraint satisfaction problems, no unified approach to solve them has been proposed to date, and very diverse ad hoc methods have been used. The main advantage of the present approach is that one is now required only to formulate the avoidability problem in the MiniZinc language. Then, the actual search for a solution does not need to be implemented ad hoc and is instead conducted using a standard CSP-solver.

In Chapter 5, we continue our research path in the area of avoidability by solving another open problem related to this field. We consider an avoidance problem that was introduced by Mousavi and Shallit. Let $\mathsf{pexp}_i(w)$ be the supremum of the exponent over the products (concatenation) of i factors of the word w. The repetition threshold $\mathrm{RT}_i(k)$ is then the infimum of $\mathsf{pexp}_i(w)$ over all words $w \in \Sigma_k^\omega$. Mousavi and Shallit obtained that $\mathrm{RT}_i(2) = 2i$ and $\mathrm{RT}_2(3) = \frac{13}{4}$. We show that $\mathrm{RT}_i(3) = \frac{3i}{2} + \frac{1}{4}$ if i is even and $\mathrm{RT}_i(3) = \frac{3i}{2} + \frac{1}{6}$ if i is odd and $i \geqslant 3$.

Chapter 2

Preliminaries

In this section, we give the definitions and terminology that we will use in the rest of the thesis. For detailed definitions regarding combinatorics on words we refer to [Lot97], [Lot02].

Define alphabets $\Sigma_k = \{0, \ldots, k-1\}$ and $\Sigma'_k = \{1, 2, \ldots, k\}$. We use w^R, to denote the reversal of word w. The empty word is denoted by ε, $w[i]$ denotes the i^{th} symbol of w, $|w|$ denotes the length of a word $w \in \Sigma_k^*$, and $|w|_a$ denotes the number of occurrences of the letter $a \in \Sigma_k$ in w. For words u and w, we say that u is a prefix (resp. suffix) of w, if there exists a word v such that $w = uv$ (resp. $w = vu$). If $f : \Sigma_k \to \Sigma_k$ is a permutation, we say that the order of f, denoted $\mathbf{ord}(f)$, is the minimum value $m > 0$ such that f^m is the identity. If $a \in \Sigma_k$ is a letter, the order of a with respect to f, denoted $\mathbf{ord}_f(a)$, is the minimum number m such that $f^m(a) = a$.

2.1 Patterns and avoidability

A pattern with functional dependencies is a term over (word) variables and function variables (where concatenation is an implicit functional constant). For example, $x\pi(y)\pi(\pi(x))y$ is a pattern involving the variables x and y and the function variable π. An instance of a pattern p in Σ_k is the result of substituting uniformly every variable by a word in Σ_k^+ and every function variable by a function over Σ_k^*. A *substitution* (*for a word* δ) is a mapping $h : X \to \Sigma^+$. For every variable x occurring in δ, we say that x *is substituted by* $h(x)$. The word obtained by substituting every occurrence of a variable x in δ by $h(x)$ is denoted by $h(\delta)$. For instance, we consider the pattern $\beta = xyy$ and the words $u = \texttt{bcaca}$, $v = \texttt{aaaaa}$ and have $h(\beta) = u$, for $h(x) = \texttt{b}$ and $h(y) = \texttt{ca}$, and $g(\beta) = v$, for $g(x) = \texttt{a}$ and $g(y) = \texttt{aa}$. If there exists a substitution h such that $h(\delta) = w$, we say

that δ matches w. A pattern is avoidable in Σ_k if there is an infinite word over Σ_k that does not contain any instance of the pattern. The *avoidability problem* for a pattern asks whether, for a given pattern δ and an alphabet of terminals Σ, there exists an infinite word (stream) u over Σ, such that δ does not match any finite factor of u. The size of the smallest alphabet (w.r.t. cardinality) over which a pattern is avoidable is the *avoidability index* of that pattern. In [MMN12a] it is shown how to find for a cube under anti-/morphic permutations $x\pi^i(x)\pi^j(x)$ all the alphabets Σ_k over which the pattern is avoidable; as an interesting phenomenon, such a pattern is avoidable does not have an avoidability index, but rather an avoidability interval: it is unavoidable for very small and very large alphabets, and avoidable in between. A pattern with reversals is a pattern with functional dependencies, where all function variables are replaced with the mirror function, denoted here as $(\cdot)^r$.

A formula ϕ, as introduced by Cassaigne [Cas94], over the set X of variables is a finite set of patterns over X. A formula is avoidable in an alphabet Σ_k if there exists an infinite word over Σ_k that avoids simultaneously all the patterns in the formula. Cassaigne showed that every formula corresponds in a natural way to a pattern with the same avoidability index (see [Cas94] for details). Therefore, formulas can be seen as a natural generalization of patterns in the context of avoidability. Naturally, the notion of formulas can be also used for patterns with functional dependencies, e.g., patterns with reversals. For example, the set $\{xx, xyyzx^r, xyy^rx\}$ is a formula. In order to show that this formula is avoidable, an infinite word avoiding each of the patterns $xx, xyyzx^r, xyy^rx$ should be constructed.

For a word w over an alphabet Σ_k, the Parikh vector of w is an array Ψ_w indexed by the letters of Σ_k, such that $\Psi_w[a] = |w|_a$. Two words u and v are Abelian equivalent, denoted by $u \sim_a v$, if v and u have the same Parikh vector. For instance, $11122 \sim_a 12121, 31213 \sim_a 31312$. In other words, $u \sim_a v$ means that v is a jumbled version of u. A word $w \in \Sigma_k^\star$ realizes (or matches) in the Abelian sense the pattern $\delta \in X^*$ if there are $u_1, \ldots, u_{|P|} \in \Sigma_k^+$ such that $w = u_1 \cdots u_{|\delta|}$ and for all i, j we have that $u_i \sim_a \delta_j$ if and only if $\delta[i] = \delta[j]$. For instance, 121121 realizes the pattern xx in the Abelian sense.

2.2 Morphisms

A morphism f (respectively, antimorphism) of Σ_k^* is defined by its values on letters; $f(uv) = f(u)f(v)$ (respectively, $f(uv) = f(v)f(u)$) for all words $u, v \in \Sigma_k^*$. When we define an anti-/morphism it is enough to define $f(a)$, for all $a \in \Sigma_k$. If the restriction of f to Σ_k is a permutation of Σ_k, we call f an anti-/morphic permutation. In this case, denote by $\mathbf{ord}(f)$ the order of f, i.e., the minimum positive integer m such that f^m is the identity. If $\mathbf{ord}(f) = 2$, we call f an involution. If $a \in \Sigma_k$ is a letter, the order of a with respect to f, denoted $\mathbf{ord}_f(a)$, is the minimum number m such that $f^m(a) = a$. The fixed point of the infinite iteration $f^\omega(w)$ is the word associated to this morphism. A morphism f is called n uniform for $n \in \mathbb{N}$ if $|f(\mathtt{a})| = n$ holds for all letters $\mathtt{a} \in \Sigma$.

A pattern which involves functional dependencies is a term over (word) variables and function variables (where concatenation is an implicit functional constant); a pattern with only one word variable is called unary. For example, $x\pi(x)\pi(\pi(x))x = x\pi(x)\pi^2(x)x$ is a unary pattern involving the variable x and the function variable π. An instance of a pattern p in Σ_k is the result of substituting every variable by a word in Σ_k^+ and every function variable by a function over Σ_k^*. A pattern is avoidable in Σ_k if there is an infinite word over Σ_k that does not contain any instance of the pattern.

The infinite Thue-Morse word t is defined as $t = \lim_{n\to\infty} \phi_t^n(0)$, for the morphism $\phi_t : \Sigma_2^* \to \Sigma_2^*$ where $\phi_t(0) = 01$ and $\phi_t(1) = 10$. It is well-known (see [Lot97]) that the word t avoids the patterns xxx (cubes) and $xyxyx$ (overlaps).

The infinite ternary Thue word h is defined as $h = \lim_{n\to\infty} \phi_h^n(0)$, for the morphism $\phi_h : \Sigma_3^* \to \Sigma_3^*$ where $\phi_h(0) = 012$, $\phi_h(1) = 02$ and $\phi_h(2) = 1$. The infinite word h avoids the pattern xx (squares).

We investigate the factors of an infinite word $\mathbf{g}$ that have the form

$$\pi_{i_1}(x)\pi_{i_2}(x)\dots\pi_{i_r}(x)$$

with x a non-empty word and each π_{i_j} a morphic or antimorphic permutation for $1 \leqslant j \leqslant r$. Replacing x by $\pi_{i_1}^{-1}(x)$ and $\pi_{i_j}(x)$ by $\pi_{i_j}(\pi_{i_1}^{-1}(x))$ for $1 \leqslant j \leqslant r$, this is equivalent to investigating factors of $\mathbf{g}$ of the form $x\pi_{j_1}(x)\dots\pi_{j_{r-1}}(x)$ with x a non-empty word, and each π_{j_ℓ} a morphic or

antimorphic permutation for all $1 \leqslant \ell \leqslant r-1$.

2.3 Powers

For a nonempty word $x \in \Sigma^+$, the notation x^n, $n \in \mathbb{N}$, denotes concatenating n copies of x. A word of the form x^n is called an n-power. A 2-power is also called a square; a 3-power is also called a cube. The notion of integral powers was extended to fractional powers by Dejean [37]. Our formulation of fractional powers is based on Brandenburg [20].

A fractional power or a repetition is a word of the form $z = x^n y$, where $n \in \mathbb{Z}_{\geqslant 1}$, $x \in \Sigma^+$, and y is a proper prefix of x. Equivalently, z has a $|x|$-period and $|y| = |z| \mod |x|$. If $|z| = p$ and $|x| = q$, we say that z is a p/q-power, or $z = x^{p/q}$. For example, the word sense is a $5/3$-power, $sense = (sen)^{5/3}$. In the expression $x^{p/q}$, the number p/q is the power's exponent, and the word x is the power block.

Let $|w|$ denote the word's length for a finite word $w \in \Sigma^*$. We will call the operation of the free monoid Σ^* *product* which is also known as concatenation: for words $u, v \in \Sigma^\infty$ the product uv is the word that starts with u followed by v.

We say $u \in \Sigma^*$ is a *factor* of $w \in \Sigma^*$, if $w = xuy$ for some words $x, y \in \Sigma^*$. A word $u \in \Sigma^*$ is said to *occur strictly inside (is a proper factor of)* another word $w \in \Sigma^*$ if u is a factor of w, other than a prefix or a suffix. An infinite word $w \in \Sigma^\infty$ is called *recurrent* if every factor u of w has infinitely many occurrences in w. Two words $u, v \in \Sigma^*$ are called *conjugates* of each other if there exists $x, y \in \Sigma^*$ with $u = xy$ and $v = yx$. The finite powers of a word $w \in \Sigma^*$ are defined recursively by $w^0 = \varepsilon$, $w^n = ww^{n-1}$ for $n \geqslant 1$. If w cannot be expressed as a power of another word, then w is said to be *primitive*.

Chapter 3

Unary patterns under permutations

3.1 Introduction

In this chapter, we show that all patterns $x\pi^i(x)\pi^j(x)\pi^k(x)$ with morphic or antimorphic permutations are avoidable in Σ_3. Similarly, all such patterns under anti-/morphic permutations are also avoidable in Σ_4, but not in Σ_6. The case of Σ_5 remains open. These results are provided in section 3.3.2. Sections 3.2 and 3.3.1 appeared originally in [CMN15]. They are given here as they provide the framework in which the author's result can be best described and motivated.

3.2 A general result

We define the infinite word $\mathbf{v}$ over Σ_3' as $\mathbf{v} = \prod_{i=0}^{\infty} v_i$, with $v_i = h_i + 1$, where h the infinitie ternary Thue word defined in Preliminaries. The infinite word $\mathbf{v}$ (respectively, the word $\mathbf{h}$) avoids squares xx and does not contain the factors 121 and 323 (respectively, the factors 010 and 212).

Now, we define the word $\mathbf{u} \in \Sigma_3^{\omega}$ as

$$\mathbf{u} = \prod_{i=0}^{\infty} (0^{v_{3i}} 1^{v_{3i+1}} 2^{v_{3i+2}}).$$

Theorem 3.1. *The word* $\mathbf{u}$ *has no factor of the form* $x\pi_i(x)\pi_j(x)\pi_k(x)$ *with* $|x| \geqslant 2$ *and* π_i, π_j *and* π_k *are each a morphic or antimorphic permutation.*

Proof. **(Morphic case)** Suppose, to the contrary, that $\mathbf{u}$ has a factor $w = x\pi_i(x)\pi_j(x)\pi_k(x)$ with $|x| \geqslant 2$, where each π_r is a morphic permutation.

We consider the block structure of x; that is, we parse x as

$$x = a_1^{k_1} a_2^{k_2} \cdots a_{n-1}^{k_{n-1}} a_n^{k_n}$$

where the $a_i \in \Sigma_3$, with $a_\ell \neq a_{\ell+1}$, $k_\ell \geqslant 1$, $1 \leqslant \ell \leqslant n$. Certainly, $\pi_r(x)$ has the similar block structure for each r:

$$\pi_r(x) = (\pi_r(a_1))^{k_1} (\pi_r(a_2))^{k_2} \cdots (\pi_r(a_{n-1}))^{k_{n-1}} (\pi_r(a_n))^{k_n}$$

and letters $\pi_r(a_\ell)$ and $\pi_r(a_{\ell+1})$ are distinct, since π_r is a permutation.

We consider several cases based on n, k_1 and k_n as follows:

Case 1: $n = 1$**.** This means that $w = a_1^{k_1} (\pi_i(a_1))^{k_1} (\pi_j(a_1))^{k_1} (\pi_k(a_1))^{k_1}$.

Since $|x| \geqslant 2$, we have $k_1 \geqslant 2$. If $a_1 = \pi_i(a_1)$, then w contains the factor $a_1^{k_1} (\pi_i(a_1))^{k_1} = a_1^{2k_1}$. Since $2k_1 \geqslant 4$, this is impossible; the block lengths in **u** are 1, 2 or 3. We conclude that $a_1 \neq \pi_i(a_1)$. Similarly, $\pi_i(a_1) \neq \pi_j(a_1)$, and $\pi_j(a_1) \neq \pi_k(a_1)$. It follows that, in the context of w, $(\pi_i(a_1))^{k_1}$ and $(\pi_j(a_1))^{k_1}$ are successive blocks of **u**; however, this implies that $k_1 k_1$ is a factor of **v**. Since **v** is square-free, this is impossible.

Case 2a: $n > 1$**, and** $k_1 = 3$ **or** $k_n = 3$

Suppose $k_1 = 3$. This implies that $a_n \neq \pi_i(a_1)$; otherwise w contains a block $a_n^{k_n} (\pi_i(a_1))^3 = a_n^{k_n+3}$, of length 4 or greater. Similarly, $\pi_i(a_n) \neq \pi_j(a_1)$ and $\pi_j(a_n) \neq \pi_k(a_1)$. Each $(\pi_i(a_\ell))^{k_\ell}$ and $(\pi_j(a_\ell))^{k_\ell}$ is thus a complete block of **u**, and **v** contains the factor $(k_1 k_2 \cdots k_n)^2$. This is impossible. Similarly, one argues that $k_n = 3$ gives a contradiction.

Case 2b: $n > 1$**, and** $k_1, k_n \leqslant 2$

If $a_n = \pi_i(a_1)$ and $\pi_i(a_n) = \pi_j(a_1)$, then **u** contains the factor

$$a_{n-1} a_n^{k_n+k_1} (\pi_i(a_2))^{k_2} \ldots (\pi_i(a_{n-1}))^{k_{n-1}} \pi_i(a_n)^{k_n+k_1} (\pi_j(a_2))^{k_2} \ldots (\pi_j(a_{n-1}))^{k_{n-1}} \pi_j(a_n),$$

and **v** contains the square factor $((k_n + k_1) k_2 k_3 \cdots k_{n-1})^2$, which is impossible. Similarly, if $a_n \neq \pi_i(a_1)$ and $\pi_i(a_n) \neq \pi_j(a_1)$, then **v** contains the factor

$$a_1 a_2^{k_2} \ldots a_n^{k_n} (\pi_i(a_1))^{k_1} (\pi_i(a_2))^{k_2} \ldots (\pi_i(a_n))^{k_n} (\pi_j(a_1))^{k_1} \pi_j(a_2),$$

and then v contains the factor $(k_2k_3\cdots k_1)^2$, which is again impossible. In conclusion, exactly one of the equations $a_n = \pi_i(a_1)$ and $\pi_i(a_n) = \pi_j(a_1)$ holds. Similarly, exactly one of $\pi_i(a_n) = \pi_j(a_1)$ and $\pi_j(a_n) = \pi_k(a_1)$ holds.
Case 2bi: $k_1, k_n \leqslant 2$, and $n \geqslant 3$. Suppose $a_n = \pi_i(a_1)$ and $\pi_i(a_n) \neq \pi_j(a_1)$. (The other case is similar.)

Since $\pi_i(a_n) \neq \pi_j(a_1)$, but

$$\pi_i(a_{n-1})(\pi_i(a_n))^{k_n}(\pi_j(a_1))^{k_1}\pi_j(a_2)$$

is a factor of **u**, we see that k_nk_1 is a factor of **v**, whence $k_n \neq k_1$. Since we have already reasoned that $k_1, k_n \leqslant 2$, we see that $k_1 + k_n = 3$. Now $a_{n-2}(a_{n-1})^{k_{n-1}}a_n^3$ is a factor of **u**, so that $k_{n-1} \neq 3$. On the other hand, since

$$\pi_i(a_{n-2})(\pi_i(a_{n-1}))^{k_{n-1}}(\pi_i(a_n))^{k_n}\pi_j(a_1)$$

is a factor of **u**, and $\pi_i(a_n) \neq \pi_j(a_1)$, we conclude that $k_{n-1}k_n$ is a factor of **v**; therefore, $k_{n-1} \neq k_n$, and since $k_n, k_{n-1} < 3$, we have $k_{n-1} = 3 - k_n = k_1$.

Similar reasoning shows that $k_2 = k_n$. But then

$$\pi_i(a_{n-2})(\pi_i(a_{n-1})^{k_{n-1}}(\pi_i(a_n))^{k_n}(\pi_j(a_1))^{k_1}(\pi_j(a_2))^{k_2}\pi_j(a_3)$$

is a factor of **u**, so that $k_{n-1}k_nk_1k_2 = (k_1k_n)^2$ is a factor of **v**. This is impossible.
Case 2bii: $k_1, k_n \leqslant 2$, and $n = 2$. We make four subcases, depending on whether $(k_1, k_2) = (1, 2)$ or $(2, 1)$, and $a_2 = \pi_i(a_1)$, $\pi_i(a_2) \neq \pi_j(a_1)$ and $\pi_j(a_2) = \pi_k(a_1)$, or alternatively, $a_2 \neq \pi_i(a_1)$, $\pi_i(a_2) = \pi_j(a_1)$ and $\pi_j(a_2) \neq \pi_k(a_1)$.

1. $(k_1, k_2) = (1, 2)$, $a_2 = \pi_i(a_1)$, $\pi_i(a_2) \neq \pi_j(a_1)$, $\pi_j(a_2) = \pi_k(a_1)$:

   In this case, **u** contains the word

   $$\begin{aligned} &a_1a_2^2\pi_i(a_1)(\pi_i(a_2))^2\pi_j(a_1)(\pi_j(a_2))^2\pi_k(a_1)(\pi_k(a_2))^2 \\ &\quad = a_1a_2^3(\pi_i(a_2))^2\pi_j(a_1)(\pi_j(a_2))^3(\pi_k(a_2))^2 \end{aligned}$$

   so that **v** contains a word $\alpha 3213\beta$, $\alpha, \beta \in \{1, 2, 3\}$, $\beta \geqslant 2$. In fact, if $\beta = 3$, then **v** contains the square 3^2. Assume then that $\beta = 2$. Thus 32132 is a factor of **v**; however, 32132 has no right extension in **v**, since 321321 and 321322 end in squares, while 321323 ends in 323. This is impossible.

2. $(k_1, k_2) = (2, 1)$, $a_2 = \pi_i(a_1)$, $\pi_i(a_2) \neq \pi_j(a_1)$, $\pi_j(a_2) = \pi_k(a_1)$:

   In this case, $\mathbf{u}$ contains the word

   $$a_1^2 a_2 (\pi_i(a_1))^2 \pi_i(a_2) (\pi_j(a_1))^2 \pi_j(a_2) (\pi_k(a_1))^2 \pi_k(a_2)$$
   $$= a_1^2 a_2^3 \pi_i(a_2) (\pi_j(a_1))^2 (\pi_j(a_2))^3 \pi_k(a_2)$$

   so that $\mathbf{v}$ contains a word $\alpha 3123 \beta$, $\alpha, \beta \in \{1, 2, 3\}$, $\alpha \geqslant 2$. In fact, if $\alpha = 3$, then $\mathbf{v}$ contains 3^2. Assume then that $\alpha = 2$, so that $\mathbf{v}$ contains 23123. Since $\mathbf{v}$ is recurrent, 23123 must have a left extension in $\mathbf{v}$; however, none of 123123, 223123 and 323123 is a possible factor of $\mathbf{v}$.

3. $(k_1, k_2) = (1, 2)$, $a_2 \neq \pi_i(a_1)$, $\pi_i(a_2) = \pi_j(a_1)$, $\pi_j(a_2) \neq \pi_k(a_1)$:

   In this case, w contains the word

   $$a_1 a_2^2 \pi_i(a_1) (\pi_i(a_2))^2 \pi_j(a_1) (\pi_j(a_2))^2 \pi_k(a_1) (\pi_k(a_2))^2$$
   $$= a_1 a_2^2 \pi_i(a_1) (\pi_i(a_2))^3 (\pi_j(a_2))^2 \pi_k(a_1) (\pi_k(a_2))^2$$

   so that v contains a word $\alpha 21321 \beta$. No left extension of this word is a factor of $\mathbf{v}$.

4. $(k_1, k_2) = (2, 1)$, $a_2 \neq \pi_i(a_1)$, $\pi_i(a_2) = \pi_j(a_1)$, $\pi_j(a_2) \neq \pi_k(a_1)$: In this case, w contains the word

   $$a_1^2 a_2 (\pi_i(a_1))^2 \pi_i(a_2) (\pi_j(a_1))^2 \pi_j(a_2) (\pi_k(a_1))^2 \pi_k(a_2)$$
   $$= a_1^2 a_2 (\pi_i(a_1))^2 (\pi_i(a_2))^3 \pi_j(a_2) (\pi_k(a_1))^2 \pi_k(a_2)$$

   so that v contains a word $\alpha 12312 \beta$. No right extension of this word is a factor of $\mathbf{v}$.

We see that w contains no instance $x \pi_i(x) \pi_j(x) \pi_k(x)$ with $|x| \geqslant 2$ where each π_r is a morphic permutation.

(Antimorphic case) Suppose, for the sake of getting a contradiction, that $\mathbf{u}$ has a factor $w = x \pi_i(x) \pi_j(x) \pi_k(x)$ with $|x| \geqslant 2$, where one of the π_r is an antimorphic permutation.

For notational simplicity, we will suppose that π_i is antimorphic; the other cases are similar.

We consider the block structure of x:

$$x = a_1^{k_1} a_2^{k_2} \cdots a_{n-1}^{k_{n-1}} a_n^{k_n}$$

where the $a_i \in \Sigma_3$, with $a_\ell \neq a_{\ell+1}$, $k_\ell \geqslant 1, 1 \leqslant \ell \leqslant n$. Since π_i is antimorphic,

$$\pi_i(x) = (\pi_i(a_n))^{k_n}(\pi_i(a_{n-1}))^{k_{n-1}} \cdots (\pi_i(a_2))^{k_2}(\pi_i(a_1))^{k_1}.$$

If $k_n = 3$, then $\mathbf{u}$ has the factor $a_n^{k_n}\pi_i(a_n)^{k_n}$, and either $\mathbf{u}$ has a block of length 6 (if $a_n = \pi_i(a_n)$), or $\mathbf{v}$ has a factor 33; both cases are impossible.

If $k_n = 2$: If $n = 1$, then $w = a_1^2\pi_i(a_1^2)\pi_j(a_1^2)\pi_k(a_1^2)$, and the factor $\pi_i(a_1^2)\pi_j(a_1^2)$ of w implies that either $\mathbf{u}$ has a block of length 4, or $\mathbf{v}$ has a factor 22; both cases are impossible.

If $n > 1$, then factor $a_{n-1}^{k_n-1}a_n^{k_n}\pi_i(a_n^{k_n})\pi_i(a_{n-1}^{k_{n-1}})$ gives the same contradiction.

We conclude that $k_n = 1$. Since $|x| \geqslant 2$, we have $n \geqslant 2$. If $a_n \neq \pi_i(a_n)$, then the factor $a_{n-1}^{k_{n-1}}a_n^1\pi_i(a_n^1)\pi_i(a_{n-1}^{k_{n-1}})$ of w implies that 11 is a factor of $\mathbf{v}$, which is impossible. We conclude that $a_n = \pi_i(a_n)$.

Suppose $|x| \geqslant 3$. If $k_{n-1} = 1$, w contains $a_{n-2}^{k_{n-2}}a_{n-1}^1a_n^2\pi_i(a_{n-1}^1)\pi_i(a_{n-2})$, so that $\mathbf{v}$ has the factor 121. This is impossible. Thus $k_{n-1} > 1$. This forces $\mathbf{u}$ to contain a block $a_{n-1}^y a_n^2\pi_i(a_{n-1}^z)$, where $y, z \geqslant 2$ and $y2z$ is a factor of $\mathbf{v}$. However, then $\mathbf{v}$ has 22 or 323 as a factor, which is impossible. We conclude that $|x| = 2$. It follows that $n = 2$, $k_1 = k_2 = 1$.

Write $w = a_1a_2b_1b_2c_1c_2d_1d_2$, each $a_i, b_i, c_i, d_i \in \Sigma_3$, and $a_1 \neq a_2$, $b_1 \neq b_2$, $c_1 \neq c_2$, $d_1 \neq d_2$. We have arrived at this case by considering the word $x\pi_i(x)$, assuming that π_i is antimorphic. If, instead, π_i is morphic and π_j is antimorphic, (resp., π_i and π_j are morphic, π_k is antimorphic) the same analysis goes through considering the word $\pi_i(x)\pi_j(x)$ (resp., $\pi_j(x)\pi_k(x)$).

We must have $a_2 = b_1$, or $\mathbf{v}$ contains the square 11. Similarly, $b_2 = c_1$. Now, however, $\mathbf{v}$ contains the square 22. This is a contradiction. □

Consequently, $\mathbf{u}$ has no factor of the form $\pi_\ell(x)\pi_i(x)\pi_j(x)\pi_k(x)$ with $|x| \geqslant 2$ and π_i, π_j and π_k are each a morphic or antimorphic permutation. This means that $\mathbf{u}$ does not have a factor that contains, at its turn, an instance of a pattern $\pi_\ell(x)\pi_i(x)\pi_j(x)\pi_k(x)$ with $|x| \geqslant 2$ and π_i, π_j and π_k are each a morphic or antimorphic permutation. So, the following general theorem follows.

Theorem 3.2. *The word* $\mathbf{u}$ *has no factor of the form* $\pi_{i_1}(x)\pi_{i_2}(x)\ldots\pi_{i_r}(x)$ *with* $|x| \geqslant 2$, $r \geqslant 4$, *and* π_{i_j} *is a morphic or antimorphic permutation for* $1 \leqslant j \leqslant r$.

3.3 Avoidability for small alphabets

3.3.1 Ternary alphabets

We now show that all patterns of length at least 4 under anti-/morphisms which are powers of the same permutation are avoidable in Σ_3. More precisely, we show that for each pattern $\mathcal{P} = x\pi^i(x)\pi^j(x)\pi^k(x)$ there exists an infinite word (that depends on $\mathcal{P}$) that does not contain any instance of $\mathcal{P}$ with π an anti-/morphic permutation of Σ_3.

Let us note from the beginning that the permutations of Σ_3 are either cycles (i.e., $\mathbf{ord}(a) = 3$ for all $a \in \Sigma_3$) or involutions (i.e., $\mathbf{ord}(a) \leqslant 2$ for all $a \in \Sigma_3$).

We use as a basic lemma the following corollary of Theorem 3.1.

Corollary 3.3. *The word* $\mathbf{u}$ *has no factor of the form* $x\pi^i(x)\pi^j(x)x$*, where* π *is an anti-/morphic permutation of* Σ_3*.*

Proof. By Theorem 3.1 we know that $\mathbf{u}$ has no factor $x\pi^i(x)\pi^j(x)x$ with $|x| \geqslant 2$, by just taking in the statement of the theorem $\pi_i = \pi^i$, $\pi_j = \pi^j$, and π_k the identity on Σ_3. We assume, for the sake of contradiction, that $\mathbf{u}$ has a factor $x\pi^i(x)\pi^j(x)x$ with $|x| = 1$. Say $x = a \in \Sigma_3$. Due to the form of $\mathbf{u}$ we get that between the two occurrences of the letter $x = a$ we must find the two other letters of Σ_3 (that is, both letters from $\Sigma_3 \backslash \{a\} = \{b, c\}$). Indeed $\mathbf{u}$ does not contain a block of 4 consecutive identical letters, so the two occurrences of the letter $x = a$ belong to separate maximal blocks made of letters $x = a$ of the word $\mathbf{u}$, and between two such blocks the other two letters of Σ_3 must occur. But this would mean that $\mathbf{u}$ contains the factor *abca*, so $\mathbf{h}$ should contain the factor 11, a contradiction. □

The following lemma is immediate, as $\mathbf{v}$ avoids squares.

Lemma 3.4. *The word* $\mathbf{v}$ *has no factor of the form* $xx\pi^j(x)\pi^k(x)$*,* $x\pi^i(x)\pi^i(x)x$*,* $x\pi^i(x)\pi^j(x)\pi^j(x)$ *where* π *is an anti-/morphic permutation and* i*,* j*,* k *are non-negative integers.*

In [MMN12b] the following was shown.

Lemma 3.5. *There exists an infinite word* $\mathbf{v}_m$ *(respectively,* $\mathbf{v}_a$*) over* Σ_3 *that has no factor of the form* $x\pi(x)x$*, when* π *is replaced by a morphic (respectively, antimorphic) permutation.*

The final result we need is from [BCN12].

Lemma 3.6. *For each pattern $\mathcal{P} = x\pi^i(x)\pi^j(x)\pi^k(x)$, where i, j, k are non-negative integers, there exists an infinite word $\mathbf{u}_\mathcal{P} \in \Sigma_2^\omega$ (respectively, $\mathbf{u}'_\mathcal{P} \in \Sigma_2^\omega$) that does not contain an instance of $\mathcal{P}$ when π is replaced by a morphic (respectively, antimorphic) involution of Σ_2.*

We can now show the two main results of this section.

Lemma 3.7. *For each pattern $\mathcal{P} = x\pi^i(x)\pi^j(x)\pi^k(x)$, where i, j, k are non-negative integers, there exists an infinite ternary word $\mathbf{w}_\mathcal{P}$ that does not contain any instance of this pattern with π a morphic permutation of Σ_3.*

Proof. Clearly, each morphic permutation π of Σ_3 is either a cycle or an involution. In all cases, π^6 is the identity morphism on Σ_3^*. Consequently, we can replace the exponents i, j, k by their values modulo 6.

By Corollary 3.3 and Lemmas 3.4 and 3.5, all patterns $x\pi^i(x)\pi^j(x)\pi^k(x)$ with one of i, j, k equal to 0 and π replaced by a morphic permutation are avoidable, either by $\mathbf{v}$ (when $i = 0$), either by $\mathbf{v}_m$ (when $j = 0$), or by $\mathbf{u}$ (when $k = 0$). Similarly, the patterns $x\pi^i(x)\pi^j(x)\pi^k(x)$ with $i = k$ are avoided by $\mathbf{v}_m$, since this word does not contain instances of any pattern $\pi^i(x)\pi^j(x)\pi^i(x)$, while those with $i = j$ or $j = k$ are avoided by $\mathbf{v}$.

Consequently, we only have to consider the case when $0, i, j, k$ are pairwise distinct, and each is at most 5 in the following.

We look at the reminders of i, j and k modulo 3.

If $\{1, 2\} \subseteq \{i(\mathrm{mod}3), j(\mathrm{mod}3), k(\mathrm{mod}3)\}$, we get that when replacing π with a cycle of Σ_3 (e.g., $\pi(0) = 1, \pi(1) = 2, \pi(2) = 0$), the instance of $\mathcal{P} = x\pi^i(x)\pi^j(x)\pi^k(x)$ will contain all the three letters 0, 1, and 2. It follows that $\mathbf{u}_\mathcal{P}$ (from Lemma 3.6) avoids p. Indeed, when π is replaced by an involution of Σ_2 the result follows from the definition of $\mathbf{u}_\mathcal{P}$, while when π is replaced by any other permutation of Σ_3, its instances will contain the letter 2, so $\mathbf{u}_\mathcal{P}$ canonically avoids all of them.

So, the only case left to consider is when $\{i(\mathrm{mod}3), j(\mathrm{mod}3), k(\mathrm{mod}3)\}$ is either $\{0, 1\}$ or $\{0, 2\}$. If i, j, k are all equal modulo 3 it follows that at least two of them are actually equal, a contradiction to our earlier assumption that each two of the exponents are different. So, one of i, j, and k should be 3.

It is not hard to see that $x\pi^i(x)\pi^3(x)\pi^k(x)$ is avoided by $\mathbf{v}$. Indeed, an instance of this pattern will always contain a square. In the case when π is a cycle of Σ_3 we can only obtain words which have the form $xf(x)xf(x)$ for some morphic permutation f of Σ_3, while for π an involution the words we obtain definitely contain an instance of either xx or $\pi(x)\pi(x)$. So, in all cases, the instances of $x\pi^i(x)\pi^3(x)\pi^k(x)$ contain squares. By a similar argument, every pattern $x\pi^i(x)\pi^j(x)\pi^3(x)$ or $x\pi^3(x)\pi^j(x)\pi^k(x)$ is avoided by $\mathbf{v}$, as each instance of such a pattern contains a square. □

Lemma 3.8. *For each pattern $\mathcal{P} = x\pi^i(x)\pi^j(x)\pi^k(x)$, where i, j, k are non-negative integers, there exists an infinite ternary word $\mathbf{w}_\mathcal{P}$ that does not contain any instance of this pattern with π an antimorphic permutation of Σ_3.*

Proof. Just like in the previous proof, for each antimorphic permutation π of Σ_3, we have that π^6 is the identity morphism on Σ_3^*. Consequently, we can replace the exponents i, j, k by their values modulo 6.

Using Lemma 3.5, we get that the patterns $x\pi^i(x)\pi^j(x)\pi^k(x)$ with one of i, j, k equal to 0 and π replaced by a antimorphic permutation are avoidable, either by $\mathbf{v}$ (when $i = 0$), either by $\mathbf{v}_a$ (when $j = 0$), or by $\mathbf{u}$ (when $k = 0$). The patterns $x\pi^i(x)\pi^j(x)\pi^k(x)$ with $i = k$ are avoided by $\mathbf{v}_a$, while those with $i = j$ or $j = k$ contain squares, so are avoided by $\mathbf{v}$.

So, just like before, we only have to consider in the following the case when each two of $0, i, j, k$ are distinct and each is at most 5. And, again, if we have that $\{1, 2\} \subseteq \{i(\mod 3), j(\mod 3), k(\mod 3)\}$, we get that when replacing π with a cycle of Σ_3 the instance of $\mathcal{P} = x\pi^i(x)\pi^j(x)\pi^k(x)$ will contain all the three letters 0, 1, and 2. So, by Lemma 3.6, it follows that $\mathbf{u}'_\mathcal{P}$ avoids $\mathcal{P}$.

Hence, the only case left to consider is when $\{i(\mod 3), j(\mod 3), k(\mod 3)\}$ is either $\{0, 1\}$ or $\{0, 2\}$. The simple case is again when i, j, k are all equal modulo 3, as it follows that at least two of them are actually equal, which is a contradiction to our assumption that each two of the exponents are different. So, one of i, j, and k should be 3. A more detailed analysis is needed now.

Let us first look at patterns $x\pi^3(x)\pi^j(x)\pi^k(x)$. Obviously, j and k are not of the same parity; actually, the pair (j, k) is one of the pairs $(1, 4), (4, 1), (2, 5), (5, 2)$. Generally, when substituting π with a cycle of Σ_3, the pattern $x\pi^3(x)\pi^j(x)\pi^k(x)$ equals $xx^R\pi^j(x)\pi^k(x)$. But the instances

of $xx^R\pi^j(x)\pi^k(x)$ always contain a square: the last letter of x equals the first letter of x^R. When substituting π with an involution of Σ_3, the pattern $x\pi^3(x)\pi^j(x)\pi^k(x)$ either equals $x\pi(x)x\pi(x)$ if j is even and k is odd, or $x\pi(x)\pi(x)x$ if j is odd and k is even. Also in these cases the instances of the pattern contain squares. So, every instance of the pattern $x\pi^3(x)\pi^j(x)\pi^k(x)$ contains a square. This means that $\mathbf{v}$ avoids $x\pi^3(x)\pi^j(x)\pi^k(x)$.

Next we consider the patterns $x\pi^i(x)\pi^j(x)\pi^3(x)$. Like before, i and j do not have same parity as (i,j) must be one of pairs $(1,4),(4,1),(2,5),(5,2)$. Let us assume that i is even and j is odd. If π is a cycle, we have that a factor of the form $x\pi^i(x)\pi^j(x)\pi^3(x)$ has the form $xf(x)f(x^R)x^R$ for some morphic permutation f, which contains the square formed by the last letter of $f(x)$ and the first letter of $f(x^R)$. If π is an involution then each factor of the form $x\pi^i(x)\pi^j(x)\pi^3(x)$ starts with xx. Therefore, $\mathbf{v}$ avoids $x\pi^i(x)\pi^j(x)\pi^3(x)$ with i even and j odd. Further, we assume that i is odd and j is even. If π is a cycle, we have that a factor of the form $x\pi^i(x)\pi^j(x)\pi^3(x)$ has the form $xf(x^R)f(x)x^R$ for some morphic permutation f, which contains the square formed by the last letter of $f(x^R)$ and the first letter of $f(x)$. If π is an involution then each factor of the form $x\pi^i(x)\pi^j(x)\pi^3(x)$ is, in fact, $x\pi(x)x\pi(x)$. So, $\mathbf{v}$ avoids $x\pi^i(x)\pi^j(x)\pi^3(x)$ also for i odd and j even.

Finally, we consider the patterns $x\pi^i(x)\pi^3(x)\pi^k(x)$. Let us assume first that i is odd; consequently, k is even (the pair (i,k) is either $(1,4)$ or $(2,5)$). By Theorem 3.1, the word $\mathbf{u}$ has no instances of $x\pi^i(x)\pi^3(x)\pi^k(x)$ with $|x| \geqslant 2$. We show that $\mathbf{u}$ does not contain instances of this pattern with $|x| = 1$. Assume that $x = a \in \Sigma_3$. If π is a cycle then the factors $x\pi^i(x)\pi^3(x)\pi^k(x)$ are, in fact, *abab* with $b \in \Sigma_3$ such that $\pi^i(x) = b$. If π is an involution then the factors $x\pi^i(x)\pi^3(x)\pi^k(x)$ are *abba* with $b \in \Sigma^3$ such that $\pi(x) = b$. By the structure of $\mathbf{u}$ (which has the form $(0^+1^+2^+)^\omega$), we get that it cannot contain such factors. So $\mathbf{u}$ avoids such patterns.

We now consider the case when i is even and k is odd. Let us write the Thue-Morse word as $\mathbf{t} = \prod_{i=0}^{\infty} t_i$ with $t_i \in \{0,1\}$. Consider the word $\mathbf{t}' \in \{0,1\}^\omega$ (also used in [BCN12]) given by $\mathbf{t}' = \prod_{i=0}^{\infty} 01^{t_i+2}$.

We show now that $\mathbf{t}'$ avoids $x\pi^i(x)\pi^3(x)\pi^k(x)$ with i even and k odd. If π is a cycle then $x\pi^i(x)\pi^3(x)\pi^k(x)$ equals $xf(x)x^Rf(x^R)$ for some morphic permutation f (which is also a cycle). If x starts with 0, then $f(x)$

starts with 1, x^R ends with 0, and $f(x^R)$ ends with 1. But 0101 is not a factor $\mathbf{t}'$ (there are always at least 2 symbols 1 in a block). Thus, if $\mathbf{t}'$ contains an instance of $xf(x)x^Rf(x^R)$ with x starting with 0, then $|x| \geqslant 2$. Now, 0 is always followed by an 1, so x should start with 01. This means that $f(x)$ starts with 10, x^R ends with 10, and $f(x^R)$ ends with 01. Clearly, 01100110 is not a factor of $\mathbf{t}'$ (as this infinite word does not contain consecutive 0 letters), so if $\mathbf{t}'$ contains an instance of our pattern, then $|x| \geqslant 3$. Now, as $f(x^R)$ ends with 01 and there are no two consecutive 0's in $\mathbf{t}'$ we get that $f(x^R)$ should end with 101. This means that x should start with 010, a contradiction, as 1 letters always occur in blocks of length at least 2. In conclusion $\mathbf{t}'$ contains no instance of $x\pi^i(x)\pi^3(x)\pi^k(x)$ with x starting with 0 and π an antimorphic cycle; analogously, one can show that $\mathbf{t}'$ contains no instance of $x\pi^i(x)\pi^3(x)\pi^k(x)$ with x starting with 1 and π an antimorphic cycle. Moreover, by a very similar analysis one can show that $\mathbf{t}'$ does not contain any instance of $x\pi^i(x)\pi^3(x)\pi^k(x)$ with π an antimorphic involution. We have, thus, shown that $\mathbf{t}'$ avoids the pattern $x\pi^i(x)\pi^3(x)\pi^k(x)$ with i even and k odd.

This concludes the proof of this lemma. □

By Lemmas 3.7 and 3.8 we obtain the following theorem.

Theorem 3.9. *All patterns $x\pi^i(x)\pi^j(x)\pi^k(x)$, where i, j, k are non-negative integers, and π is substituted by an anti-/morphic permutation, are avoidable over Σ_3.*

We conclude this section with the following result, which follows from the previous theorem.

Theorem 3.10. *All patterns $\pi^{i_1}(x)\pi^{i_2}(x)\dots\pi^{i_r}(x)$ with $r \geqslant 4$, the i_j non-negative integers, and π an anti-/morphic permutation, are avoidable over Σ_3.*

3.3.2 Four and five letter alphabets: the morphic case

The morphic case

Let us consider a pattern $x\pi^i(x)\pi^j(x)\pi^k(x)$, with $i, j, k \geqslant 0$. For simplicity, the factors x, $\pi^i(x)$, π^j, or $\pi^k(x)$ of the pattern are called x-items in the following. Our analysis is based on the relation between the possible

images of the four x-items occurring in a pattern, following the ideas of [MMN12b]. For instance, we want to check whether in a possible image of our pattern, all four x-items can be mapped to a different word, or whether the second x-item and the last one can be mapped to the same word, and so on.

To achieve this, we define in Table 3.1 the numbers δ_i, $1 \leqslant i \leqslant 14$. Intuitively, they allow us to define, for a pattern $x\pi^i(x)\pi^j(x)\pi^k(x)$, which are the alphabets Σ_m in which we can model certain (non-)equality relationships between the images of the x-items. For example, in alphabets Σ_m with $m \geqslant \delta_1$ we can assign values to x and π such that the images of every two of $\pi^i(x)$, $\pi^j(x)$, and $\pi^k(x)$ are different (and this property does not hold in alphabets with less than δ_1 letters), while for Σ_m with $m \geqslant \delta_2$ we can assign values to x and π such that the images of x and $\pi^i(x)$ are equal to some word, while the images of $\pi^j(x)$ and $\pi^k(x)$ are assigned to two other words (also different between them; again, this property does not hold in smaller alphabets). To simplify, we use a simple digit-representation for any of these cases, defined in the last column of Table 3.1. In this representation, we assign different digits to the x-items that can be mapped to different words. For example, we use the representation 0123 for the case defined through δ_1 and 0012 for the case defined by δ_2. In general, when considering a δ_i, we assign a 4-digit representation to the pattern $x\pi^i(x)\pi^j(x)\pi^k(x)$ in the following manner: we start with 0, and then put a 0 on all of the remaining three positions corresponding to an x-item $\pi^t(x)$ to such that $\delta_i \mid t$. We then put a 1 on the the leftmost empty position. If the x-item on the respective position is $\pi^r(x)$, we put 1 on all empty positions s such that $\delta_i \mid (r-s)$, and so on.

Recall that $\inf \varnothing = \infty$, so some of the δ_is may be infinite. However, note that the set $\{t : t \mid i, t \mid j, t \mid k, t \mid |i-j|, t \mid |i-k|, t \mid |j-k|\}$ defining δ_1 is always non-empty, and also that $\delta_1 > 3$. Indeed, at least two of i, j, k have the same parity, so δ_1 should not divide 2. Similarly, out of $0, i, j, k$ at least two have the same reminder modulo 3, so δ_1 should also not divide 3. Let $K = \{\delta_1, \delta_2, \ldots, \delta_{14}\}$.

For a pattern $x\pi^i(x)\pi^j(x)\pi^k(x)$, we say that some δ_i and the digit-string encoding it model an instance $uf^i(u)f^j(u)f^k(u)$ of the pattern if each two of the factors $u, f^i(u), f^j(u), f^k(u)$ are equal if and only if the digits corresponding to the respective factors in the digit representation of that

Table 3.1. Definition of the values δ_a, with $1 \leqslant a \leqslant 14$.

$\delta_1 = \inf\{t : t \nmid i, t \nmid j, t \nmid k, t \nmid \lvert i-j\rvert, t \nmid \lvert i-k\rvert, t \nmid \lvert j-k\rvert\}$	0123
$\delta_2 = \inf\{t : t \mid i, t \nmid j, t \nmid k, t \nmid \lvert j-k\rvert\}$	0012
$\delta_3 = \inf\{t : t \nmid i, t \mid j, t \nmid k, t \nmid \lvert i-k\rvert\}$	0102
$\delta_4 = \inf\{t : t \nmid i, t \nmid j, t \mid \lvert i-k\rvert\}$	0121
$\delta_5 = \inf\{t : t \nmid i, t \nmid j, t \nmid \lvert i-j\rvert, t \nmid \lvert i-k\rvert, t \mid \lvert j-k\rvert\}$	0122
$\delta_6 = \inf\{t : t \mid i, t \mid j, t \nmid k\}$	0001
$\delta_7 = \inf\{t : t \mid i, t \nmid j, t \mid k\}$	0010
$\delta_8 = \inf\{t : t \nmid i, t \mid j, t \mid k\}$	0100
$\delta_9 = \inf\{t : t \nmid i, t \mid \lvert i-j\rvert, t \mid \lvert i-k\rvert\}$	0111
$\delta_{10} = \inf\{t : t \mid i, t \nmid j, t \mid \lvert j-k\rvert\}$	0011
$\delta_{11} = \inf\{t : t \nmid i, t \mid j, t \mid \lvert i-k\rvert\}$	0101
$\delta_{12} = \inf\{t : t \nmid i, t \mid k, t \mid \lvert i-j\rvert\}$	0110
$\delta_{13} = \inf\{t : t \nmid i, t \nmid k, t \mid \lvert i-j\rvert\}$	0112
$\delta_{14} = \inf\{t : t \nmid i, t \nmid j, t \mid \lvert i-j\rvert\}$	0120

δ_i are equal.

Lemma 3.11. *The pattern $x\pi^i(x)\pi^j(x)\pi^k(x)$, with $i \neq j \neq k \neq i$ is unavoidable in Σ_m, for $m \geqslant \max\{\delta_1, \delta_3, \delta_6, \delta_{12}, \delta_{13}\}$.*

Proof. Let $p = x\pi^i(x)\pi^j(x)\pi^k(x)$.

Because $m \geqslant \delta_1$, we have that for every word $u \in \Sigma_m^+$ there exists a morphic permutation f such that every two words of $u, f^i(u), f^j(u), f^k(u)$ are different. Indeed, we take f to be a permutation such that the orbit of $u[1]$ is a cycle of length δ_1, which means that the the first letters of $u, f^i(u), f^j(u)$ and $f^k(u)$ are pairwise different. Similarly, the fact that $m \geqslant \delta_3$ (when $\delta_3 \neq \infty$) means that for every $u \in \Sigma_m^+$ there exists a morphic permutation f such that $f^i(u) \neq u = f^j(u) \neq f^k(u) \neq f^i(u)$. In this case, we take f to be a permutation such that $\mathbf{ord}_f(u[1]) = \delta_3$. We can derive similar observations for δ_6, δ_{12}, and δ_{13}, as well as for all the δ_i values we defined.

One can check with the aid of a computer, by a straightforward backtracking algorithm, that if $m \geqslant \max\{\delta_1, \delta_3, \delta_6, \delta_{12}, \delta_{13}\}$ then the longest word over Σ_m that does not contain an instance of this pattern has length 14. Our computer program (available at `https://www.dropbox.com/s/5q5yavloti9nm3h/lemma6.rb`) tries to construct a word as long as possible by always adding

a letter to the current word it constructed by backtracking; this letter is chosen in all possible ways from the letters contained in the word already, or it may also be a new letter. An example of one of the longest words our program constructed, over an alphabet of size greater or equal to m, which is at its turn greater or equal to 3 as $\delta_1 \geqslant 3$, is 00120120120111 (adding new letters to this word does not lead to a longer one). This concludes our proof. □

In the following lemmas we show a series of avoidability and unavoidability results. Our first result uses the morphism $\delta : \Sigma_3^* \to \Sigma_4^*$ that is defined by

$$0 \to 01220112, \qquad 1 \to 0, \qquad 2 \to 03110223$$

Lemma 3.12. *Consider the infinite word:*
$\boldsymbol{h}_\delta = \delta(\boldsymbol{h}) = 01220112003110223012201120311022300122011200311022 30\ldots$

If $\boldsymbol{h}_\delta$ contains an instance of the pattern $x\pi^i(x)\pi^j(x)\pi^k(x)$ then this instance is not modelled by any element of the set

$$\{\delta_3, \delta_4, \delta_6, \delta_7, \delta_8, \delta_9, \delta_{10}, \delta_{11}, \delta_{12}, \delta_{14}\}.$$

Proof. The maximum length of a factor of $\mathbf{h}_\delta$ that does not contain a full image of the letter 2 from the Hall word under δ is 24. Using a computer program (available at `https://www.dropbox.com/s/faeyam3lb5kky59/lemma7.rb`) we checked that, there is no factor of the form $uf(u)g(u)h(u)$ with $|u| < 25$ which can be modelled by any of the δ_is mentioned above (with f, g, and h morphic permutations). By this we mean that there is no factor $uf(u)g(u)h(u)$ of $\mathbf{h}_\delta$, with $|u| < 25$, such that two of the factors $u, f(u), g(u)$,
$h(u)$ are equal if and only if the digits on their respective positions (i.e., 1, 2, 3, and, respectively, 4) in δ_i are equal. Furthermore, if u is a word of length $\geqslant 25$, and $\mathbf{h}_\delta$ contains an instance $uf^i(u)f^j(u)f^k(u)$ of the pattern $x\pi^i(x)\pi^j(x)\pi^k(x)$, then u contains the factor 3110223. Based on the repetitions of 1, 2 and 3, the factor 3110223 of u should be aligned with some factors of the form *abbcdda* from $f^i(u)$, $f^j(u)$ and $f^k(u)$, respectively. The only possible such alignment is to align 3110223 from u with other occurrences of 3110223. This means that f^i, f^j, f^k are all the identity, so $\mathbf{h}_\delta$ contains a 4-power u^4, with $|u| \geqslant 25$. Looking at the occurrence of u^4 in $\mathbf{h}_\delta$ we get that the i^{th} occurrence of u in this repetition can be written

as $u_i' h(x_{1,i}) h(x_{2,i}) \dots h(x_{t,i}) u_i''$ where $1 \leqslant i \leqslant 4$, $x_{1,i}, \dots, x_{k,i} \in \{0,1,2\}$ and, for $1 \leqslant j \leqslant 3$ we have $u_j'' u_{j+1}' = h(x_j)$ for some letter $x_i \in \{0,1,2\}$. As the image of each letter under h starts with 0, and none starts with 02 nor with 011, we get that $u_1' = u_2' = u_3' = u_4'$. Thus, $u_1' = u_2'' = u_3''$ (so, $x_1 = x_2$ as well), and $x_{j,1} = x_{j,2}$ for $1 \leqslant j \leqslant k$. Accordingly, **h** should contain a square, which is a contradiction. □

Lemma 3.13. *Let $K' = \{\delta_{i_1}, \delta_{i_2}, \delta_{i_3}\} \subset K$ be any subset of size 3 of K. There exists an infinite word w such that w does not contain 4-powers and if w contains an instance of the pattern $x\pi^i(x)\pi^j(x)\pi^k(x)$ then it can not be modelled by any tuples of the set of patterns K'.*

Proof. We consider all possible combinations of size three of δ_is, and we check the avoidability of each combination.

To begin with, assume that $\delta_{i_1}, \delta_{i_2}, \delta_{i_3}$ are all δ_is whose representations contain at least three different digits. Then if the word **t** contains an instance of the pattern $x\pi^i(x)\pi^j(x)\pi^k(x)$, it can not be modelled by any $\delta_i \in K'$, as such a δ_i can model only instances of the pattern over an alphabet of size greater or equal to 3.

Assume now $\delta_{i_1}, \delta_{i_2}, \delta_{i_3}$ are all δ_is whose representations contain at most two different digits (e.g., δ_6, δ_7, etc.). Then if the word **h** contains an instance of the pattern $x\pi^i(x)\pi^j(x)\pi^k(x)$, it can not be modelled by any $\delta_i \in K'$, since **h** does not contain any square, but these δ_is can only model instances of the pattern that contain at least one square.

If two of $\delta_{i_1}, \delta_{i_2}, \delta_{i_3}$ are among the δ_is whose representation has at least three different digits, and the other one of them is a δ_i whose representation has at most two different digits, then, similar to the previous section, we can show that $\mathbf{u}_\mathcal{P}$ (from Lemma 3.6) does not contain any instance of pattern that can be modelled by the respective δ_i. Indeed, this word does not contain instances of the pattern modelled by the any $k \in K$ that can be represented with only two digits (as such an instance could also be modelled with the restriction that π is replaced by an involution), and the remaining two δ_is can, once more, model only instances of the pattern over an alphabet greater or equal to 3.

Assume two of $\delta_{i_1}, \delta_{i_2}, \delta_{i_3}$ are among the δ_is whose representation has at most two different digits, and one of them is any δ_i except $\delta_3, \delta_4, \delta_{14}$, which is represented using at least three different digits. Then **h** does not

contain instances of patterns that can be modelled by any $\delta_i \in K'$, because all such δ_is model only instances of the pattern that contain squares or have 4 different letters (e.g., the instances modeled by δ_1).

Assume two of $\delta_{i_1}, \delta_{i_2}, \delta_{i_3}$ are among δ_is whose representation has at most two different digits, and one of them is δ_3 or δ_4 or δ_{14}, the word defined in Lemma 3.12 can avoid them. □

Lemma 3.14. *For each pattern $\mathcal{P} = x\pi^i(x)\pi^j(x)\pi^k(x)$, where i, j, k are non-negative integers, there exists an infinite word $\boldsymbol{w}_p$ that does not contain any instance of this pattern with π a morphic permutation of Σ_4.*

Proof. In this proof, we do a case analysis depending on the possible permutations of Σ_4 that π can be assigned to.

If π is assigned to the identical permutation 1_{Σ_4}, then the image obtained is a 4-power. If π is assigned to a 4-cycle f, then any instance $uf^i(u)f^j(u)f^k(u)$ is equal to the word $uf^{i \bmod 4}(u)f^{j \bmod 4}(u)f^{k \bmod 4}(u)$. In this case, if we have two different exponents then also the factors corresponding to them are different. Accordingly, these instances (which are not 4-powers) are modelled by exactly one δ_i, called δ_{i_1} in the following. If π is assigned a permutation f that permutes in a cycle three elements of Σ_4 and fixes the remaining one, then any instance $uf^i(u)f^j(u)f^k(u)$ is either a 4-power, if $u = a^k$ and a is the fixed point of f, otherwise we have $uf^{i \bmod 3}(u)f^{j \bmod 3}(u)f^{k \bmod 3}(u)$. Again these instances are modelled by exactly one δ_i, called δ_{i_2} in the following, and a 4-power. Assume π is mapped to a permutation f that is the composition of two disjoint cycles of length 2, or f consists of a cycle of length 2 and two fixed points (in other words, f is an involution). Then any instance $uf^i(u)f^j(u)f^k(u)$ is either the identity, if $u \in \{a, b\}^*$ and a, b are fixed point of f, or it is equal to $uf^{i \bmod 2}(u)f^{j \bmod 2}(u)f^{k \bmod 2}(u)$, otherwise. Yet again, these instances are modelled by exactly one δ_i, called δ_{i_3}, and a 4-power.

Our statement follows now from Lemma 3.13, as there exists a word that does not contain any instance of the pattern that is a 4-power or modelled by $\{\delta_{i_1}, \delta_{i_2}, \delta_{i_3}\}$. □

As an example, for $i = 3, j = 16, k = 2$, if π is mapped to a 4-cycle, we can obtain instances which are 4-powers or are modelled by 0102. If π is mapped to a 3-cycle and a fixed point, the instances we obtain are

4-powers or are modelled by 0012, and if π is mapped to an involution, we can obtain instances which are 4-powers or are modelled by 0100. All such instances can be avoided by a single infinite word, according to Lemma 3.13.

According to the previous lemmas, we can prove the following theorems.

Theorem 3.15. *All patterns $x\pi^i(x)\pi^j(x)\pi^k(x)$, where i, j, k are non-negative integers, and π is substituted by a morphic permutation, are avoidable over Σ_4.*

Theorem 3.16. *All patterns $\pi^{i_1}(x)\pi^{i_2}(x)\dots\pi^{i_r}(x)$ with $r \geqslant 4$, the i_j non-negative integers, and π a morphic permutation, are avoidable over Σ_4.*

The antimorphic case

Now we try to check these results for the antimorphic case too, but this case was a more complicated, since when we had to take into account the parity of the exponents i, j, k, as an odd exponent means that the image of the word replacing x is the catenation of the images of its letters, under the permutation iterated as many times as the exponent indicates, in reverse order. To that end, we needed to generate more words that avoid different sets of cases and this took a long time for us until we were able to generate these words using Ruby programs running on the server. Furthermore, more complicated analysis were needed to consider different cases that would be generated with both odd and even parties of the powers of the π functions.

We now show a series of similar results for the case when the function variable π is replaced by an antimorphic permutation. For uniformity, we use the same notations as in Table 3.1 . However, a finer case distinction should be made in this case. Namely, as before, we can associate to a pattern $x\pi^i(x)\pi^j(x)\pi^k(x)$ one of the values δ_i as in the respective table. The difference is that when we define the digit-representation of δ_i one should also take into account the parity of the exponents i, j, k, as an odd exponent means that the image of the word replacing x is the catenation of the images of its letters, under the permutation iterated as many times as the exponent indicates, in reverse order. To this end, we associate to a value δ_i a canonical digit-representation $0d_1d_2d_3$. Then, we define the

derived digit-representations of δ_i: $\{0e_1e_2e_3 \mid e_\ell = d_\ell \text{ or } e_\ell = d'_\ell, 1 \leqslant \ell \leqslant 3\}$. The set of digit-representations of a value δ_i consists in its canonical digit-representation (d_ℓ) and its derived digit-representation (d'_ℓ) which denotes that the parity of the exponent is odd and we apply the permutation on x in a reverse order). As such, each δ_i has 8 digit-representations. For instance δ_2 has the digit-representations 0012, 0012′, 001′2, 001′2′, 00′12, 00′1′2, 00′12′, 00′1′2′.

The sequence $0e_1e_2e_3$ models the instances of pattern $x\pi^i(x)\pi^j(x)\pi^k(x)$ if $0e_1e_2e_3$ is one of the digit-representations of the value δ_i associated to that pattern and e_ℓ is a primed digit (i.e., it belongs to $\{0', 1', 2', 3'\}$) if and only if the exponent of x-item occurring in the pattern on position $\ell + 1$ is odd. For example, the instances of $xx\pi(x)\pi^2(x)$ can be modelled by 001′2. Moreover, δ_i models the instances of all patterns $x\pi^i(x)\pi^j(x)\pi^k(x)$ if $0e_1e_2e_3$ whose instances are modelled by the digit-representations of δ_i.

Lemma 3.17. *The pattern $x\pi^i(x)\pi^j(x)\pi^k(x)$, with $i \neq j \neq k \neq i$ is unavoidable in Σ_m, for $m \geqslant \max\{\delta_1, \delta_3, \delta_6, \delta_{12}, \delta_{13}\}$.*

Proof. Once more, by a backtracking algorithm, we get that if $m \geqslant \max\{\delta_1, \delta_3, \delta_6, \delta_{12}, \delta_{13}\}$, the longest word that does not contain an instance of this pattern has length 13, and one example of such word over an alphabet of size greater or equal to 3 is 0210212121222. For example, to check whether our word does not contain instances of this set of patterns, we start checking the case that the parity of all i, j and k are even, so in this case we should check whether our word does not contain instances of the patterns 0000, 0123, 0102, 0001, 0110, 0112. In the next step we consider the parity of i and j to be even, and the parity of k to be odd, so we should check whether our word does not contain instances of the patterns 0000′, 0123′, 0102′, 0001′, 0110′, 0112′. Since each time we should also check whether our word does not contain an instance of a four-power, in this case that k is odd, the identity acts as the mirror image, so we need to check 0000′. We continue these checking, and in the end, as a final case, we consider the parity of all i, j and k to be odd, and we check whether our word does not contain instances of the patterns 00′0′0′, 01′2′3′, 01′0′2′, 00′0′1′, 01′1′0′, 01′1′2′. □

As in the morphic case, we first establish a series of results regarding

basic patterns. To begin with, we introduce the morphism $\delta : \Sigma_3^\star \to \Sigma_4^\star$ defined by

$0 \to 0012201123, \qquad 1 \to 0021300123, \qquad 2 \to 0023302113$

Lemma 3.18. *Consider the infinite word:*
$\boldsymbol{h}_\delta = \delta(\boldsymbol{h}) = 00122011230021300123002330211300122011230023302113\ldots$
If $\mathbf{h}_\delta$ contains an instance of the pattern $x\pi^i(x)\pi^j(x)\pi^k(x)$ then this instance can not be modelled by any tuple of the set $\{\delta_3, \delta_4, \delta_6, \delta_7, \delta_8, \delta_9, \delta_{10}, \delta_{11}, \delta_{12}, \delta_{14}\}$.

Proof. The maximum length of a factor of $\mathbf{h}_\delta$ that does not contain the image of letter 0 from the Hall word under δ is 48. Using a computer program we checked that, there is no factor of the form $uf(u)g(u)h(u)$ with $|u| \leqslant 48$ which can be modelled by the any of the δ_is mentioned above (with f, g, and h morphic or antimorphic permutations). Here, this means that there is no factor $uf(u)g(u)h(u)$ of $\mathbf{h}_\delta$ with $|u| \leqslant 48$ and f, g, h both morphic or antimorphic permutations, such that two of the factors $u, f(u), g(u), h(u)$ are equal if and only if the digits on their respective positions in δ_i are equal. Further, if u is a word of length $\geqslant 49$, and $\mathbf{h}_\delta$ contains an instance of the pattern, then all the words u, $f(u)$, $g(u)$, $h(u)$ contain as a factor the prefix 00122011 of the image of 0 under δ.

So, let us assume $\mathbf{h}_\delta$ contains a factor $uf^i(u)f^j(u)f^k(u)$, with f antimorphic permutation, and $x\pi^i(x)\pi^j(x)\pi^k(x)$ modelled by any tuple of the set given in the statement. If all i, j, and k are even, then the occurrence of 0012201123 from u must be aligned to a factor *aabccabbcd* of $f^t(u)$, where t is an even value from i, j, k. The only possibility is that the factor *aabccabbcd* of $f^t(u)$ is also 0012201123. This means that f^t is the identity, so, it follows that $f^i(u) = u$ (respectively, $f^j(u) = u$, or $f^k(u) = u$). If i is even (or, respectively, both j and k are even), we immediately get that h_δ contains the square $uf^i(u)$ (or, respectively, $f^j(u)f^k(u)$). Looking at the form of the morphism δ we get immediately that such a square must be the image of a square in $\mathbf{h}$, a contradiction. A similar conclusion can be reached, in an analogous way, when i, j are both odd. Thus, the only remaining case is that i, k are odd and j is even. As above, we get $f^j(u) = u$ and $f^i(u) = f^k(u)$. So, again, we have the square $uf^i(u)uf^i(u) = uf^i(u)f^j(u)f^k(u)$ in $\mathbf{h}_\delta$. This square must be the image of a square from $\mathbf{h}$, a contradiction.

It is worth noting that $\mathbf{h}_\delta$ does not contain any factor $uf^i(u)f^j(u)f^k(u)$ with $|u| \geqslant 49$, no matter how i, j, k are chosen. However, it contains factors

of the form $uf^i(u)f^j(u)f^k(u)$ with $|u| \leqslant 48$, only that, in that case, i, j, k are not modelled by the δ_is from the statement.

□

Let $\delta : \Sigma_3^* \to \Sigma_2^*$ be the morphism that is defined by
$0 \to 00010001, \quad 1 \to 00010101, \quad 2 \to 00010001$

Lemma 3.19. *Consider the infinite word:*
$\boldsymbol{h}_\delta = \delta(\boldsymbol{h}) = 0001000100010101000101010001000100010101000101010001\ldots$
If $\boldsymbol{h}_\delta$ contains an instance of the pattern $x\pi^i(x)\pi^j(x)\pi^k(x)$ then this instance can not be modelled by any tuple of the set $\{\delta_1, \delta_2, \delta_3, \delta_4, \delta_5, \delta_{10}, \delta_{12}, \delta_{13}, \delta_{14}\}$.

Proof. The maximum length of a factor of $\mathbf{h}_\delta$ that does not contain the image of letter 0, 1 or 2 from the Hall word under δ is 38. Using a computer program we checked that, there is no factor of the form $uf(u)g(u)h(u)$ with $|u| \leqslant 38$ which can be modelled by the any of the δ_is mentioned above (with f, g, and h morphic or antimorphic permutations). Considering the occurrence of 00010101 from u, and based on the parity of i, j, and k, we can apply the same analysis used in Lemma 3.18 to prove this Lemma. If u is a word of length $\geqslant 38$, and $\mathbf{h}_\delta$ contains an instance of the pattern, then all the words u, $f(u)$, $g(u)$, $h(u)$ contain as a factor the prefix 00010101 of the image of 0 under δ.

So, let us assume $\mathbf{h}_\delta$ contains a factor $uf^i(u)f^j(u)f^k(u)$, with f antimorphic permutation, and $x\pi^i(x)\pi^j(x)\pi^k(x)$ modelled by any tuple of the set given in the statement. If all of i, j, and k are even, then the occurrence of 00010101 from u must be aligned to a factor *aaababab* of $f^t(u)$, where t is an even value from i, j, k. The only possibility is that the factor *aaababab* of $f^t(u)$ is also 00010101. This means that f^t is the identity, so, it follows that $f^i(u) = u$ (respectively, $f^j(u) = u$, or $f^k(u) = u$). If i is even (or, respectively, both j and k are even), we immediately get that h_δ contains the square $uf^i(u)$ (or, respectively, $f^j(u)f^k(u)$). Looking at the form of the morphism δ we get immediately that such a square must be the image of a square in $\mathbf{h}$, a contradiction. A similar conclusion can be reached, in an analogous way, when i, j are both odd. Thus, the only remaining case is that i, k are odd and j is even. As above, we get $f^j(u) = u$ and $f^i(u) = f^k(u)$. So, again, we have the square $uf^i(u)uf^i(u) = uf^i(u)f^j(u)f^k(u)$ in $\mathbf{h}_\delta$. This square must be the image of a square from $\mathbf{h}$, a contradiction.

It is worth noting that $\mathbf{h}_\delta$ does not contain any factor $uf^i(u)f^j(u)f^k(u)$ with $|u| \geqslant 39$, no matter how i, j, k are chosen. However, it contains factors of the form $uf^i(u)f^j(u)f^k(u)$ with $|u| \leqslant 38$, only that, in that case, i, j, k are not modelled by the δ_is from the statement. □

Let $\delta : \Sigma_3^* \to \Sigma_2^*$ be the morphism that is defined by
$0 \to 00010001, \quad 1 \to 00010011, \quad 2 \to 00010011$

Lemma 3.20. *Consider the infinite word:*
$\boldsymbol{h}_\delta = \delta(\boldsymbol{h}) = 0001000100010011000100110001000100010011000100110001\ldots$
If $\boldsymbol{h}_\delta$ contains an instance of the pattern $x\pi^i(x)\pi^j(x)\pi^k(x)$ then this instance can not be modelled by any tuple of the set $\{\delta_1, \delta_2, \delta_3, \delta_4, \delta_5, \delta_{11}, \delta_{13}, \delta_{14}\}$.

Proof. The maximum length of a factor of $\mathbf{h}_\delta$ that does not contain the image of letter 0, 1 or 2 from the Hall word under δ is 38. Using a computer program we checked that, there is no factor of the form $uf(u)g(u)h(u)$ with $|u| \leqslant 38$ which can be modelled by the any of the δ_is mentioned above (with f, g, and h morphic or antimorphic permutations). Considering the occurrence of 00010011 from u, and based on the parity of i, j, and k, we can apply the same analysis used in Lemma 3.18 to prove this Lemma. If u is a word of length $\geqslant 38$, and $\mathbf{h}_\delta$ contains an instance of the pattern, then all the words u, $f(u)$, $g(u)$, $h(u)$ contain as a factor the prefix 00010011 of the image of 0 under δ.

So, let us assume $\mathbf{h}_\delta$ contains a factor $uf^i(u)f^j(u)f^k(u)$, with f antimorphic permutation, and $x\pi^i(x)\pi^j(x)\pi^k(x)$ modelled by any tuple of the set given in the statement. If all i, j, and k are even, then the occurrence of 00010011 from u must be aligned to a factor *aaababab* of $f^t(u)$, where t is an even value from i, j, k. The only possibility is that the factor *aaabaabb* of $f^t(u)$ is also 00010011. This means that f^t is the identity, so, it follows that $f^i(u) = u$ (respectively, $f^j(u) = u$, or $f^k(u) = u$). If i is even (or, respectively, both j and k are even), we immediately get that h_δ contains the square $uf^i(u)$ (or, respectively, $f^j(u)f^k(u)$). Looking at the form of the morphism δ we get immediately that such a square must be the image of a square in $\mathbf{h}$, a contradiction. A similar conclusion can be reached, in an analogous way, when i, j are both odd. Thus, the only remaining case is that i, k are odd and j is even. As above, we get $f^j(u) = u$ and $f^i(u) = f^k(u)$. So, again, we have the square $uf^i(u)uf^i(u) = uf^i(u)f^j(u)f^k(u)$ in $\mathbf{h}_\delta$. This square must be the image of a square from $\mathbf{h}$, a contradiction.

It is worth noting that $\mathbf{h}_\delta$ does not contain any factor $uf^i(u)f^j(u)f^k(u)$ with $|u| \geqslant 39$, no matter how i, j, k are chosen. However, it contains factors of the form $uf^i(u)f^j(u)f^k(u)$ with $|u| \leqslant 38$, only that, in that case, i, j, k are not modelled by the δ_is from the statement. □

Lemma 3.21. *Let $K' = \{\delta_{i_1}, \delta_{i_2}, \delta_{i_3}\} \subset K$. There exists an infinite word w such that w does not contain 4-powers and if w contains an instance of the pattern $x\pi^i(x)\pi^j(x)\pi^k(x)$ then it can't be modelled by any tuples of the set of patterns K'.*

Proof. In this entire proof we consider digit d and d' to be similar to each other, then for all possible combinations of size three of δ_is, we check the avoidability of each combination.

To begin with, assume that $\delta_{i_1}, \delta_{i_2}, \delta_{i_3}$ are any δ_is whose representation has at least three different digits. Then if the word $\mathbf{t}$, which is over an alphabet of size 2, contains an instance of the pattern $x\pi^i(x)\pi^j(x)\pi^k(x)$, it can not be modelled by any $\delta_i \in K'$, as such a δ_i can model only instances of the pattern over an alphabet greater or equal to 3.

Assume now $\delta_{i_1}, \delta_{i_2}, \delta_{i_3}$ are member of the set $\{\delta_6, \delta_7, \delta_8, \delta_9, \delta_{10}, \delta_{11}, \delta_{12}\}$. If parity of all the exponents i, j, k are even, then the representation of δ_is have at most two different digits, it always contain square. Since the longest square free word over two letter alphabet has length three, then any word over two letter alphabet does not contain an instances of such patterns. If the parity of all i, j, k are not even, then we may have cases like $0101'$ and $011'0$ which does not contain squares, so a square free word can no avoid it, but the word defined in Lemma 3.18 does not have instances of any combination of such patterns.

If two of $\delta_{i_1}, \delta_{i_2}, \delta_{i_3}$ are among the δ_is whose representation has at least three different digits, and the other one of them is a δ_i whose representation has at most two different digits, then if the δ_i whose representation has at most two different digits is a member of the set $\{\delta_6, \delta_7, \delta_8, \delta_9\}$, then either it contains squares or factors like xx' which contain squares. For example, if $x = 012$ then the instance of xx' will be 012210 which contains a square. In these cases, the word that can be modelled by these patterns can be avoided by the Hall word. If the δ_i whose representation has at most two different digits is a member of the set $\{\delta_{10}, \delta_{11}, \delta_{12}\}$, then words defined in Lemma 3.19, 3.20 can avoid them.

Assume two of $\delta_{i_1}, \delta_{i_2}, \delta_{i_3}$ are any of δ_is whose representation has at most two different digits, and one of them is any δ_i except $\delta_3, \delta_4, \delta_{14}$, which is represented using at least three different digits. Then again either we have squares or factors like xx' which contain squares, so the word $\mathbf{h}$ does not contain instances of patterns that can be modelled by any $\delta_i \in K'$.

Assume two of $\delta_{i_1}, \delta_{i_2}, \delta_{i_3}$ are any of δ_is whose representation has at most two different digits, and one of them is δ_3 or δ_4 or δ_{14}, then the word defined in Lemma 3.18 can avoid them. □

We can, therefore, obtain the following lemma.

Lemma 3.22. *For each pattern $\mathcal{P} = x\pi^i(x)\pi^j(x)\pi^k(x)$, where i, j, k are non-negative integers, there exists an infinite word $\boldsymbol{w}_p$ that does not contain any instance of this pattern with π an antimorphic permutation of Σ_4.*

Proof. If π is assigned the identical permutation 1_{Σ_4}, then the image obtained is a 4-power, in the case that i, j, and k are all even, and if one of the powers is odd, the identity acts as the mirror image, and we will have factors like $0000'$, $000'0$, $000'0'$, $00'00$, $00'00'$, $00'0'0$, $00'0'0'$ which contain squares and can be avoided by the Hall word. If π is assigned a 4-cycle f, then any instance $uf^i(u)f^j(u)f^k(u)$ is, actually, equal to the word $uf^{i \bmod 4}(u)f^{j \bmod 4}(u)f^{k \bmod 4}(u)$. In this case, if we have two different exponents then also the factors corresponding to them are different. Accordingly, these instances (which are not 4-powers) are modelled by exactly one δ_i.

If π is assigned a permutation f that permutes cyclicly three elements of Σ_4 and fixes the remaining one, then any instance $uf^i(u)f^j(u)f^k(u)$ is either a 4-power, if $u = a^k$ and a is the fixed point of f, otherwise $uf^{i \bmod 6}(u)f^{j \bmod 6}(u)f^{k \bmod 6}(u)$. Indeed, in the later case, although f^{6k+3} and f^{6k} are both the identity on letters, they act differently on words: the first is the mirror image, while the second is the identity. However, once more, these instances are modelled by exactly one δ_i and a 4-power.

Assume π is mapped to a permutation f that is the composition of two disjoint cycles of length 2, or f consists of a cycle of length 2 and two fixed points (in other words, f is an involution). Then any instance $uf^i(u)f^j(u)f^k(u)$ is either the identity, if $u \in \{a, b\}^*$ and a, b are fixed point of f, or $uf^{i \bmod 2}(u)f^{j \bmod 2}(u)f^{k \bmod 2}(u)$, otherwise. Yet again, these

instances are modelled by exactly one δ_i and a 4-power. Our statement follows now from Lemma 3.21. □

As an example, for $i = 3, j = 16, k = 2$, if π is mapped to a 4-cycle, we can obtain instances which are like 00′00 or are modelled by 01′02. If π is mapped to a 3-cycle and a fixed point, the instances we obtain are like 00′00 or are modelled by 00′12, and if π is mapped to an involution, we can obtain instances which are like 00′00 or are modelled by 01′00. All such instances can be avoided by a single infinite word, according to Lemma 3.22.

The above results are close to optimal, in the sense that they cannot be extended for Σ_6. This is shown by the next lemma. The case of Σ_5 remains open.

Lemma 3.23. *There exists a pattern $x\pi^i(x)\pi^j(x)\pi^k(x)$, with $i \neq j \neq k \neq i$ which is unavoidable in Σ_m, for $m \geqslant 6$ with π an antimorphic permutation .*

Proof. To show that there exists a pattern $x\pi^i(x)\pi^j(x)\pi^k(x)$ which is unavoidable in Σ_m it is enough to find i, j and k such that the instances of the pattern $x\pi^i(x)\pi^j(x)\pi^k(x)$ are all modelled by the set $\{\delta_1, \delta_3, \delta_6, \delta_{12}, \delta_{13}\}$, from Lemma 4.1.

To this end, let us consider the pattern $x\pi^2(x)\pi^{56}(x)\pi^{33}(x)$. Note that $\max\{\delta_1, \delta_3, \delta_6, \delta_{12}, \delta_{13}\} = 6$.

We show that this pattern is unavoidable in Σ_6. Indeed, if π is mapped to a 6-cycle, the instances we obtain are modelled by 0112. If π is mapped to a permutation composed of a 5-cycle and a fixed point, the instances we obtain are either 4-powers or modelled by 0123. If π is mapped to a permutation composed of a 4-cycle and two fixed points, the instances we obtain are either 4-powers or modelled by 0102. If π is mapped to a permutation composed of a 4-cycle and a 2-cycle, the instances we obtain are again modelled by 0112. If π is mapped to an involution, the instances we obtain are modelled by 0001. If π is mapped to the composition of a 3-cycle and three fixed points, then the words we obtain are either 4-powers or are modelled by 0110. If π is mapped to the composition of a 3-cycle, a 2-cycle, and a fixed point, then the instances we obtain are either 4-powers, or are modelled by 0112, 0110, or 0001. Conversely, all words 0123, 0001, 0102, 0110, and 0112 are instances of the pattern $x\pi^2(x)\pi^{56}(x)\pi^{33}(x)$ for some choice of x and π.

Putting all these together, by Lemma 4.1, we get that the pattern $x\pi^2(x)\pi^{56}(x)\pi^{33}(x)$ is unavoidable. □

Thus, we have shown the following theorem.

Theorem 3.24. *All patterns $x\pi^i(x)\pi^j(x)\pi^k(x)$, with $i, j, k > 0$ and π antimorphic permutation, are avoidable in Σ_m, $m = 4$. There are patterns $x\pi^i(x)\pi^j(x)\pi^k(x)$, with $i, j, k > 0$ and π antimorphic permutation, which are unavoidable in Σ_m, for all $m \geqslant 6$.*

Chapter 4

Unary Patterns of Size Four with Morphic Permutations

4.1 Introduction

In this chapter, we investigate the avoidability of unary patterns of size of four with morphic permutations. We consider only unary patterns (i.e., containing only one variable) with morphic permutations, that is, all function variables are unary and are substituted by morphic permutations only. The structure of the chapter is as follows: we first give a series of basic definitions and preliminary results. Then we define the aforementioned parameters, and show how to use them to compute, for a given pattern p, the value σ such that p is unavoidable over alphabets with $m > \sigma$ letters. Finally, we show the dual of the previous result: for alphabets with at most $\sigma - 1$ symbols the pattern p is avoidable.

This topic is a continuation of the study of the avoidability of cubic patterns with permutations from [MMN12b]. In the respective paper for a given pattern $x\pi^i(x)\pi^j(x)$ the authors defined the following four values: $\delta_1 = \inf\{t : t \nmid |i-j|, t \nmid i, t \nmid j\}$, $\delta_2 = \inf\{t : t \mid |i-j|, t \nmid i, t \nmid j\}, \delta_3 = \inf\{t : t \mid i, t \nmid j\}, \delta_4 = \inf\{t : t \nmid i, t \mid j\}$. Further, for $k = \min\{\max\{\delta_1, \delta_2\}, \max\{\delta_1, \delta_3\}, \max\{\delta_1, \delta_4\}\}$, it was shown that $x\pi^i(x)\pi^j(x)$ is unavoidable in Σ_m, for $m \geqslant k$, and avoidable in Σ_m, for $4 \leqslant m < k$. The avoidability of $x\pi^i(x)\pi^j(x)$ in Σ_2 and Σ_3 was separately investigated, and a complete characterisation of the alphabets over which a pattern $x\pi^i(x)\pi^j(x)$ is avoidable was obtained.

The reader is referred to [Lot97; MMN12b] for further details. All computer programs referenced in this paper can be found at `http://media.informatik.uni-kiel.de/zs/patterns.zip`.

4.2 Avoidability of patterns under permutations

In this section we try to identify an upper bound on the size of the alphabets Σ_m in which a pattern $x\pi^i(x)\pi^j(x)\pi^k(x)$, with $i, j, k \geqslant 0$ is unavoidable, when π is substituted by a morphic permutation.

In the pattern $x\pi^i(x)\pi^j(x)\pi^k(x)$, the factors x, $\pi^i(x)$, $\pi^j(x)$, or $\pi^k(x)$ are called x-items in the following. Our analysis is based on the relation between the possible images of the four x-items occurring in a pattern, following the ideas of [MMN12b]. For instance, we want to check whether in a possible image of our pattern, all four x-items can be mapped to a different word, or whether the second and the last x-items can be mapped to the same word, etc.

To achieve this, we define in Table 4.1 the parameters δ_a, with $1 \leqslant a \leqslant 14$. Intuitively, they allow us to define, for a pattern $x\pi^i(x)\pi^j(x)\pi^k(x)$, which are the alphabets Σ_m in which we can model certain (in-)equality relationships between the images of the x-items. For example, in alphabets Σ_m with $m \geqslant \delta_1$ we can assign values to x and π such that the images of every two of $\pi^i(x)$, $\pi^j(x)$, and $\pi^k(x)$ are different (and this property does not hold in alphabets with less than δ_1 letters). Also, in Σ_m with $m \geqslant \delta_2$ we can assign values to x and π such that the images of x and $\pi^i(x)$ are equal to some word, while the images of $\pi^j(x)$ and $\pi^k(x)$ are assigned to two other distinct words (also different between them; again, this property does not hold in smaller alphabets). To simplify, we use a simple digit-representation for any of these cases, defined in the last column of Table 4.1. In this representation of each δ_a, we assign different digits to the x-items that can be mapped to different words in alphabets of size at least δ_a. For example, we use the representation 0123 for the case defined through δ_1 and 0012 for the case defined by δ_2. In general, when considering an δ_a, we assign a 4-digit representation to the pattern $x\pi^i(x)\pi^j(x)\pi^k(x)$ in the following manner: we start with 0, and then put a 0 on all of the remaining three positions corresponding to an x-item $\pi^t(x)$ to such that δ_a divides t. We then put a 1 on the the leftmost empty position. If the x-item on the respective position is $\pi^r(x)$, we put 1 on all empty positions s such that δ_a divides $(r - s)$, and so on.

Please note that the values of the parameters δ_a, with $1 \leqslant a \leqslant 14$ depend on the pattern $x\pi^i(x)\pi^j(x)\pi^k(x)$, and, more precisely, on i, j, k.

Table 4.1. Definition of the values δ_a, with $1 \leqslant a \leqslant 14$.

$\delta_1 = \inf\{t : t \nmid i, t \nmid j, t \nmid k, t \nmid \lvert i-j\rvert, t \nmid \lvert i-k\rvert, t \nmid \lvert j-k\rvert\}$	0123
$\delta_2 = \inf\{t : t \mid i, t \nmid j, t \nmid k, t \nmid \lvert j-k\rvert\}$	0012
$\delta_3 = \inf\{t : t \nmid i, t \mid j, t \nmid k, t \nmid \lvert i-k\rvert\}$	0102
$\delta_4 = \inf\{t : t \nmid i, t \nmid j, t \mid \lvert i-k\rvert\}$	0121
$\delta_5 = \inf\{t : t \nmid i, t \nmid j, t \nmid \lvert i-j\rvert, t \nmid \lvert i-k\rvert, t \mid \lvert j-k\rvert\}$	0122
$\delta_6 = \inf\{t : t \mid i, t \mid j, t \nmid k\}$	0001
$\delta_7 = \inf\{t : t \mid i, t \nmid j, t \mid k\}$	0010
$\delta_8 = \inf\{t : t \nmid i, t \mid j, t \mid k\}$	0100
$\delta_9 = \inf\{t : t \nmid i, t \mid \lvert i-j\rvert, t \mid \lvert i-k\rvert\}$	0111
$\delta_{10} = \inf\{t : t \mid i, t \nmid j, t \mid \lvert j-k\rvert\}$	0011
$\delta_{11} = \inf\{t : t \nmid i, t \mid j, t \mid \lvert i-k\rvert\}$	0101
$\delta_{12} = \inf\{t : t \nmid i, t \mid k, t \mid \lvert i-j\rvert\}$	0110
$\delta_{13} = \inf\{t : t \nmid i, t \nmid k, t \mid \lvert i-j\rvert\}$	0112
$\delta_{14} = \inf\{t : t \nmid i, t \nmid j, t \mid \lvert i-j\rvert\}$	0120

Thus, for different patterns we will have different parameters.

Recall that $\inf \varnothing = \infty$, so the value of some δ_as may be infinite. However, note that the set $\{t : t \nmid i, t \nmid j, t \nmid k, t \nmid |i-j|, t \nmid |i-k|, t \nmid |j-k|\}$ defining δ_1 is always non-empty, and also that $\delta_1 > 3$. Indeed, at least two of i, j, k have the same parity, so δ_1 should not divide 2. Similarly, out of $0, i, j, k$ at least two have the same reminder modulo 3, so δ_1 should also not divide 3. Let $K = \{\delta_1, \delta_2, \ldots, \delta_{14}\}$.

For a pattern $x\pi^i(x)\pi^j(x)\pi^k(x)$, we say that one of the numbers δ_a (and its corresponding representation) models an instance $uf^i(u)f^j(u)f^k(u)$ of the pattern in the case when two of the factors $u, f^i(u), f^j(u), f^k(u)$ are equal if and only if the digits associated to the respective factors in the representation of δ_a are equal. An infinite word w over some alphabet Σ avoids a set $S \subseteq K$ if w contains no instance of the pattern $x\pi^i(x)\pi^j(x)\pi^k(x)$ that is modelled by the parameters of S; note that when we discuss about words avoiding a set of parameters, we implicitly assume that the pattern $x\pi^i(x)\pi^j(x)\pi^k(x)$ is fixed.

Before showing our first results, we need several new notations.

Let w_1 and w_2 be the digit representation of some δ_ℓ, and δ_p respectively, with $\ell, p \geqslant 1$, we say that w_1 is a swapped form of w_2 if there exists a position $i \leqslant 4$ such that $w_1[i] = w_2[i+1]$, and $w_1[i+1] = w_2[i]$, and

$w_1[j] = w_2[j]$ for all $j \notin \{i, i+1\}$. For instance, 0012 and 0102 are swapped forms of each others.

Let δ be the digit representation of $x\pi^i(x)\pi^j(x)\pi^k(x)$. We say that δ has a prefix square if it starts with 00, while the other two digits are 1 and 2; this is the case for $0012 = \delta_2$. Furthermore, a digit representation has a suffix square if it ends with 22 and the two other digits are 0 and 1; this is the case for $0122 = \delta_5$. We say that δ has a gapped square, if it is 0102, where the 0s form the gapped square, or if it is 0121, where the 1s form the gapped square. We say that δ contains a cube if it is 0001 or 0111. We say δ has two squares if it is 0011. Finally, δ contains gapped cubes if it is 0010 or 0100.

Now based on these relations, we define the following collections of sets. The idea behind all these collections is to generate sets of parameters δ_as that cannot be avoided and have a minimal cardinality. No matter what will be added to these sets, they will preserve their unavoidability, while erasing something from them will make them avoidable. To obtain these collections we used a computer program and randomly generated some unavoidable sets of parameters of size five. Using the similarities between the instances modelled by these sets, defined in terms of (gapped) squares and cubes occurring in their digit representation, we developed an algorithm to generate more sets of patterns.

Let $\mathcal{S}_1$ be the collection of sets (each with five elements) that contain δ_1 and:

- one of the δ_as whose representation has a prefix or a suffix square, but no gapped cube. That is: δ_2 or δ_5.
- one of the δ_as that has a gapped square, but does not have two gapped squares. These are δ_3 or δ_4.
- one of the δ_as that contains cubes or two squares: δ_6 or δ_9 or δ_{10}.
- one of the δ_as that contains gapped cubes: δ_7 or δ_8.

For example, one possible set from $\mathcal{S}_1$ is $\{\delta_1, \delta_2, \delta_4, \delta_6, \delta_7\}$. Note that more sets like this one can be constructed using this scheme, and we should consider all of them, but because of lack of the space, we do not list all the examples here.

We also have the restriction that if the representations of the squares and gapped squares of a set from $\mathcal{S}_1$ are not swapped form of each other, then the elements of $\mathcal{S}_1$ representing cubes or gapped cubes should have the same digit on all positions of equal digits from the representations of squares and gapped squares. For example, in the case of 0012 we have that the first and second position contain the same digit and for 0121 we have that the second and the last position contain the same digit, so our gapped cube should be 0010 meaning that the first, second and the last position should contain the same digits.

Let $\mathcal{S}_2$ be the collection of sets (with five elements) that contain δ_1 and:

- one of the δ_as of the set $\{\delta_2, \delta_3, \delta_4\}$, and
- one of the δ_as of the set $\{\delta_6, \delta_7, \delta_9\}$, and
- both δ_as that contain a square in the middle of the word (δ_{12} and δ_{13}).

Moreover, we have the restriction that if we choose δ_2 then δ_7 should be added to the set. For example, one possible set from $\mathcal{S}_2$ is $\{\delta_1, \delta_2, \delta_7, \delta_{12}, \delta_{13}\}$.

Let $\mathcal{S}_3$ be the collection of sets (with five elements) that contain δ_1 and δ_{10} (the only δ_a that has two square factors) as well as:

- one of the δ_as whose representation has a prefix or a suffix square, but no gapped cube. That is: δ_2 or δ_5.
- one of the δ_as whose representation has a gapped square, but does not have two gapped squares. That is: δ_3 or δ_4.
- one of the δ_as whose representation contains gapped cubes: δ_7 or δ_8.

We also have the restriction that if the representation of the squares and gapped squares of a set from $\mathcal{S}_3$ are not swapped form of each other, then its elements representing cubes or gapped cubes should have the same digit on all positions of equal digits from the representations of squares and gapped squares. For example, one possible set from $\mathcal{S}_3$ is $\{\delta_1, \delta_2, \delta_4, \delta_7, \delta_{10}\}$.

Let $\mathcal{S}_4$ be the collection of sets that contain δ_1, δ_2, δ_7, and:

- one of the δ_as whose representations contain cubes (δ_6 or δ_9), or two square (δ_{10}) and

- one of the δ_as for whose representation only the first and last digits are equal, and they are different from all other digits (δ_{14}).

One such set is, for example, $\{\delta_1, \delta_2, \delta_7, \delta_{10}, \delta_{14}\}$.

Let $\mathcal{S}_5$ be the collection of sets that contain δ_1, δ_{12}, δ_{13}, and δ_{14}, as well as one of the δ_as whose representation contains cubes (δ_6 or δ_9). One example is $\{\delta_1, \delta_9, \delta_{12}, \delta_{13}, \delta_{14}\}$.

Let $\mathcal{S}_6$ be the collection of sets that contain δ_1, δ_{10}, δ_{13}, δ_{14}, and

- one of the δ_as whose representation contains a cube (δ_6 or δ_9), and
- one of the δ_as whose representation contains a gapped cube (δ_7 or δ_8), and
- one of the δ_as whose representation contains a gapped square (δ_3 or δ_4).

Here we have the restriction that if we have in a set of $\mathcal{S}_6$ the element δ_7, whose representation has on the first, the second and the last positions the same digit, we should add δ_4 whose second and last digit are the same. Furthermore, the presence of both δ_4 and δ_8 in a set is not permitted. For example, one possible such set is $\{\delta_1, \delta_4, \delta_6, \delta_7, \delta_9, \delta_{10}, \delta_{13}, \delta_{14}\}$.

Let $\mathcal{S}_7$ be the collection of sets that contain δ_1, δ_{12}, δ_{13}, and one of the δ_as of the set $\{\delta_2, \delta_5\}$, and one of the δ_as of the set $\{\delta_3, \delta_4\}$, and one of the δ_as of the set $\{\delta_7, \delta_8\}$. Here we have this restriction that the combination of $\{\delta_2, \delta_4\}$ and $\{\delta_2, \delta_7\}$ is not permitted and if the δ_as of a set from $\mathcal{S}_7$ whose representation contain squares and gapped squares are not swapped form of each other, then its elements representing cubes or gapped cubes should have the same digit on all positions of equal digits from the representations of δ_as that contain squares and gapped squares. For example, one possible such set $\{\delta_1, \delta_3, \delta_5, \delta_8, \delta_{12}, \delta_{13}\}$.

Let $\mathcal{S}_8$ be the collection of sets (with six elements) that contain δ_1 and all elements of the set $\{\delta_3, \delta_5, \delta_7, \delta_{14}\}$, and one of the δ_as of the set $\{\delta_6, \delta_9\}$. One example is $\{\delta_1, \delta_3, \delta_5, \delta_7, \delta_9, \delta_{14}\}$.

Let $\mathcal{S}_9$ be the collection of sets (with seven elements) that contain δ_1 and , δ_{10}:

- one or two elements of the set $\{\delta_2, \delta_5\}$, and

- all elements of the set $\{\delta_7, \delta_{14}\}$ or $\{\delta_8, \delta_{14}\}$ or one or two elements of the set $\{\delta_{12}, \delta_{13}, \delta_{14}\}$,.
- one of the δ_as of the set $\{\delta_3, \delta_4\}$, and
- one of the δ_as of the set $\{\delta_6, \delta_9\}$, and
- one of the δ_as whose representation has two gapped squares δ_{11}.

Here we have the restriction that if two elements of the set $\{\delta_2, \delta_5, \delta_{10}\}$ were selected, one element of the set $\{\delta_{12}, \delta_{13}, \delta_{14}\}$ should also be chosen, and the other way around. Furthermore, if we choose δ_2 or δ_5 or δ_{10} as the δ_as with squares in a set, then δ_3, δ_4, and δ_{14} should be selected as gapped squares, respectively, and in the first two conditions, δ_{10} and (δ_{12} or δ_{13}), and in the last condition, (δ_3 or δ_4) and (δ_7 or δ_8 or δ_{12} or δ_{13}) should be added to the set. The other restriction is that, if we have the numbers δ_7, δ_{14}, then δ_3, and if we have the numbers δ_4, δ_{14}, then (δ_8 or δ_{12}) should be chosen as the gapped squares in a set. In the end, the union of the sets $\{\delta_6, \delta_{10}, \delta_{13}, \delta_{14}\}$, and $\{\delta_{12}, \delta_{13}\}$ is not allowed, and the following sets: $\{\delta_1, \delta_3, \delta_6, \delta_8, \delta_{10}, \delta_{11}, \delta_{14}\}$, $\{\delta_1, \delta_4, \delta_5, \delta_6, \delta_{10}, \delta_{12}, \delta_{14}\}$, $\{\delta_1, \delta_4, \delta_6, \delta_7, \delta_{10}, \delta_{11}, \delta_{14}\}$ are also exceptions that should not be considered. A set fulfilling all the above is $\{\delta_1, \delta_2, \delta_3, \delta_6, \delta_{10}, \delta_{11}, \delta_{13}\}$.

Let $\mathcal{S}_{10}$ be the collection of sets (with eight elements) that contain δ_1 and (only) one of the following sets:

- δ_3 and one of the sets $\{\delta_5, \delta_6, \delta_{10}, \delta_{11}, \delta_{13}, \delta_{14}\}$ or $\{\delta_5, \delta_9, \delta_{10}, \delta_{11}, \delta_{13}, \delta_{14}\}$,
- $\{\delta_2, \delta_4, \delta_{13}\}$ and one of the sets $\{\delta_6, \delta_{10}, \delta_{11}, \delta_{14}\}$ or $\{\delta_9, \delta_{10}, \delta_{11}, \delta_{14}\}$.

For example, one possible such set is $\{\delta_1, \delta_3, \delta_5, \delta_6, \delta_{10}, \delta_{11}, \delta_{13}, \delta_{14}\}$.

While the choice of these classes may seem, in a sense, arbitrary, we tried to group together in the same class sets of parameters, according to the common combinatorial features of the elements defining them. The way we obtained these sets is by computer exploration. The idea behind the definition is to generate unavoidable sets of parameters δ_a which have a minimal cardinality. We basically started with the sets of size 5, and tried to extend them one element at a time in order to obtain unavoidable sets. As such, we ensured that removing some element from them leads to an avoidable set of parameters, while, no matter what element we add

to them preserves their unavoidability. The reason to start with sets of 5 parameters is given in the next lemma.

Lemma 1. Let $K' \subset K$ be any subset of size at most 4 of K. There exists an infinite word w such that w does not contain 4-powers and if w contains an instance of the pattern $x\pi^i(x)\pi^j(x)\pi^k(x)$ then it can not be modelled by any tuples of the set of patterns K'.

Proof. The statement is shown for sets of cardinality 3 in [CMN+18]. We show it for sets of size 4.

To begin with, consider the set $\{\delta_1, \delta_2, \delta_3, \delta_i\}$ for all $4 \leqslant i \leqslant 14$. Since this set is a subset of the sets defined in Lemma 10, Lemma 11, Lemma 3, then the same word that avoids the larger sets can avoid the smaller ones too.

Consider the set $\{\delta_1, \delta_2, \delta_4, \delta_i\}$ for all $5 \leqslant i \leqslant 14$. Since this set is subset of the sets defined in Lemma 14, Lemma 25, then the same word that avoids the larger sets can avoid the smaller ones too.

Consider the set $\{\delta_1, \delta_4, \delta_5, \delta_i\}$ for all $6 \leqslant i \leqslant 14$. Since these sets are subsets of the sets defined in Lemma 10, Lemma 14, Lemma 15, then the same word that avoids the larger sets can avoid the smaller ones too.

Consider the sets $\{\delta_1, \delta_3, \delta_i, \delta_j\}$ for all $10 \leqslant i \leqslant 13$, and $i+1 \leqslant j \leqslant 14$, and the set $\{\delta_1, \delta_4, \delta_i, \delta_j\}$ for all $12 \leqslant i \leqslant 13$, and $i+1 \leqslant j \leqslant 14$. Since these sets are subset of the set defined in Lemma 15, then the same word that avoids the larger set can avoid the smaller ones too.

Consider the sets $\{\delta_1, \delta_3, \delta_i, \delta_j\}$ for all $4 \leqslant i \leqslant 5$, and $i+1 \leqslant j \leqslant 14$. Since this set is subset of the sets defined in Lemma 2, and Lemma 14, then the same word that avoids the larger sets can avoid the smaller ones too.

Consider the sets $\{\delta_1, \delta_3, \delta_i, \delta_j\}$ for all $6 \leqslant i \leqslant 9$, and $i+1 \leqslant j \leqslant 14$, and the set $\{\delta_1, \delta_4, \delta_i, \delta_j\}$ for all $6 \leqslant i \leqslant 11$, and $i+1 \leqslant i \leqslant 14$. Since they are all subsets of the sets defined in Lemma 11, Lemma 15, Lemma **??**, and Lemma 18, then the same word that avoids the larger sets can avoid the smaller ones too.

Consider the sets $\{\delta_1, \delta_2, \delta_i, \delta_j\}$ for all $5 \leqslant i \leqslant 13$, $i+1 \leqslant j \leqslant 14$, and the set $\{\delta_1, \delta_i, \delta_j, \delta_k\}$ for all $5 \leqslant i \leqslant 12$, $i+1 \leqslant j \leqslant 13$, and $j+1 \leqslant k \leqslant 14$. Since these sets are all subset of the set defined in Lemma 16, then the same word that avoids the larger sets can avoid the smaller ones too. As

far as the sets not containing δ_1 are concerned, in Theorem 4.2 we proved that any tuple of the set $\{\delta_2, \delta_3, \delta_4, \delta_5, \delta_6, \delta_7, \delta_8, \delta_9, \delta_{10}, \delta_{11}, \delta_{12}, \delta_{13}, \delta_{14}\}$ can be avoided by the word defined there.

□

The main result of this section is the following theorem.

Theorem 4.1. *Given positive integers i, j, k such that $i \neq j \neq k \neq i$, consider the pattern $p = x\pi^i(x)\pi^j(x)\pi^k(x)$. Let $\sigma = \min\{\max(S) \mid S = S_\ell$ for some $\ell = 1, \ldots, 10\}$. Then $\sigma \geqslant 4$ and p is unavoidable in Σ_m, for all $m > \sigma$.*

Proof. Because $m \geqslant \delta_1$, we have that for every word $u \in \Sigma_m^+$ there exists a morphic permutation f such that every two words of $u, f^i(u), f^j(u), f^k(u)$ are different. Indeed, we take f to be a permutation such that the orbit of $u[1]$ is a cycle of length δ_1, which means that the first letters of $u, f^i(u), f^j(u)$ and $f^k(u)$ are pairwise different. Similarly, the fact that $m \geqslant \delta_2$ (when $\delta_2 \neq \infty$) means that for every $u \in \Sigma_m^+$ there exists a morphic permutation f such that $f^k(u) \neq u = f^i(u) \neq f^j(u) \neq f^k(u)$. In this case, we take f to be a permutation such that $\mathbf{ord}_f(u[1]) = \delta_2$. We can derive similar observations for all the δ_a parameters involved in the definition of σ.

One can check with the aid of a computer, by a backtracking algorithm, that if $m \geqslant \max(S) + 1$, when $S = S_\ell$ for some $\ell = 1, \ldots, 10$, then p is unavoidable in Σ_m. Our computer program tries to construct a word as long as possible by always adding a letter to the current word (obtained by backtracking). This letter is chosen in all possible ways from the letters contained in the word already, or it may also be a new letter, and we just check whether it creates an instance of the pattern as a suffix of the word. Generally, we were not able to check if an arbitrary instance of the pattern is created, due to the complexity of checking all permutations as possible image of π. But, in most of the cases we need to check, we got the result even when we explicitly allowed π to be only a cycle. In the remaining cases, we needed to allow π to act as the identity on a symbol, and as a cycle on the rest of the alphabet. This latter case, which was still easy to check, is the reason why we got that p is only avoidable over alphabets of size at least $\sigma + 1$ and not already over an alphabet of size σ.

For instance, looking at $\mathcal{S}_1$, using a computer program that explores all the possibilities by backtracking we get that if $m \geqslant \max\{\delta_1, \delta_2, \delta_4, \delta_6, \delta_7\}$,

Table 4.2. Longest words avoiding factors modelled by some of the sets in the statement of Theorem 4.1.

$\{\delta_1, \delta_2, \delta_4, \delta_6, \delta_7\}$	0102110022110022110022110000
$\{\delta_1, \delta_2, \delta_7, \delta_{12}, \delta_{13}\}$	0101010202000111222000111222000111222000111112
$\{\delta_1, \delta_2, \delta_4, \delta_7, \delta_{10}\}$	012012012011100
$\{\delta_1, \delta_2, \delta_7, \delta_{10}, \delta_{14}\}$	010201020102010301010102000 1
$\{\delta_1, \delta_3, \delta_5, \delta_8, \delta_{12}, \delta_{13}\}$	000111222000111222000111222000111201201201 2
$\{\delta_1, \delta_2, \delta_3, \delta_6, \delta_{10}, \delta_{11}, \delta_{13}\}$	012012112112111
$\{\delta_1, \delta_2, \delta_3, \delta_6, \delta_{10}, \delta_{11}, \delta_{13}\}$	012012112112111

which is at its turn greater or equal to 4 as $\delta_1 > 3$, the longest word that does not contain an instance of this pattern, even when constraining π to be a cycle, has length 36, and it is 010210210210033001133001133001133000 (adding new letters to this word does not lead to a longer one). On the other hand, we found arbitrarily long words that contain instances of the pattern modelled by $\delta_1, \delta_2, \delta_3, \delta_6, \delta_{10}, \delta_{11}, \delta_{12}$ when we allow π to be replaced only by cycles. However, if we allow π to be more general (i.e., only fix one symbol of the alphabet and be a cycle on the rest), we obtain that there are no infinite words that avoid the pattern in this case. So, over alphabets of size $m \geqslant \max\{\delta_1, \delta_2, \delta_3, \delta_6, \delta_{10}, \delta_{11}, \delta_{12}\} + 1$ the pattern is unavoidable.

All the sets of α_is that are used to define σ are given in the Appendix. For some of them, we show the longest words that do not contain their instances (Table 4.2); these words, as well as words for all the other cases, can be easily found by backtracking.

Note that by the results (Theorem 4 and Theorem 6) of [CMN+18] it also follows that $\sigma + 1 \geqslant 5$. □

4.3 Algorithm to generate avoidable cases

In Lemma 4.1, we proved that given the pattern $x\pi^i(x)\pi^j(x)\pi^k(x)$, for each i, j, and k, we can compute an upper bound on the minimum size of an alphabet over which the pattern is unavoidable. Now to show that

this is the minimum cardinality over which the pattern of size four is unavoidable, we proceed as follows.

Let K_a be the class that contains all nonempty sets of δ_a parameters S' such that S' does not include any set $S = S_\ell$ for some $\ell = 1, \ldots, 10$. In other words, K_a contains all nonempty strict subsets of the sets $S = S_\ell$ for some $\ell = 1, \ldots, 10$ as well as any other sets of parameters that do not include any of the sets $S = S_\ell$ for some $\ell = 1, \ldots, 10$. We already know that all subsets of the sets $S = S_\ell$ for some $\ell = 1, \ldots, 10$ are avoidable. Also, all supersets of the sets $S = S_\ell$ for some $\ell = 1, \ldots, 10$ are unavoidable (as the sets S already are unavoidable), so we try to show that all the other sets of parameters are avoidable. However, K_a has about 1400 sets of patterns, so checking each of them is hard to be done by pen and paper.

Fortunately, there is an observation we can exploit at this point: all subsets of an avoidable set of parameters is avoidable as well. For instance, if the set $\{\delta_1, \delta_2, \delta_5, \delta_6, \delta_8, \delta_{14}\}$ can be avoided by a word $\mathbf{w}$, then the set $\{\delta_1, \delta_2, \delta_5\}$ can also be avoided by $\mathbf{w}$. Thus, we can look for the sets of parameters with maximal cardinality that belong to K_a and are avoidable. Clearly, the entire K is unavoidable. However, $K \backslash \{\delta_1\}$ can be shown to be avoidable. Our approach is implemented in the following algorithmic scheme.

Algorithm to generate avoidable cases

1: Let $n = 10$. Using the sets $\mathcal{S}_i, (1 \leqslant i \leqslant 10)$, generate all sets of δ_as of cardinality n, that have no unavoidable sets of patterns as subset; show that they are avoidable;

2: For all n from 9 down to 4, generate all sets of cardinality n that have no unavoidable sets of patterns as subset; these sets should not be subsets of the avoidable sets of δ_as of cardinality $n + 1$ (to avoid generating repetitive avoidable sets of cases generated in the past step); show that they are avoidable.

The following theorem states which sets of δ_as can be avoided, according to the algorithm above, concluding thus our approach. It is worth noting that the search space was drastically reduced by our approach.

Theorem 4.2. *For each of the following sets there exists an infinite word over*

an alphabet of size at most 5*, such that if this word contains an instance of* $x\pi^i(x)\pi^j(x)\pi^k(x)$ *then this instance can not be modelled by an element of the set.*

$\{\delta_2,\delta_3,\delta_4,\delta_5,\delta_6,\delta_7,\delta_8,\delta_9,\delta_{10},\delta_{11},\delta_{12},\delta_{13},\delta_{14}\}$,
$\{\delta_1,\delta_2,\delta_3,\delta_4,\delta_5,\delta_6,\delta_9,\delta_{10},\delta_{11},\delta_{14}\}$, $\{\delta_1,\delta_2,\delta_3,\delta_4,\delta_5,\delta_6,\delta_9,\delta_{10},\delta_{12},\delta_{14}\}$,
$\{\delta_1,\delta_2,\delta_3,\delta_4,\delta_5,\delta_6,\delta_9,\delta_{10},\delta_{13},\delta_{14}\}$, $\{\delta_1,\delta_2,\delta_3,\delta_4,\delta_5,\delta_6,\delta_9,\delta_{11},\delta_{12},\delta_{14}\}$,
$\{\delta_1,\delta_2,\delta_3,\delta_4,\delta_5,\delta_6,\delta_9,\delta_{11},\delta_{13},\delta_{14}\}$, $\{\delta_1,\delta_2,\delta_3,\delta_4,\delta_5,\delta_7,\delta_8,\delta_{11},\delta_{12},\delta_{14}\}$,
$\{\delta_1,\delta_2,\delta_3,\delta_4,\delta_5,\delta_7,\delta_8,\delta_{11},\delta_{13},\delta_{14}\}$, $\{\delta_1,\delta_2,\delta_3,\delta_4,\delta_5,\delta_{10},\delta_{11},\delta_{12},\delta_{13},\delta_{14}\}$,
$\{\delta_1,\delta_2,\delta_4,\delta_6,\delta_8,\delta_9,\delta_{10},\delta_{12}\}$, $\{\delta_1,\delta_2,\delta_4,\delta_6,\delta_9,\delta_{10},\delta_{11},\delta_{12}\}$,
$\{\delta_1,\delta_2,\delta_4,\delta_6,\delta_8,\delta_9,\delta_{11},\delta_{12},\delta_{14}\}$, $\{\delta_1,\delta_2,\delta_4,\delta_8,\delta_{10},\delta_{11},\delta_{12},\delta_{13}\}$,
$\{\delta_1,\delta_2,\delta_4,\delta_8,\delta_{10},\delta_{12},\delta_{13},\delta_{14}\}$, $\{\delta_1,\delta_2,\delta_4,\delta_8,\delta_{11},\delta_{12},\delta_{13},\delta_{14}\}$,
$\{\delta_1,\delta_2,\delta_4,\delta_6,\delta_8,\delta_9,\delta_{10},\delta_{11},\delta_{13},\delta_{14}\}$, $\{\delta_1,\delta_2,\delta_5,\delta_6,\delta_7,\delta_8,\delta_9,\delta_{10},\delta_{11},\delta_{12},\delta_{13},\delta_{14}\}$,
$\{\delta_1,\delta_3,\delta_4,\delta_6,\delta_7,\delta_8,\delta_9,\delta_{10},\delta_{11},\delta_{12},\delta_{14}\}$, $\{\delta_1,\delta_3,\delta_4,\delta_6,\delta_7,\delta_8,\delta_9,\delta_{10},\delta_{11},\delta_{13}\}$,
$\{\delta_1,\delta_3,\delta_4,\delta_6,\delta_7,\delta_8,\delta_9,\delta_{11},\delta_{13},\delta_{14}\}$, $\{\delta_1,\delta_2,\delta_4,\delta_6,\delta_8,\delta_9,\delta_{10},\delta_{11},\delta_{13}\}$,
$\{\delta_1,\delta_3,\delta_4,\delta_6,\delta_9,\delta_{10},\delta_{11},\delta_{13},\delta_{14}\}$, $\{\delta_1,\delta_3,\delta_4,\delta_7,\delta_8,\delta_{10},\delta_{11},\delta_{12},\delta_{13}\}$,
$\{\delta_1,\delta_3,\delta_4,\delta_7,\delta_8,\delta_{10},\delta_{11},\delta_{13},\delta_{14}\}$, $\{\delta_1,\delta_3,\delta_4,\delta_7,\delta_8,\delta_{11},\delta_{12},\delta_{13},\delta_{14}\}$,
$\{\delta_1,\delta_3,\delta_4,\delta_7,\delta_8,\delta_{10},\delta_{12},\delta_{13},\delta_{14}\}$ $\{\delta_1,\delta_3,\delta_5,\delta_6,\delta_7,\delta_9,\delta_{10},\delta_{12}\}$,
$\{\delta_1,\delta_3,\delta_5,\delta_6,\delta_9,\delta_{10},\delta_{11},\delta_{12}\}$, $\{\delta_1,\delta_3,\delta_5,\delta_7,\delta_{10},\delta_{11},\delta_{12},\delta_{13}\}$,
$\{\delta_1,\delta_3,\delta_5,\delta_7,\delta_{10},\delta_{12},\delta_{13},\delta_{14}\}$, $\{\delta_1,\delta_3,\delta_5,\delta_7,\delta_{11},\delta_{12},\delta_{13},\delta_{14}\}$,
$\{\delta_1,\delta_3,\delta_5,\delta_6,\delta_7,\delta_9,\delta_{10},\delta_{11},\delta_{13},\delta_{14}\}$, $\{\delta_1,\delta_3,\delta_5,\delta_6,\delta_7,\delta_9,\delta_{11},\delta_{12},\delta_{14}\}$
$\{\delta_1,\delta_3,\delta_7,\delta_{10},\delta_{11},\delta_{12},\delta_{13},\delta_{14}\}$, $\{\delta_1,\delta_4,\delta_6,\delta_8,\delta_9,\delta_{10},\delta_{12},\delta_{14}\}$,
$\{\delta_1,\delta_4,\delta_6,\delta_8,\delta_9,\delta_{10},\delta_{13},\delta_{14}\}$, $\{\delta_1,\delta_4,\delta_8,\delta_{10},\delta_{11},\delta_{12},\delta_{13},\delta_{14}\}$

Proof. We only show the statement for the set $T = \{\delta_2,\delta_3,\delta_4,\delta_5,\delta_6,\delta_7,\delta_8,\delta_9,\delta_{10},\delta_{11},\delta_{12},\delta_{13},\delta_{14}\}$. The other cases can be proved in a similar fashion. Words avoiding them are given in Appendix in Lemmas $2-37$.

Let $h_\delta = \delta(h)$, where $\delta : \Sigma_3^* \to \Sigma_5^*$ is the morphism defined by

`0` $\to$ `0123041203410234, 1` $\to$ `0132403124302134, 2` $\to$ `0123402134201324`

We show that if h_δ contains an instance of the pattern $x\pi^i(x)\pi^j(x)\pi^k(x)$ then this instance can not be modelled by any tuple of the set T. Assume, for the sake of contradiction, that h_δ contains a factor of the form $uf^i(u)f^j(u)f^k(u)$ which can be modelled by any of the $\delta_a \in T$ (with f

morphic permutation). The maximum length of a factor of h_δ that does not contain a full image of any letter of the ternary Thue word under δ is 30. Using a computer program we checked that h_δ has no factor of the form $uf^i(u)f^j(u)f^k(u)$ with $|u| < 31$ which can be modelled by any of the $\delta_a \in T$. Further, if u is a word of length $\geqslant 31$, each of the factors $u, f^i(u), f^j(u), f^k(u)$ contains a full image of a letter of h. If all these factors contain only the image of 1 then we get a contradiction, as it would mean that h contains a square (either 11 or a longer square whose image covers $uf^i(u)f^j(u)f^k(u)$, see Appendix). If one of them contains the image of 0 or 2 we proceed as follows. Note that the letters 0 in the image of 0 under δ and the letters 2 in the image of 2 occur repeatedly four times, with 3 symbols between them. So, in one of $u, f^i(u), f^j(u), f^k(u)$, we will have either the image of 0 under δ, or the image of 2 under δ, and, consequently, four occurrences of 0, with 3 other symbols between two consecutive 0s, or, respectively, four occurrences of 2, with three other symbols between two consecutive 2s. Consequently, the four occurrences of 0 or 2 should be aligned to four occurrences of another symbol, when considering the alignment of the factors $u, f^i(u), f^j(u), f^k(u)$. Thus, if at least one of the f^i, f^j, or f^k is not the identity, in $uf^i(u)f^j(u)f^k(u)$, based on the repetition of the letters 0 in the image of 0, we should have one of the following alignments: 0123041203410234 aligned with 0412034102340132 (a contradiction, because this would mean that there is a function mapping 3 to both 2 and 3); 01230412034102340 aligned with 04120341023401234 (a contradiction, because this would mean that there is a function mapping 0 to both 0 and 4); 0123041203410234 aligned with 3240312430213401 (a contradiction, because this would mean that there is a function mapping 1 to both 1 and 4); 0123041203410234012 aligned with 2340213420132401230 (a contradiction, because this would mean that there is a function mapping 0 to both 2 and 3); 012304120341023401324031243 02 aligned with 2340213420132401230412034102302 3 (a contradiction, because this would mean that there is a function mapping 3 to both 1 and 2); 01230412034102340 aligned with 23402134201324013 (a contradiction, because this would mean that there is a function mapping 3 to both 0 and 1); 01230412034102340 aligned with 21342013240123041 (a contradiction, because this would mean that there is a function mapping 4 to both 0 and 4). We can apply the same reasoning for the alignments based on the repe-

tition of the letters 2 in the image of 2, and get again only contradictions. Therefore, no instance of the pattern is contained in h_δ. This concludes our proof.

We were in the case when u is a word of length $\geqslant 31$. So each of the factors $u, f^i(u), f^j(u), f^k(u)$ contains a full image of a letter of **h**. If all these factors contain only the image of 1 then either one of them contains two consecutive images of 1 (and we get a contradiction, as **h** contains no squares) or each factor $u, f^i(u), f^j(u), f^k(u)$ contains exactly one full image of a letter. The image of 1 in u is preceded in $\mathbf{h}_\delta$ by the image of 0 or 2. Let us assume it is preceded by the image of 0 (the other case follows analogously). Then between the images of 1 from u and $f^i(u)$ we must have the image of 2 (or we would get again a square in **h**). Similarly, between the images of 1 from $f^i(u)$ and $f^j(u)$ we must have the image of 0 and between the images of 1 from $f^j(u)$ and $f^k(u)$ we must have the image of 2. Therefore, we get that the square 01210121 occurs in **h**, a contradiction.

□

Theorem 4.3. *Given a pattern $p = x\pi^i(x)\pi^j(x)\pi^k(x)$ we can determine effectively the value σ, such that p is avoidable in Σ_m for $m \leqslant \sigma - 1$ and unavoidable in Σ_m for $m \geqslant \sigma + 1$.*

Proof. By [CMN15; CMN+18], we get that all the unary patterns of size 4 with permutations are avoidable in Σ_m for $m \in \{2, 3, 4\}$. If $i = j$ or $j = k$ then all the instances of the pattern contain squares, so the pattern is avoidable in Σ_m for all $m \geqslant 3$. If $i = k$, then the pattern is avoidable in Σ_m, for all $m \geqslant 3$, according to the results of [MMN12b], where it is shown that $\pi^i(x)\pi^j(x)\pi^i(x)$ is avoidable in such alphabets.

Let us thus assume that $i \neq j$, $i \neq k$, and $j \neq k$ (which is also the setting of Theorem 4.1). We compute the parameters δ_a, with $1 \leqslant a \leqslant 14$, for the given pattern. Then, we consider the sets S_i, with $1 \leqslant i \leqslant 10$, and compute $\sigma = \min\{\max(S) \mid S = S_\ell \text{ for some } \ell = 1, \ldots, 10\}$. By Theorem 4.1 we get that $x\pi^i(x)\pi^j(x)\pi^k(x)$ is unavoidable in Σ_m, for $m \geqslant \sigma + 1$. Let now $S' = S_\ell$ for some $\ell = 1, \ldots, 10$ be a set such that $\max(S') = \sigma$. Assume that there exists $\ell \geqslant 5$ such that $x\pi^i(x)\pi^j(x)\pi^k(x)$ is unavoidable in Σ_ℓ and $\ell < \sigma$. Let A_0 be the set containing all δ_a parameters which are at most ℓ, or, in other words, let A_0 be the maximal subset (with respect to inclusion)

of K such that if $\delta \in A_0$ then $\delta \leqslant \ell$. Clearly, A_0 is either a strict subset of a set $S = S_\ell$ for some $\ell = 1, \ldots, 10$ or A_0 is incomparable to any of the sets of S. It cannot include any set $S'' = S_\ell$ for some $\ell = 1, \ldots, 10$ as then $\max(S'') < \max(S') = \min\{\max(S) \mid S = S_\ell \text{ for some } \ell = 1, \ldots, 10\}$, a contradiction. Thus A_0 is included in one of the sets from the statement of Theorem 4.2. Consequently, there exists an infinite word w over a five letter alphabet that avoids A_0. In fact, w avoids A_0 over all alphabets Σ_m such that the instances of p over Σ_m correspond only to δ_as contained in A_0. This means that w avoids A_0 in Σ_m with $5 \leqslant m \leqslant \ell$. So, p is avoidable in Σ_ℓ, a contradiction.

In conclusion, the pattern $p = x\pi^i(x)\pi^j(x)\pi^k(x)$ is avoidable in Σ_m when $2 \leqslant m < \sigma$. This concludes our proof. □

We get the next corollary, by taking, in the setting of the previous theorem, $\beta = \sigma$, if $x\pi^i(x)\pi^j(x)\pi^k(x)$ is avoidable in Σ_σ, or $\beta = \sigma - 1$, otherwise.

Corollary 4.4. *Given a pattern $p = x\pi^i(x)\pi^j(x)\pi^k(x)$ there exists β a natural number or $+\infty$, such that p is avoidable in Σ_m for $m \in \{2, 3, \ldots, \beta\}$ and unavoidable in $\mathbb{N}\backslash\{2, 3, \ldots, \beta\}$.*

4.4 Conclusions

We have shown how to compute, given a pattern $x\pi^i(x)\pi^j(x)\pi^k(x)$, a rather precise approximation of the size of the alphabets where this pattern is avoidable. More importantly, we show that the sizes of these alphabets form an interval of integers. Our results extend the results of [MMN12b] and [CMN+18]. The method we used is to explore the number theoretic connections between i, j, and k, in relation to the possible orders the permutation π may have. This approach follows the initial ideas of [MMN12b], but requires a much more careful and deeper analysis. Essentially, while the relations between i and j in a cubic pattern $x\pi^i(x)\pi^j(x)$ can be modelled with four parameters only, in the case of $x\pi^i(x)\pi^j(x)\pi^k(x)$ we have 14 such parameters. Exhaustively analysing all the possible relations between these parameters, as it was done in [MMN12b], would take too long, so we devised a less complex way of exploring them. We basically see, on

the one hand, which minimal combinations (in the sense of cardinality) of such parameters lead to the conclusion that the pattern is unavoidable in alphabets of large enough size, while also looking for the maximal combinations of the parameters that lead to the conclusion that the pattern is avoidable in alphabets of small enough size. This approach produced a rather large, but still tractable, case analysis.

In order to extend our results to arbitrarily long unary patterns with permutations, we expect that a valid approach would still be based on defining similar sets of parameters and exploring their combinatorial properties. However, it is to be expected that a direct generalization of the ideas above would lead to a number of parameters which grows exponentially with the length of the pattern, hence to a way too complex exploration in the end.

Chapter 5

On Modelling the Avoidability of Patterns as CSP

5.1 Introduction

In this section we propose a unifying approach that can be used to construct long words satisfying certain avoidability properties. Exploiting the fact that, actually, we want to construct a long word satisfying a series of constraints, we will use a constraint satisfaction problem (CSP) solver to achieve this. As such, it all comes down to a rather similar solution for most avoidability problems: specify the restrictions of the problem we want to solve as constraints in the language of the solver, and then use this standardised, and usually very optimised, software to generate the long words we are looking for. Furthermore, we present here several examples, emphasising that the same ideas and specification methods can be applied for different problems, and that the resulting programs are usually much easier to read and check than many of the imperative programs that were used to show technical lemmas in the literature. Compared, e.g., to the programs we used to analyse the avoidability of patterns under permutations [MMN12a; Res19], this new strategy is also more efficient. After giving a short overview of how MiniZinc works, we will describe our results in the following.

5.1.1 The MiniZinc Language

To begin with, the CSP solver-language we decided to use is MiniZinc. According to the authors of this language, it is designed with the purpose of specifying constraint optimization and decision problems over integers

and real numbers. The programmer specifies a model by formalizing all the constraints, without actually telling the software how to solve the problem (although the model can contain annotations which are used to guide the underlying solver). As such, the actual solution is obtained by a solver invisible to the user. MiniZinc is designed to interface easily to different backend solvers. It does this by transforming an input MiniZinc model and the input data into a FlatZinc model, which consists of variable declaration and constraint definitions as well as a definition of the objective function for optimization problems. Then a general CSP solver is used to decide whether a solution for the specified model exists, and, if yes, to actually find it.

We show how this approach can be used in several well-studied avoidability problems.

Firstly, we approach the avoidability of formulas. Essentially, we are given a set of patterns and an alphabet and we want to construct a long word that does not contain any factor that matches one of the patterns in the set. We do this by specifying a MiniZinc model that defines this problem through a system of constraints, and then solving this system as a CSP. Our model allows for formulas with reversals, and it can be further constrained so that only words that are morphic image of a given standard infinite word are constructed (we used here the binary and ternary Thue-Morse words, but others can be easily used). Secondly, we show how the model can be adapted to check the avoidability of patterns in the abelian sense. Finally, we discuss the avoidability of formulas of patterns under permutations. Here the relatively simple model used in the previous cases needs to be extended with the usage of a non-trivial data file, which is, however, also automatically generated. Such data files are a standard way MiniZinc (and other modelling languages) uses to set the values of certain parameters declared in the model, based on input from the user.

In the following, we give an introduction of how MiniZinc works. For more details, see `http://www.minizinc.org/downloads/doc-latest/minizinc-tute.pdf`.

Information for the reference and use of programs is stored by employing variables or parameters, which are declared and assigned a type, which, at its turn, gives them their value. The fundamental types of parameters are strings (string), integers (int), Booleans (bool) and floating point numbers (float). MiniZinc also supports arrays and sets. As such,

one-and multi-dimensional arrays, are declared as `array[ <index-set1>, ..., <index-setn> ] of <type-inst>`. MiniZinc has a requirement for the array declaration to contain the index set of each dimension. This index set must either be a set variable initialized to an integer range, or an integer range itself (as we use here). Arrays may hold any of the base types. In our models, we also use the `sum` function which provides the arithmetic arrays aggregation function which adds its element.

MiniZinc models may also employ another type of variables, namely decision variables, which are variables in the logical sense. They differ from variables and parameters from standard programming languages in that there is no need for the model to assign them a value. Their values remain unknown, until, during the execution of the MiniZinc model, the solving system decides that it is possible for a certain decision variable to be given a certain value satisfying the model's constraints. MiniZinc makes a careful distinction between parameter and decision variables.

As far as the syntax is concerned, variables are assigned values by *assignment items*. They take the form: `<variable>=<expression>;`. The most important part of a model are the *constraint items*. These take the form: `constraint< Boolean expression>;`. `Forall` and `exists` conditions can be used for arrays of constraints: `forall` ensures that every constraint in an array holds, while `exists` ensures that at least one constraint holds. *Solve items* define precisely the type of solution being sought in our model. In our case we will use only `solve satisfy;` items. In this instance, the problem is a constraint satisfaction problem: we need to discover a value for the decision variables that can satisfy the constraints; the exact value is not important (as it would be in the case of optimisation problems). The last element of the model is the *output statement*. This statement informs MiniZinc what it should print once the model has run and a solution has been discovered. Output items give a good presentation of the model execution's results. They take the form: `output [ <string expression>,..., <string expression> ];`.

The solutions we propose are based on the following standard workflow. Firstly, we generate the *data files*, which encode the input to our model. Secondly, *compile the MiniZinc model together with the data file* into Flatzinc. Thirdly, in an implicit step, run a CSP solver on the Flatzinc file.

The generation of the data files is done by executing a Java program.

Depending on the problem we solve, we need to proceed at this step as follows (the precise semantic of each parameter is explained in the respective sections).

For all problems, we give these parameters as the input arguments of the respective programs sInputGenerator.java: `t/h morphicWordLength morphicWordImagesLengths sigma inputFileName pattern1 [pattern2 ...].`

These will generate .dzn data files, with the name `inputFileName`. The next steps consist in running MiniZinc and the CSP solver. In our case, one can either open the MiniZinc model from the folder minizinc, using a standard IDE, or call the tools from the command line:

- cd minizinc
- mzn2fzn rep.mzn ../permGen/Input.dzn (generates the flatzinc file)
- fzn-gecode rep.fzn (runs gecode on this example)

To set the value of parameters using the generated .dzn data files, on the configuration part of the MiniZinc, and on the Data file menu, one needs to choose the specific data file that you want to use.

We verified that the results generated using Minizinc are correct solutions to the avoidability problems we considered using a checker program ResultChecker.java. This Java program is using a standard backtracking algorithm to check whether the word which is obtained via Minizinc contains instances of the patterns that we wanted to avoid or not. The result was that the generated words did not contain instances, so the Minizinc model produced a correct solution. The input arguments for the java program should be: `solution sigma wordLength pattern1 [pattern2 pattern3 ...]`. Note that we should give the solution generated by Minizinc model between quotation marks as the `solution` argument of the java program. An example of possible arguments for this program are: `"[1, 2, 3, 1, 4, 1, 4, 1, 2, 3]" 5 10 x1x2x2x1r`.

5.1.2 Code

An archive containing the code for all the MiniZinc models we describe in the following, as well as the programs used to generate their input files and the checkers we employed, is available at:

`https://media.informatik.uni-kiel.de/zs/AvoidabilityUsingMinizinc.zip`.

5.2 Checking the Avoidability of Formulas with Reversal

As announced in the introduction, we are interested in constructing words over a given alphabet Σ_ℓ that avoid a set of patterns with reversals, i.e., a formula with reversals. The input of our MiniZinc model is taken from a data file `Input.dzn` in the following form. We are given the size ℓ of the alphabet Σ_ℓ, as the parameter `sigma` which is set in the data file as a positive integer. We are also given the length of the word we want to construct as the parameter `wordLength`. Then we are given the number of patterns in the formula, as the parameter `nrPatterns`, as well as the maximum length of a pattern `maxPatternLength` and the maximum number of variables occurring in one of the patterns in the formula, `maxNrVars`. Finally, we are given the patterns, in an array `patterns`. A pattern with k variables $x_1, \ldots, x_k$ is encoded as a word over $\{1, 2, \ldots, k\} \cup \{-1, -2, \ldots, -k\}$, by replacing all the occurrences of x_i with i and all the occurrences of x_i^r by $-i$ (for all $i \in \Sigma_k$). Connected to the formula, we are also given an array `nrVarsInPattern`, which contains the number of variables occurring in each of the patterns in the formula.

The data file is generated easily by a Java program `InputGenerator.java`, which gets as input `sigma`, `wordLength`, and the formula written as a string of variables. Formally, the most important parameters that we send in the list of arguments of this program are `sigma, wordLength, dataFileName, pattern1, pattern2, pattern3, ....` Alongside we need to send several other parameters `t` or `h`, which specifies that the generated word should be the image of the binary ternary Thue-Morse word, `morphicWordLength` which specifies the length of the prefix of `t/h` that we will map to the generated word, `morphicWordImagesLengths` which specifies the lengths of images of the letters of `t` or `h`. These arguments will be explained in more details in Section 5.5. The formal of the call of this program is given in Listing 1.

```
1 java InputGenerator t/h morphicWordLength morphicWordImagesLengths sigma
      inputFileName pattern
```

```
2
3 Example arguments: t 10 3 2 5 input x1x2x2x1r
```

Listing 5.1. How to Generate Input.dzn

We can now proceed and describe how the MiniZinc model was designed in order to include all the constraints fulfilled by the infinite word we want to construct.

The general idea is the following. We want to construct a word `word` that does not contain any image of the patterns in the formula.

```
1 array [1.. wordLength] of var 1..sigma: word;
```

Listing 5.2. The word we construct

Therefore, we will have a set of constraints that specify that for a given pattern `p` the word we construct does not contain any instance of `p`. Then this set of constraints is used for all the patterns of the formula. Essentially, this is defined as in the Listing 3.

```
1 forall (p in 1.. nrPatterns)(
2   % Here we will define the constraints for the pattern patterns[p].
3 )
```

Listing 5.3. Dealing with formulas

Now, we are ready to define the constraints for the pattern `patterns[p]`. We do not want any instance of this pattern to occur in the string `word` we construct. So, for each position `start` of the word we do not want an instance of `patterns[p]` to occur starting there. To this end, we specify that for each possible assignment of the lengths of the variables occurring in `patterns[p]`, the word `word[start..]` does not start with an instance of `patterns[p]` under a substitution of the variables corresponding to the respective length assignment.

More details are needed here. Firstly, we explain how we generate all the possible length assignments. In principle, it would suffice to have `nrVarsInPattern[p]` stacked loops, assigning to the length of each variable values between 1 and `wordLength`. However, it is not possible to define in MiniZinc such a structure that contains a variable number of stacked loops. Therefore, a new strategy to implement this general idea is needed. For that, we will have for each assignment a `label`, a variable integer. The

variable `label` ranges from 1 to $\texttt{wordLength}^{\texttt{nrVarsInPattern[p]}}$. Now, the length of the variable x_i is encoded in `label` using the formula

$$|x_i| = \left\lceil \frac{\texttt{label}}{\texttt{wordLength}^{\texttt{nrVarsInPattern[p] -i}}} \right\rceil \mod \texttt{wordLength}.$$

If this formula gives us that the length of x_i is 0, then we set the length of x_i to be `wordLength`. This is implemented using the function in Listing 4, where `var` is the variable whose length we want to compute.

```
function int: length(int: var, int: wordLength, int: nrVars, int: label) =
  if (ceil(label / pow(wordLength,nrVars − var) mod wordLength == 0) then
    wordLength
  else
    ceil(label / pow(wordLength,nrVars − var) mod wordLength
  endif;
```

Listing 5.4. Computing the length of variable `var` encoded by `label`

Secondly, for the lengths of the variables occurring in `patterns[p]`, given by the label `i`, we can compute the length of its image under a substitution consistent with those lengths. This is obtained using the MiniZinc code from Listing 5, taking into account that the patterns shorter than `maxPatternLength` are padded with 0s, up to the respective length.

```
sum(k in 1..maxPatternLength where patterns[p, k]!=0)
    (length(abs(patterns[p, k]), wordLength, nrVarsInPattern[p], i)) − 1
```

Listing 5.5. Length of the image of `pattern[p]`

Once the length of the image of `patterns[p]` is computed, we have two cases: this image fits in the suffix of length `wordLength - start + 1` of `word` or not. If not, then there is no image of `patterns[p]` with the respective lengths of the variable occurring at `start` in `word`, so not more constraints are needed. If yes, we need to add more constraints. The idea is that looking at the string occurring at position `start` in `word`, whose length equals the length of the image `patterns[p]`, we are able to identify its factors that correspond to the image of each occurrence of each variable. In our constraints, we ask that there are at least two such factors, that, by length reasons, should correspond to the same variable, and which are not identical. That is, we require that the respective string, occurring at position `start` in `word`, whose length equals the length of the image

`patterns[p]`, cannot be obtained by a consistent assignment of the variables of `patterns[p]`, which also respects the computed lengths for the variables. This is done by the code in Listing 6.

```
exists(varOcc in 1..maxPatternLength where patterns[p,varOcc] != 0)(
  exists(nextOcc in (varOcc + 1)..maxPatternLength where
                  abs(patterns[p,varOcc]) == abs(patterns[p, nextOcc]))(
    let {
      var int: varLength = length(abs(patterns[p,varOcc]), wordLength,nrVarsInPattern[
    p], i),
    } in
    exists(l in 1..varLength)(
      let {
        var int: occInWord = start +
            (if varOcc > 1 then
                (sum(k in 1..(varOcc − 1))
                (length(abs(patterns[p, k]), wordLength, nrVarsInPattern[p], i)))
             else 0 endif),
        var int: nextOccInWord = occInWord +
              sum(k in varOcc..(nextOcc − 1))
                  (length(abs(patterns[p, k]), wordLength, nrVarsInPattern[p], i)),
        var int: posFirst = occInWord + l − 1,
        var int: posSecond =
          (if (patterns[p, varOcc] == patterns[p, nextOcc]) then (nextOccInWord + l −
    1)
           else (nextOccInWord + varLength − l) endif)
      } in
      word[posFirst] != word[posSecond]
    )
  )
)
```

Listing 5.6. The core constraints for `pattern[p]` and label `i`

Basically, in the above Listing we ask for the existence of an occurrence `varOcc` of a variable x in `patterns[p]`, such that x occurs again at least on more time on position `nextOcc` of `patterns[p]`. For the respective variable we denote by `varLength` the length of its image in the assignment defined by the label `i`. Now, we can compute the positions `occInWord` and `nextOccInWord`, respectively, that correspond to the positions where the images of the variable occurring on positions `varOcc` and `nextOcc` of `patterns[p]`, respectively, occur in `word`. The constraints we specify ask for the existence of a position `l` such that the $\texttt{l}^{th}$ symbol of the first image of

the variable (the one starting on `occInWord`), which is found on position `posFirst` in `word`, is different from the `l`th symbol of the second image of the variable (the one starting on `nextOccInWord`), which is found on position `posSecond` in `word`. A particularity of the code is that one has to take into account if the occurrence of the variable x on `nextOcc` is really x or its mirror image x^r when computing `posSecond`.

It is clear that if the respective constraints are satisfied for all choices of `start` and all possible labels, then the word that satisfies our model successfully avoids `patterns[p]` for all choices of `p`.

For example, the word `[1, 2, 3, 1, 4, 1, 4, 1, 2, 3, 1, 4, 1, 2, 3, 1, 2, 3, 1, 4, 1, 4, 1, 2, 3, 1, 2, 3, 1, 4]`, generated by model, does not have instances of the pattern `x1x2x2x1r` over an alphabet of size 4. This word of length 30 was obtained in less than one second on a standard desktop computer. Clearly, this pattern is already avoidable over three letter alphabets, for instance by the Hall word.

5.3 Checking the Avoidability of Patterns in the Abelian Sense

In our second example, we design a model that is satisfied by a word that avoids a certain formula in the Abelian sense. As explained, we first generate an input file for the MiniZinc model. To do so, we use the program `InputGenerator.java` with the input arguments `t/h morphicWordLength morphicWordImageLength sigma inputFileName pattern1 [pattern2 ...]`. For example, a list of such arguments is: `t 5 2 3 3 input xxx`.

The only main difference with respect to the code above occurs when we have computed the length of the image of a pattern `patterns[p]` from the formula, and we are in the case when this image fits in the suffix of `word` that starts on `start`. Now, as above, in the string occurring at position `start` in `word`, whose length equals the length of the image `patterns[p]`, we want to identify two factors corresponding to the image of the same variable, which are not equivalent in the Abelian sense, or in other words do not have the same Parikh vector. Basically, our constraints asks for the existence of a letter (occurring on position `l` in the string that is supposed to be the image of the variable x) whose number of occurrences in the

string corresponding to x starting on position `occInWord` is not equal to its number of occurrences in the string corresponding to x starting on position `nextOccInWord`. To achieve this we count how many times this letter occurs in the each of the two strings that should correspond to x by summing up its occurrences in these images, respectively. Then we ask that these two sums are not equal.

This is entire strategy implemented following the main ideas of the previous section, as shown in Listing 6, with the constraint on line 22 changed as described in Listing 7.

```
1 sum(k in occInWord..(occInWord + length(patterns[p, varOcc], wordLength,
      nrVarsInPattern[p], i) − 1) where word[k] == word[firstPos]) (word[k]) !=
2 sum(k in nextOccInWord..(nextOccInWord + length(patterns[p, nextOcc], wordLength,
      nrVarsInPattern[p], i) − 1) where word[k] == word[firstPos]) (word[k])
```

Listing 5.7. Constraints for abelian avoidability

For example, the word `[1, 2, 1, 3, 1, 2, 1, 4, 1, 2, 1, 3, 2, 1, 2, 4, 2, 1, 2, 3, 2, 4, 3, 1, 3, 2, 1, 2, 4, 1]`, obtained with our model, does not have abelian instances of the pattern `xx` over an alphabet of size 4. Again, the running time on a standard desktop computer was less than a second.

5.4 Checking Avoidability of Patterns under Permutations

In this section we address the avoidability of patterns under permutations. Recall that a pattern under permutations, of length m, is a pattern of the form $\pi_1^{i_1}(x_1)\pi_2^{i_2}(x_2)\cdots\pi_m^{i_m}(x_m)$, where, for $1 \leqslant i \leqslant m$, x_i is a word variables and π_i is a functional variable, to be replaced by morphic or antimorphic permutation of the alphabet of terminals (see [CMN+18]). For example, $x\pi(y)\pi(\pi(x))\sigma(y)$ is a pattern under permutations, where x, y a are word variables and π and σ are functional variables which can be replaced by morphic or antimorphic permutations of the alphabet of terminals. Note that, for simplicity of the exposure, we exclude the case when multiple different functions are applied on the same word variable, i.e, we exclude cases like $\pi(\sigma(x))$.

Now, given a formula consisting of patterns under permutations (i.e., a set of patterns under permutations) and an alphabet Σ_k, we want to design a model that is satisfiable if and only if there exists a word, whose length is also given as input, which avoids the respective formula over Σ_k. Just like before, the model gets as input a data file, which is constructed automatically by a Java program from a formula that is given by the user. Basically, the user is required to run the Java program `InputGenerator.java` with the input `sigma`, the size of the alphabet, `wordLength`, the length of the word to be generated, the name of the data file to be generated (say `Input.dzn`), and the actual formula written as a sequence of patterns, where the word variables are symbols from the set $\{x_1, x_2, x_3, \ldots\}$ and the function variables are symbols from the set $\{p_1, p_2, \ldots\}$. We allow reversed variables, denoted $x_i r$, and antimorphic permutations denoted as $p_i^k(x_j)r$; in the latter, we encode that an antimoprhism p_i, iterated k times, is applied to the word variable x. Let us exemplify this. Consider the pattern $x_1x_2p_1^5(x_2)p_2^3(x_1)$ where p_1 is to be replaced by morphic permutations and p_2 by antimorphic permutations. This will be given as parameter to the Java program as `x1x2p1^5(x2)p2^3(x1)r`.

Now, for a set of patterns the program `InputGenerator.java` generates the data file `Input.dzn`. This file is now more complex. It contains several simple numerical values `sigma, wordLength, nrPatterns` with the same meaning as before. Moreover, we compute and store the integer `nrPermutations` which is simply the total number of permutations over an alphabet of size `sigma`, so `sigma!`. Also, we set `maxNrOccs` as the maximum length of a pattern, i.e. the maximum number of items (all occurrences -not necessarily distinct- of word variables or word variables under morphic or antimorphic permutations) occurring in a pattern, just as we did before. The more complex part is how to encode a pattern. Essentially, a pattern under permutations, of length k, will be encoded as a sequence of k 4-tuples as follows. The i^{th} word variable occurring in the pattern (when read left to right) is mapped to the number i; also, the i^{th} functional variable occurring in the pattern is mapped to the number i. Now, if on position i of the pattern we have $p^k(x)$ (so p is morphic) where x is mapped to i and p to j, we encode this as $(j, k, i, 0)$; if on position i of the pattern we have $p^k(x)r$ (so p is antimorphic) where x is mapped to i and p to j, we encode this as $(j, k, i, 1)$. If on position i of the pattern we have x where

x is mapped to i, we encode this as $(1, 0, i, 0)$ (respectively, $(1, 0, i, 1)$ if we would have had xr on the i^{th} position). It is worth emphasising that we see word variables on which no function variable is applied as word variables on which we apply the identity morphism, so p^0, where p is the first morphism occurring in the pattern. An example is given in the following listing. These tuples are kept in a 3-dimensional array `repetitions`.

```
(1,0,1,0) ,   (1,0,2,0) ,   (1,1,1,0)
```

Listing 5.8. Encoding of the pattern $x_1x_2p_2(x_1)$

The next important part is how to encode the permutations in the file. We will use a 4-dimensional array `permutations`, where `permutations[i][j]` gives us all the possible ways in which the j^{th} permutation acts each time it occurs in the pattern (that is, how p^k is defined, each time some p^k appears in the pattern, where p is the j^{th} permutation). In the following listing we have `permutations[1][1]` for the pattern $x_1x_2p_2(x_2)$. Note that here p_2 is actually the first permutation occurring in the pattern (so, the name the user uses is not important, as it is rehashed to the correct number by our program).

```
(1,2,3) ,   (1,2,3) ,   (1,2,3) ,
(1,2,3) ,   (1,2,3) ,   (1,3,2) ,
(1,2,3) ,   (1,2,3) ,   (2,1,3) ,
(1,2,3) ,   (1,2,3) ,   (2,3,1) ,
(1,2,3) ,   (1,2,3) ,   (3,1,2) ,
(1,2,3) ,   (1,2,3) ,   (3,2,1)
```

Listing 5.9. `permutations[1][1]`

Finally, we add to `Input.dzn` two one-dimensional arrays `nrVarsInPattern` and, respectively, `nrPermsInPattern` which simply encode for each pattern of the formula the number of variables, respectively, functional variables occurring the respective pattern.

Now we can describe how the MiniZinc model works, given the `Input.dzn` data file. Like in the case of a formula with reversals, for each pattern of the formula we check separately whether the generated word contains an image of it. The check is done mainly just like in the case of formulas with reversals: we assign possible lengths to the word variables, and then we check if there exists an assignment of this word variables, as well as one assignment of the function variables that make them fit a factor

of the generated word. The main difference is done in the actual check, which is performed as follows. Firstly, for the pattern with the index `p` in the set of patterns we identify the occurrences of the same variable in the pattern, with permutations applied on it.

```
1  exists(m in 1..nrPermsInPattern[p]) (
2      forall(z in 1..nrPermutations) (
3          exists(varOcc in 1..maxPatternLength where
4              repetitions[p, varOcc, 3] != 0 /\ repetitions[p, varOcc, 1] == m) (
5                  exists (nextOcc in 1..maxPatternLength where
6                          (nextOcc != varOcc /\
7                          (repetitions[p, varOcc, 3] == repetitions[p, nextOcc, 3]) /\
8                          (repetitions[p, nextOcc, 2] == 0 \/
9                           repetitions[p, varOcc, 1] == repetitions[p, nextOcc, 1])))
10             )
11          % actual check will be performed here - see part $2$
12 ))
```

Listing 5.10. Checking an occurrence of a pattern under permutations - part 1

More precisely, we need a certain position of p, where a variable actually occurs. As such we look for the position `varOcc`, with `repetitions[p, varOcc, 3] != 0`. This means that on the certain position we really have a variable (as such or under a permutation) in the current pattern. Such a check is needed because in the case of multiple patterns, when one is shorter it may contain tuples which consist of 0s. See the following example.

```
1  repetitions = array3d(1..numberOfPatterns, 1..maxNumberOfRepetitions, 1..4, [
2    1,0,1,0,      1,0,2,0,      1,1,1,0,
3    1,0,1,0,      1,1,1,0,      0,0,0,0,
4  ]);
```

Listing 5.11. The encoding of the patterns `x1x2p2(x1)` and `x1p1(x1)`

Then, for the variable occurring on `varOcc` in the considered pattern we find its next occurrence on position `nextOcc` (i.e., `repetitions [p, varOcc, 3] == repetitions [p, nextOcc, 3]`)) under the same permutation (so we have `repetitions[p,varOcc,1]= repetitions[p,nextOcc,1]`) or without any permutation applied on it (i.e., `repetitions [p, nextOcc, 2] == 0`).

Then based on a similar method as in the model developed for formulas with reversals, but including now the usage of morphic or antimorphic permutations (lines 10 for the morphic case and, respectively, line 11 for

the antimorphic case, from Listing 11) , we check whether there exists a variable in the pattern occurring multiple times (identified as above), each of its occurrences being under a permutations, whose occurrences are mapped to words which are correctly mapped by the corresponding morphic or antimorphic permutations. The following MiniZinc code is doing this check, corresponding to the lines 7-23 from Listing 6.

```
1  exists(l in 1..varLength)(
2      let {
3          var int: occInWord = start + (if varOcc > 1 then
4              (sum(k in 1..(varOcc − 1))(length(repetitions[p, k, 3], wordLength,
       nrVarsInPattern[p], i))) else 0 endif),
5          var int: nextOccInWord = start + (if nextOcc > 1 then
6              (sum(k in 1..(nextOcc − 1))(length(repetitions[p, k, 3], wordLength,
       nrVarsInPattern[p], i))) else 0 endif),
7          var int: posFirst = occInWord + l − 1,
8          var int: posSecond =
9              (if (repetitions[p, varOcc, 4] == repetitions[p, nextOcc, 4]) then
10                 (nextOccInWord + l − 1)
11                 else (nextOccInWord + varLength − (l − 1) − 1) endif)
12     } in
13     permutations[p, z, varOcc, word[posFirst]] != permutations[p, z, nextOcc, word[
       posSecond]]
14 )
```

Listing 5.12. Checking an occurrence of a pattern under permutations - part 2

For example, the word `[1, 1, 2, 2, 1, 3, 2, 1, 1, 2, 2, 3, 1, 1, 2, 2, 1, 3, 2, 1, 1, 2, 2, 3, 1, 1, 2, 2, 1, 3]` obtained with the model described above does not have instances of the pattern `x1p1(x1)x1` over an alphabet of size 3. Also in this case this word was obtained in less than one second.

5.5 Generating Morphic Words

Generating long words that avoid a certain pattern or formula is usually just a first step in showing avoidability results. In many cases, one is interested in generating such a word that is the morphic image of a word whose structure is well known and studied. To this end, we enhanced our models with additional constraints so that the generated words are also the image of prefixes of well understood infinite words; we only did this

for the Thue-Morse word and for the Hall word (also known as the ternary Thue-Morse word). For simplicity, we restrict our search for morphisms that map the letters of the two aforementioned words to words whose length is given by the user as input.

Let us describe our approach in more details. Firstly, we generate an input file for our model by giving several arguments to a Java program `InputGenerator`. The first argument is either `t` or `h` and specifies which initial infinite word we use: the binary Thue word or, respectively, the Hall word. This word is called `morhpicWord` in the code. Then we give the length that this initial word should have. Then, as a list of integers, the lengths of the images of each letter of the initial word (`t` or `h`) under the morphism that will define the word our model generates. For `t` we need to give these lengths for the letters 0 and 1, while for the Hall word we need to give the lengths for the letters $0, 1, 2$. The rest of the arguments are given as before.

The following example contains a section of an input file which it will be used to generate the final morphic words.

```
1 morphicWordLength = 10;
2 morphicWord = array1d(1..morphicWordLength, [ 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, ]);
3 numberOfMorphicWordImages = 2;
4 morphicWordImagesLengths = array1d(1..numberOfMorphicWordImages, [ 2, 3, ]);
```

Listing 5.13. Specification for the word `t` and the way it should be mapped.

Note that here `morphicWordImagesLengths` is an array that specifies the lengths of the images of the letters of the morphic word under the morphism that will map it to the word avoiding the given patterns. In this example, 2 is the length of the image of letter 0, and 3 is the length of the image of 1 for the word `t`.

Once the input file is constructed, we can proceed and describe how the final word is constructed by the Minizinc model. The main idea is to construct a template-word, which has the desired avoidability properties, whose length is the sum of the lengths of the images of the letters of the morphic word. This word is used as a template for the actual images of the letters of the morphic word. Its first $morphicWordLength[1]$ letters are the image of 0, the next $morphicWordLength[2]$ letters are the image of 1, and so on. This word is called `word`. We also construct `finalWord`, which is supposed to be the word that has the desired avoidability properties and is the morphic image of `morphicWord`, under the morphism defined

with the help of `word`. The two words described above are defined as in the following listing.

```
constraint avoidPatterns(word, wordLength);
constraint avoidPatterns(finalWord, (sum(i in 1..morphicWordLength)
                                      (morphicWordImagesLengths [morphicWord[i] + 1])));
```

Listing 5.14.

The only thing left to do is to check that `finalWord` is the image of `morphicWord`, by the morphism defined using the factors of `word`. This is done in the following listing.

```
constraint
  forall (i in 1..morphicWordLength)(
    let {
      var int: morphCharPos = morphicWord[i] + 1;
      var int: posInWord = (sum (k in 1..(morphCharPos − 1)) (
      morphicWordImagesLengths[k])) + 1,
      var int: posInFinalWord = (sum (k in 1..(i − 1)) (morphicWordImagesLengths[
      morphicWord[k] + 1])) + 1,
    } in
    forall (j in 1..morphicWordImagesLengths[morphCharPos])(
      word[posInWord + j − 1] = finalWord[posInFinalWord + j − 1]
));

```

Listing 5.15. Checking whether `finalWord` is a morphic image

Here `morphCharPos` gives the current letter of `morphicWord`. `posInWord` tells us where the image of this letter starts in `word`. Finally, `posInFinalWord` gives the current position in `finalWord`. We then just have to check whether the factor of length `morphicWordImagesLengths[morphCharPos]` occurring in `word` at position `posInWord` is the same factor as the one of the same length occurring in `finalWord` at `posInFinalWord`. If this last constraint is satisfied, it is clear that `finalWord` is the morphic image of `morphicWord`, and has the avoidabiity properties that we require.

For example, the word `[1, 1, 2, 2, 3, 2, 3, 1, 1, 2, 2, 3]` is generated by MiniZinc as the morphic image of the Thue Morse word of length 5, i.e., `0, 1, 1, 0, 1`, such that the length of the image of 0 is equal to 3, i.e., `112`, and the length of the image of 1 is equal to 2, i.e., `23`.

5.6 Conclusions

In this paper, we proposed a formalization of avoidability problems from string combinatorics as constraint satisfaction problems, and, consequently, using MiniZinc to solve important cases of such problems. This can be seen as part of an ongoing trend that attempts to formalize (or automatize) proofs from combinatorics on words (e.g., [HV17; MSS16; DMR+17]). Our approach seems to provide a relatively simple and uniform way of modelling avoidability problems. In our case, the user just has to specify the input and the constraints, does not have to implement directly the searching for a solution. It can be seen that, in the examples of this paper, all solutions follow the same general pattern, and, in fact, just the correspondence between the items of the patterns is implemented differently: in one case it is equality, in another abelian equivalence, or, finally, it can also be equality under permutations. If a formalization for such an equivalence is defined, our code can be easily adapted to accommodate it.

In general, it is very hard to compare it in terms of efficiency with other existing solutions. Essentially this is because of the very heterogenous landscape in computer-based avoidability testing: each problem is solved in another way, with another programming language, other type of input, etc.. Moreover, due to the high computational complexity of the problem, it is quite usual that the programs constructing long words fulfilling avoidability properties have usually a running time that is measured in tens of hours, so this adds to the hardness of comparing such attempts. In the end, no reliable and relevant benchmarks exist. Because of this, we state as the main contributions of our paper the proposal for this new simpler formalization, rather than making a point related to the efficiency of our new approach (and leave this as potential work for the future).

This is the first attempt, that we are aware of, to solve avoidability problems as CSPs. Similarly, one can try to solve them as SAT problems, using SAT solvers. The efficiency of each such solution depends heavily, of course, on the encoding of the avoidability problem as CSP/SAT formulas: the heavier or lengthier the formula is, the slower will be the solver in solving it. It would be interesting to continue this initial steps towards finding good CSP/SAT encodings for avoidability problems, aiming to

find good encodings for such problems. This final remark also opens the door to a new research direction, that this time could be beneficial for the development of solvers. It seems interesting to us to produce SAT/CSP-benchmarks based on avoidability problems. The nature of these problems seems quite different from the problems modelled usually in such standard benchmarks, so they might lead to formulas of different nature than the one used now, and, ultimately, help identify strengths of weaknesses of CSP/SAT solvers.

Chapter 6

Repetition Avoidance in Products of Factors

6.1 Introduction

A repetition is a part occurring several consecutive time like an in *ananas* and *banana* and they are mostly written as an exponent like $\mathtt{b(an)}^2\mathtt{a}$ for indicating the repetition. The study of repetitive sequences in words is one of the central topics of combinatorics on words, with applications in e.g., pattern matching and stringology in general, or more specific in data compression and bioinformatics (see [Gus97; Lot05]). Even in music one can notice that songs with repetitions seem to be more catchy. Having a second look at *banana* and *ananas*, one may notice that the repetition is followed by a prefix of the repetition, namely a here in both cases as a prefix of an. Taking this into consideration leads to the domain of fractional powers introduced by Dejean in 1972 [Dej72]: whereas a full repetition has an integral exponent ($\mathtt{abbcacaca} = \mathtt{ab}^2\mathtt{(ca)}^3$), the fractional powers, as the word indicates, may be rational numbers - $\mathtt{banana} = \mathtt{b(an)}^{\frac{5}{2}}$, i.e. the complete length of the repetitive part is taken and divided by the length of the repetition. It is worth noticing firstly that the fractional power is not unique, since *banana* can be written as $\mathtt{b(anan)}^{\frac{5}{4}}$ as well, and secondly that the integral powers are only a special case of the fractional powers. Thus, it is not as easy as in the integral part to define that a word w does not contain an δ-power for a rational δ (w is δ-power free), since, for instance abcda has also the exponent $\frac{5}{4}$ but not $\frac{5}{2}$. This leads to the notion of the *critical exponent* of a string (word) w, that is defined as the supremum over all r such that w contains an r-power. Coming back to *banana*, the critical

exponent is $\frac{5}{2}$ since this is greater than $\frac{5}{4}$. Due to the fact that the critical exponent is defined as a supremum, the critical exponent can also be irrational as witnessed by the *Fibonacci word* for which Mignosi and Pirillo [MP92] proved that the critical exponent is the golden ratio $\frac{1+\sqrt{5}}{2}$. As mentioned at the beginning, it is not possible to avoid squares over a two-letter alphabet. This result is one of the oldest in the field of avoidability proven by Thue in 1906 [Thu06]. Like the aforementioned Fibonacci-word the Thue-Morse word is infinitely long and the fixed point of a morphism, i.e. a morphism (a function f with $f(xy) = f(x)f(y)$ for all words x, y) is defined for the letters of the alphabet and then iteratively applied. The Thue-Morse word is defined as the fixed point of the morphism $t(0) = 01$ and $t(1) = 10$, i.e. the infinite continuation of

$$0, 01, 0110, 01101001, 0110100110010110, \ldots$$

Even though the word is not square free, it is overlap-free, i.e. it does not contain a factor of the form $uvuvu$ for a letter u and a potentially empty word v. (For instance neither *banana* nor *ananas* are overlap-free witnessed in *banana* by $u = \texttt{a}$ and $v = \texttt{b}$). Since the Thue-Morse word contains squares but does not contain overlaps (and thus neither cubes), the critical exponent of the Thue-Morse word is 2. The same holds for the Hall word h which is the fixed point of the morphism $h(0) = 012$, $h(1) = 02$, and $h(2) = 1$ and was firstly invented by Thue [Thu06]. The remarkable fact about the Hall word is that it is square-free, but as proven by Thue [Thu06; Hal64] repetitions with exponent close to 2 can be found. Notice that the critical exponent of an infinite single-letter word is infinite. Regarding the critical exponent as introduced above it is worth mentioning that Dalia Krieger [Kri06] gave necessary and sufficient conditions for the critical exponent of an infinite fixed point of a binary k-uniform morphism (the images of all letters have length k) to be bounded, and an explicit formula to compute it in this case.

Summing up the results we mentioned up, factors of (in)finite words were investigated regarding the avoidability of repetitions. Since a factor is a contiguous part of the word, a natural generalisation of this problem is not only to allow considering one factor, but considering i factors for $i \geqslant 2$, and investigating this new word regarding the avoidability and unavoidability of repetitions. Consider for instance the word *ionisation*

which does not have a square. But if we delete the *isat* in the middle and concatenate the prefix and suffix, we get *ionion*, which is a square. If we delete *m*, *t*, and *us* in the word *monotonous*, we get a square in *ono*. An extension to this generalisation is to allow the concatenation of the remaining factors in an arbitrary order: if we take *mi* and *imi* of minimisation and concatenate them to *imimi* we get a $\frac{5}{2}$-power of *im*. Mousavi and Shallit [MS13] investigated the adjusted definition of the critical exponent, i.e. the supremum of $\exp(u)$ over all factors u of conjugates of factors of the word, where only two factors are allowed to be concatenated. Here a word v is a conjugate of a word u if there exists words x, y with $v = xy$ and $u = yx$. The restriction to two factors can also be seen as a circularity in the factors, since the factor *minimi* can be altered to *imimin* by rotation which is the same as building the conjugate by swapping the factors *min* and *imi*.

Before we look deeper into these generalisations, the measure for the avoidability only depending on the alphabet size introduced by Dejean [Dej72] needs to be adjusted. Fixing the alphabet size leads to the question of the smallest critical exponent if all words over this alphabet are taken into consideration, i.e. determining the smallest real number r such that an infinite word exists without any power greater than r. This measure is called the repetition threshold RT and by the Thue-Morse words $\mathrm{RT}(2) = 2$ follows. Dejean [Dej72] proved $\mathrm{RT}(3) = \frac{7}{4}$ and conjectured $\mathrm{RT}(4) = \frac{7}{5}$ and $\mathrm{RT}(k) = \frac{k}{k-1}$ for $k > 4$. Mousavi and Shallit [MS13] extended this notion to RTC - the circular repetition threshold - and they proved $\mathrm{RTC}(2) = 4$ and $\mathrm{RTC}(3) = \frac{13}{4}$. Moreover they established in [MS13] the notion $\mathrm{RT}_i(k)$ describing the repetition threshold over a k-letter alphabet and concatenating $i \in \mathbb{N}$ factors. They proved $\mathrm{RT}_2(k) = \mathrm{RTC}(k)$ for $k = 2, 3$ and conjectured $\mathrm{RT}_2(k) = \mathrm{RTC}(k)$ for $k > 3$.

In this work we act on this suggestions by investigating both $\mathrm{RT}_i(3)$ for $i > 2$ and $\mathrm{RT}_2(k)$ for $k > 3$. While the conjectures by Dejean, Mousavi, and Shallit suggest that each RT (resp. RTC) is computable by one formula depending only on the alphabet size, we will show that the repetition threshold with taking $i \in \mathbb{N}$ factors also depends on the parity of i for 3-letter-alphabets. Moreover, our first result shows that the case $i = 2$ corresponds to the first notion of repetition avoidance in conjugates.

Theorem. *For every $k \geqslant 2$, i.e. every alphabet with at least two letters, the repetition threshold with two factors and the circular repetition threshold are identical,* $\mathrm{RT}_2(k) = \mathrm{RTC}(k)$.

Mousavi and Shallit [MS13] have considered the binary alphabet and obtained that $\mathrm{RT}_i(2) = 2i$ for every $i \geqslant 1$. Our second result considers the ternary alphabet and gives the value of $\mathrm{RT}_i(3)$ for every $i \geqslant 1$. This extends the result of Dejean [Dej72] that $\mathrm{RT}_1(3) = \frac{7}{4}$ and the result of Mousavi and Shallit [MS13] that $\mathrm{RT}_2(3) = \frac{13}{4}$.

Theorem. *Considering a three-letter alphabet the repetition threshold for all $i \geqslant 1$ is given by*

- $\mathrm{RT}_i(3) = \frac{3i}{2} + \frac{1}{4}$ *if $i = 1$ or i is even.*
- $\mathrm{RT}_i(3) = \frac{3i}{2} + \frac{1}{6}$ *if i is odd and $i \geqslant 3$.*

Before we will prove these both theorems, we will give a brief introduction to combinatorics on words as well as the basic definitions for our topic in the next section. The third section will contain our results and in the last section we will give a conclusion.

6.2 Preliminaries

Now we will present the definitions being necessary for this section. The first one will define the notion of repetition. As mentioned before the word `banana` contains twice `an` followed by a prefix of this square, namely `a`. In this case $(\mathtt{anan}, \mathtt{a})$ is a repetition with the period 4 (length of the first component) and the exponent $\frac{5}{4}$ which is calculated as the quotient of the overall-length (repetition plus prefix) divided by the period. Thus `banana` contains also a repetition with period $\frac{3}{2}$, namely $(\mathtt{an}, \mathtt{a})$.

Definition 6.1. A *repetition* in a word w is a pair of words p and e such that pe is a factor of w, p is non-empty, and e is a prefix of pe. If pe is a repetition, then its *period* is $|p|$ and its *exponent* is $\frac{|pe|}{|p|} \in \mathbb{Q}$. A word is *$\delta^+$-free* (resp. *$\delta$-free*) if it contains no repetition with exponent β such that $\beta > \delta$ (resp. $\beta \geqslant \delta$). Moreover it is called (δ^+, n)-free if it does not contain

a repetition with a period at least $n \in \mathbb{N}$ and exponent strictly greater than δ.

Dejean started in [Dej72] the investigation of the smallest such δ such that there exists an infinite δ^+-free word (dependent on of the alphabet size) and denoted this as the repetition threshold.

Definition 6.2. For $k \geqslant 2$ the *repetition threshold* is defined by

$$\mathrm{RT}(k) = \min\{\delta \mid \exists w \in \Sigma_k^\omega : w \text{ is } \delta^+\text{-free}\}.$$

Dejean initiated the study of $\mathrm{RT}(k)$ in 1972 for $k = 2$ and $k = 3$ and her work was followed by a series of papers which determine the exact value of $\mathrm{RT}(k)$ for any $k \geqslant 2$:

- $\mathrm{RT}(2) = 2$ (see [Dej72]);
- $\mathrm{RT}(3) = \frac{7}{4}$ (see [Dej72])
- $\mathrm{RT}(4) = \frac{7}{5}$ (see [Pan84])
- $\mathrm{RT}(k) = \frac{k}{k-1}$, for $k \geqslant 5$ (see [Car07; Mou92; Rao11]).

Since all these results are based on Dejean's work and conjecture, a word fulfilling the property with respect to the size of the alphabet is called a *Dejean word*.

For instance, since the Thue-Morse word (defined over a two-letter-alphabet) is 2^+-free, it is a Dejean word whereas the Fibonacci word (also defined over a two-letter-alphabet) is not since it contains cubes (three consecutive equal factors).

We will now introduce the generalisation considered by Mousavi and Shallit [MS13]. The first notion considers repetitions in conjugates of factors of the infinite word whereas the second one considers repetitions in products (concatenations) of a fixed number of factors of the infinite word.

Definition 6.3. A word $w \in \Sigma^\infty$ is circularly r^+-free if it does not contain a factor pxs such that sp is a repetition of exponent strictly greater than r. The smallest real number r such that w is circularly r^+-free is denoted by $\mathtt{cexp}(w)$. Let $\mathrm{RTC}(k)$ be the minimum of $\mathtt{cexp}(w)$ over every $w \in \Sigma_k^\omega$.

Definition 6.4. For a word $w \in \Sigma^\infty$, let $\mathtt{pexp}_i(w)$ be the smallest real number r such that every product of i factors of w is r^+-free. Let $\mathrm{RT}_i(k)$ be the minimum of $\mathtt{pexp}_i(w)$ over every $w \in \Sigma_k^\omega$.

Note that $\mathrm{RT}_i(k)$ generalises the classical notion of repetition threshold which corresponds to the case $i = 1$, that is, $\mathrm{RT}_1(k) = \mathrm{RT}(k)$ for every $k \geqslant 2$.

6.3 Main results

In this section we will present our proofs for the conjecture $\mathrm{RT}_2(k) = \mathrm{RTC}(k)$ for all $k \geqslant 2$ and tight bounds for $\mathrm{RT}_i(3)$ for all $i \geqslant 1$ proposed by Mousavi and Shallit [MS13].

Theorem 6.5. *For every $k \geqslant 2$, i.e. every alphabet with at least two letters, the repetition threshold with two factors and the circular repetition threshold are identical,* $\mathrm{RT}_2(k) = \mathrm{RTC}(k)$.

Proof. The language of words in Σ_k^* avoiding circular repetitions of exponent at least ℓ (or strictly greater than ℓ) is a factorial language. As it is well-known [FBF+03], if a factorial language is infinite, then it contains a uniformly recurrent word w. By [MS13, Prop. 14], $\mathtt{pexp}_2(w) = \mathtt{cexp}(w)$. This implies that $\mathrm{RT}_2(k) = \mathrm{RTC}(k)$. □

To obtain the two equalities of our second main result, namely the bounds for $\mathrm{RT}_i(3)$ for all $i \geqslant 1$, we show firstly the two lower bounds and then the two upper bounds.

Lemma 6.6. *For every even i,* $\mathrm{RT}_i(3) \geqslant \frac{3i}{2} + \frac{1}{4}$ *holds.*

Proof. Mousavi and Shallit [MS13] have proven that $\mathrm{RT}_2(3) = \frac{13}{4}$, which settles the case $i = 2$. We have double checked their computation of the lower bound $\mathrm{RT}_2(3) \geqslant \frac{13}{4}$. Suppose that i is a fixed even integer and that w_3 is an infinite ternary word. The lower bound for $i = 2$ implies that there exists two factors u and v such that $uv = t^e$ with $e \geqslant \frac{13}{4}$. Thus, the prefix t^3 of uv is also a product of two factors of w_3. So we can form the i-terms product $(t^3)^{i/2-1}uv$ which is a repetition of the form t^x with exponent

$$x = 3\left(\tfrac{i}{2} - 1\right) + e \geqslant 3\left(\tfrac{i}{2} - 1\right) + \tfrac{13}{4} = \tfrac{3i}{2} + \tfrac{1}{4}.$$

This is the desired lower bound. □

Lemma 6.7. *For every odd $i \geqslant 3$, $\mathrm{RT}_i(3) \geqslant \frac{3i}{2} + \frac{1}{6}$ holds.*

Proof. Suppose that $i \geqslant 3$ is a fixed odd integer, that is, $i = 2j + 1$. Suppose that w_3 is a recurrent ternary word such that the product of i factors of w_3 is never a repetition of exponent at least $\frac{3i}{2} + \frac{1}{6} = 3j + \frac{5}{3}$. First, w_3 is square-free since otherwise there would exist an i-terms product of exponent $2i$. Also, w_3 does not contain two factors u and v with

$$uv = t^3 \quad \text{and} \quad u = t^e \text{ with } e \geqslant \tfrac{5}{3}$$

since this would produce the i-terms product $(uv)^j u$ which is a repetition of the form t^x with exponent $x = 3j + e \geqslant 3j + \frac{5}{3}$. Consequently, if 0, 1, and 2 are distinct letters, then w_3 does not contain both $u = 01201$ and $v = 2012$ and w_3 does not contain both $u = 0121012$ and $v = 10121$, as well as any image of these u, v under a permutation of the alphabet. A computer check shows that no infinite ternary square-free word satisfies this property. This proves the desired lower bound. □

Remark 6.8. The word `012021201021012021201020121012021201021012021201` is the longest word fulfilling the conditions in the proof of Lemma 6.7 which has length 48 resp. any images of it under a permutation of the alphabet.

We will now give the proofs for the two upper bounds. For that morphisms are necessary and we firstly describe how we found the morphism used in the proofs. Since the small morphisms are much easier to find, we only explain the technique for the big ones. For increasing k, we get a k-uniform morphism m by looking for a ternary square-free word w of length $4k$ (with the suitable $\mathrm{pexp}_i(w)$ properties) that corresponds to $m(0123)$, using backtracking. To speed things up

- ▷ we force $m(0) > m(1) > m(2) > m(3)$.
- ▷ we use early tests: if we have a candidate for $m(01)$,
  - ▷ we also test $m(10)$; if we have a candidate for $m(012)$,
  - ▷ we test all every word $m(abca)$ such that $\{a, b, c\} = \{0, 1, 2\}$

The general idea of the method is that large occurrences of the forbidden structures are ruled out thanks to an argument about the exponent of the repetitions induced by these structures. Then the small occurrences are ruled out by an exhaustive inspection of the factors of the word of some finite length.

Lemma 6.9. *For all even i, $\mathrm{RT}_i(3) \leqslant \frac{3i}{2} + \frac{1}{4}$ holds.*

Proof. Let i be any even natural number at least 2. To prove this upper bound, it is sufficient to construct a ternary word w satisfying $\mathtt{pexp}_i(w) \leqslant \frac{3i}{2} + \frac{1}{4}$. The ternary morphic word used in [MS13] to obtain $\mathrm{RT}_2(3) \leqslant \frac{13}{4}$ seems to satisfy the property. However, it is easier for us to consider another construction. Let us show that the image of every $(\frac{7}{5})^+$-free word u over Σ_4 by the following 45-uniform morphism satisfies $\mathtt{pexp}_i(u) \leqslant \frac{3i}{2} + \frac{1}{4}$. Define the morphism by

$$\begin{aligned}
\mathtt{0} &\mapsto \mathtt{010201210212021012102010212012101202101210212}\\
\mathtt{1} &\mapsto \mathtt{010201210212012101202101210201021202101210212}\\
\mathtt{2} &\mapsto \mathtt{010201210120212012102120210121021201210120212}\\
\mathtt{3} &\mapsto \mathtt{010201210120210121021201210120212012102010212}
\end{aligned}$$

Recall that a word is (α^+, n)-free if it does not contain a repetition with period at least n and exponent strictly greater than α. First, we check that such ternary images are $\left(\frac{202}{135}^+, 36\right)$-free using the method in [Och06]. By [Och06, Lemma 2.1], it is sufficient to restrict this check to the image of every $(\frac{7}{5})^+$-free word over Σ_4 of length smaller than $\frac{2\cdot\frac{202}{135}}{\frac{202}{135}-\frac{7}{5}} < 32$. Since $\frac{202}{135} < \frac{3}{2}$ holds, the period of every repetition formed by i pieces and with exponent at least $\frac{3i}{2}$ has to be at most 35. Then we check exhaustively by computer that the ternary images do not contain two factors u and v with the properties

- $uv = t^e$,
- $e > 3$, and
- $9 \leqslant |t| \leqslant 35$.

Thus, the period of every repetition formed of i pieces and with exponent strictly greater than $\frac{3i}{2}$ must be at most 8. So we only need to check that $\mathtt{pexp}_i \leqslant \frac{3i}{2} + \frac{1}{4}$ for i-terms products that are repetitions of period at most 8. But even for a bounded period, i can still be arbitrarily large, a priori. For every factor t of length at most 8, we define $\mathtt{pexp}_{i,t}$ as the length of a largest factor of t^ω that is a i-terms product divided by $|t|$. We actually consider conjugacy classes, since if t' is a conjugate of t, then $\mathtt{pexp}_{i,t'} = \mathtt{pexp}_{i,t}$. Let t be such a factor. If, for some even j, we have $\mathtt{pexp}_{j+2,t} = \mathtt{pexp}_{j,t} + 3$, then it means that by appending a 2-terms product to a j-terms product that corresponds to a maximum factor of t^ω, that can only add a cube of period $|t|$. This implies that for every k, $\mathtt{pexp}_{j+2k,t} = \mathtt{pexp}_{j,t} + 3k$.

We have checked by computer that for every conjugacy class of words t of length at most 8, there exists a (small) even j such that $\mathtt{pexp}_{j+2,t} = \mathtt{pexp}_{j,t} + 3$. Thus we have $\mathtt{pexp}_i \leqslant \frac{3i}{2} + \frac{1}{4}$ in all cases. □

Lemma 6.10. *For all odd $i \geqslant 3$, $\mathrm{RT}_i(3) \leqslant \frac{3i}{2} + \frac{1}{6}$ holds.*

Proof. Let us show that the image of every $\frac{7}{5}^+$-free word over Σ_4 by the following 514-uniform morphism satisfies $\mathtt{pexp}_i \leqslant \frac{3i}{2} + \frac{1}{6}$ for every odd

$i \geqslant 3$. Consider the morphism given by

```
0 ↦  01020120210120102120210201210120102012021020121021201020121012
     02102012102120210120102012102120102012021020121012010212021020
     12102120102012021012010212021020121021202101201020121021201020
     12101202102012102120210120102120210201210120102012021020121012
     01021202102012102120102012101202102012102120102012021012010212
     02102012101201020120210201210212021012010201210120210201210212
     01020120210201210120102120210201210212010201210120210201210212
     02101201021202102012101201020120210201210120102120210201210212
     021012010201210212
1 ↦  01020120210120102120210201210120102012021020121021201020121012
     02102012102120102012021020121012010212021020121021201020120210
     12010212021020121021202101201020121021201020121012021020121021
     20210120102120210201210120102012021020121021201020121012021020
     12101201021202102012102120210120102012102120102012021012010212
     02102012101201020120210201210120102120210201210212010201210120
     21020121021202101201021202102012101201020120210201210212021012
     01020121012021020121021201020120210201210120102120210201210212
     021012010201210212
2 ↦  01020120210120102120210201210120102012021020121021201020121012
     02102012101201021202102012102120102012021012010212021020121021
     20210120102012102120102012101202102012102120210120102120210201
     21012010201202102012101201021202102012102120102012101202102012
     10212010201202101201021202102012102120210120102012102120102012
     02102012101201021202102012102120102012021012010212021020121012
     01020120210201210212021012010201210212010201210120210201210212
     02101201021202102012101201020120210201210120102120210201210212
     021012010201210212
3 ↦  01020120210120102120210201210120102012021020121021201020121012
     02102012101201021202102012102120102012021012010212021020121021
     20210120102012101202102012102120102012021020121012010212021020
     12102120102012101202102012102120210120102012102120102012021012
     01021202102012101201020120210201210212010201210120210201210212
     01020120210201210120102120210201210212021012010201210212010201
     20210120102120210201210212010201210120210201210212021012010212
     02102012101201020120210201210120102120210201210212021012010201
     210120210201210212
```

First, we check that such ternary images are $\left(\frac{3}{2}^+, 45\right)$-free using the method in [Och06]. By [Och06, Lemma 2.1], it is sufficient to check this property for the image of every $\frac{7}{5}^+$-free word over Σ_4 of length smaller than $\frac{2\cdot\frac{3}{2}}{\frac{3}{2}-\frac{7}{5}} = 30$. Thus, the period of every repetition formed of i pieces and with exponent strictly greater than $\frac{3i}{2}$ must be at most 44. Using the same argument as in the proof of Lemma 6.9, we have checked by computer that for every conjugacy class of words t of length at most 44, there exists a (small) odd j such that $\mathtt{pexp}_{j+2,t} = \mathtt{pexp}_{j,t} + 3$. Thus we have $\mathtt{pexp}_i \leqslant \frac{3i}{2} + \frac{1}{6}$ in all cases. □

By the last four lemmas we proved in the even as well as in the odd case the tight bounds for the repetition threshold RT_i taking i factors into account for the three letter alphabet.

Theorem 6.11. *Considering a three-letter alphabet the repetition threshold for all $i \geqslant 1$ is given by*

- $\mathrm{RT}_i(3) = \frac{3i}{2} + \frac{1}{4}$ *if $i = 1$ or i is even.*
- $\mathrm{RT}_i(3) = \frac{3i}{2} + \frac{1}{6}$ *if i is odd and $i \geqslant 3$.*

6.4 Concluding remarks

In this work generalisation of the repetition threshold were investigated. We think that RT_i is a sound generalization of the classical repetition threshold RT, which corresponds to the case $i = 1$. With our first result we prove that the generalisation for the circular repetition threshold (RTC), is only a special case of RT_i, namely for $i = 2$.

The next step would be to consider the 4-letter alphabet. Obviously, $\mathrm{RT}_{i+1}(k) \geqslant \mathrm{RT}_i(k) + 1$ for every $i \geqslant 1$ and $k \geqslant 2$. Mousavi and Shallit [MS13] verified that $\mathrm{RT}_2(4) \geqslant \frac{5}{2}$, so that $\mathrm{RT}_i(4) \geqslant i + \frac{1}{2}$ for every $i \geqslant 2$. We conjecture that this is best possible, i.e., that $\mathrm{RT}_i(4) = i + \frac{1}{2}$ for every $i \geqslant 2$. However, a proof of an upper bound of the form $\mathrm{RT}_i(4) \leqslant i + c$ cannot be similar to the proof of the upper bounds of Theorem 6.11. The multiplicative factor of i, which drops from $\frac{3}{2}$ when $k = 3$ to 1 when

$k = 4$, forbids that the constructed word is the morphic image of any (unspecified) Dejean word over a given alphabet. Proving some of the conjectured values of RT_i will lead to stronger versions of the classical repetition threshold: every witness of $\mathrm{RT}_i(k) = \mathrm{RT}(k) + i - 1$ is a Dejean word with severe restrictions on the types of repetitions that are allowed to appear.

Appendix A

Unary patterns under permutations

Here is the list of all unavoidable sets of δ_is:

$\{\delta_1,\delta_2,\delta_3,\delta_6,\delta_7\}$, $\{\delta_1,\delta_2,\delta_3,\delta_6,\delta_8\}$, $\{\delta_1,\delta_2,\delta_3,\delta_7,\delta_9\}$, $\{\delta_1,\delta_2,\delta_3,\delta_7,\delta_{10}\}$,
$\{\delta_1,\delta_2,\delta_3,\delta_8,\delta_9\}$, $\{\delta_1,\delta_2,\delta_3,\delta_8,\delta_{10}\}$, $\{\delta_1,\delta_2,\delta_4,\delta_6,\delta_7\}$, $\{\delta_1,\delta_2,\delta_4,\delta_7,\delta_9\}$,
$\{\delta_1,\delta_2,\delta_4,\delta_7,\delta_{10}\}$, $\{\delta_1,\delta_3,\delta_5,\delta_6,\delta_8\}$,$\{\delta_1,\delta_3,\delta_5,\delta_8,\delta_9\}$, $\{\delta_1,\delta_3,\delta_5,\delta_8,\delta_{10}\}$,
$\{\delta_1,\delta_3,\delta_6,\delta_{12},\delta_{13}\}$,$\{\delta_1,\delta_3,\delta_9,\delta_{12},\delta_{13}\}$,$\{\delta_1,\delta_4,\delta_5,\delta_6,\delta_7\}$,$\{\delta_1,\delta_4,\delta_5,\delta_6,\delta_8\}$,
$\{\delta_1,\delta_4,\delta_5,\delta_7,\delta_9\}$, $\{\delta_1,\delta_4,\delta_5,\delta_7,\delta_{10}\}$,$\{\delta_1,\delta_4,\delta_5,\delta_8,\delta_9\}$, $\{\delta_1,\delta_4,\delta_5,\delta_8,\delta_{10}\}$,
$\{\delta_1,\delta_4,\delta_6,\delta_{12},\delta_{13}\}$, $\{\delta_1,\delta_4,\delta_9,\delta_{12},\delta_{13}\}$, $\{\delta_1,\delta_2,\delta_3,\delta_8,\delta_{12},\delta_{13}\}$,
$\{\delta_1,\delta_3,\delta_5,\delta_8,\delta_{12},\delta_{13}\}$, $\{\delta_1,\delta_4,\delta_5,\delta_7,\delta_{12},\delta_{13}\}$, $\{\delta_1,\delta_4,\delta_5,\delta_8,\delta_{12},\delta_{13}\}$,
$\{\delta_1,\delta_2,\delta_3,\delta_6,\delta_{10},\delta_{11},\delta_{12}\}$, $\{\delta_1,\delta_2,\delta_3,\delta_6,\delta_{10},\delta_{11},\delta_{13}\}$, $\{\delta_1,\delta_2,\delta_3,\delta_9,\delta_{10},\delta_{11},\delta_{12}\}$,
$\{\delta_1,\delta_2,\delta_3,\delta_9,\delta_{10},\delta_{11},\delta_{13}\}$, $\{\delta_1,\delta_4,\delta_5,\delta_6,\delta_{10},\delta_{11},\delta_{12}\}$, $\{\delta_1,\delta_4,\delta_5,\delta_6,\delta_{10},\delta_{11},\delta_{13}\}$,
$\{\delta_1,\delta_4,\delta_5,\delta_9,\delta_{10},\delta_{11},\delta_{12}\}$,$\{\delta_1,\delta_4,\delta_5,\delta_9,\delta_{10},\delta_{11},\delta_{13}\}$,$\{\delta_1,\delta_3,\delta_6,\delta_8,\delta_9,\delta_{10},\delta_{13},\delta_{14}\}$,
$\{\delta_1,\delta_4,\delta_6,\delta_7,\delta_9,\delta_{10},\delta_{13},\delta_{14}\}$.

Table A.1. All unavoidable sets of δ_is

We now consider the remaining cases, in a series of Lemmas.

Let $\delta : \Sigma_3^* \to \Sigma_5^*$ be the morphism that is defined by
$0 \to 0010020010020010$, $1 \to 0202242232322122$, $2 \to 0202212202212202$

Lemma 2. Consider the infinite word:
$\mathbf{h}_\delta = \delta(\mathbf{h}) = 0010020010020010020224223232212202022122022122020 01\ldots$
If $\mathbf{h}_\delta$ contains an instance of the pattern $x\pi^i(x)\pi^j(x)\pi^k(x)$ then this instance can not be modelled by any tuple of the set $\{\delta_1,\delta_2,\delta_3,\delta_4,\delta_5,\delta_6,\delta_9,\delta_{10},\delta_{12},\delta_{14}\}$.

Let $\delta : \Sigma_3^* \to \Sigma_5^*$ be the morphism that is defined by

$0 \to 000112220001122200011222$, $1 \to 000113330001133300011$,
$2 \to 444000114440001122200111$

Lemma 3. Consider the infinite word:
$\mathbf{h}_\delta = \delta(\mathbf{h}) = 000112220001122200011222000113330001133300011444000 1\ldots$
If $\mathbf{h}_\delta$ contains an instance of the pattern $x\pi^i(x)\pi^j(x)\pi^k(x)$ then this instance can not be modelled by any tuple of the set $\{\delta_1, \delta_2, \delta_3, \delta_4, \delta_5, \delta_7, \delta_8, \delta_{11}, \delta_{12}, \delta_{14}\}$.

Let $\delta : \Sigma_3^* \to \Sigma_5^*$ be the morphism that is defined by
$0 \to 00122011200311022100211300122011200311022100211300 1$, $1 \to 00122011200311022100211300122011200311022100211300 1$,
$2 \to 00122011200311022100211400122011200311022100211400 1$

Lemma 4. Consider the infinite word:
$\mathbf{h}_\delta = \delta(\mathbf{h}) = 00122011200311022100211300122011200311022100211300 1\ldots$
If $\mathbf{h}_\delta$ contains an instance of the pattern $x\pi^i(x)\pi^j(x)\pi^k(x)$ then this instance can not be modelled by any tuple of the set $\{\delta_1, \delta_3, \delta_4, \delta_6, \delta_7, \delta_8, \delta_9, \delta_{10}, \delta_{11}, \delta_{12}, \delta_{14}\}$.

Let $\delta : \Sigma_3^* \to \Sigma_5^*$ be the morphism that is defined by
$0 \to 0102202112133132232002033034434004022024424114122120020110133$
$1300302202332344342242002044041141 0$,

$1 \to 0102202112133132232002033034434004022024424114122120020110133$
$1300302202332344342242002044041141 0$,

$2 \to 0102202112133132232002033034434004022024424114122120020110133$
$130030220233234434224200204404$

Lemma 5. Consider the infinite word:
$\mathbf{h}_\delta = \delta(\mathbf{h}) = 010220211213313223200203303443400402202442411412212\ldots$
If $\mathbf{h}_\delta$ contains an instance of the pattern $x\pi^i(x)\pi^j(x)\pi^k(x)$ then this instance can not be modelled by any tuple of the set $\{\delta_1, \delta_2, \delta_4, \delta_6, \delta_8, \delta_9, \delta_{10}, \delta_{11}, \delta_{13}, \delta_{14}\}$.

Let $\delta : \Sigma_3^* \to \Sigma_5^*$ be the morphism that is defined by
$0 \to 0100131130330200232234334244202204004144101102002122131132332$

022030034334044020024224144121120 22,

1 → 0100131130330200232234334244202204004144101102002122131132332
022030034334044020024224144121120 22,

2 → 0100131130330200232234334244202204004144101102002122131132332
02203003433404402002422 4

Lemma 6. Consider the infinite word:
$\mathbf{h}_\delta = \delta(\mathbf{h}) =$ 0100131130330200232234334244202204004144101102002 12...
If $\mathbf{h}_\delta$ contains an instance of the pattern $x\pi^i(x)\pi^j(x)\pi^k(x)$ then this instance can not be modelled by any tuple of the set $\{\delta_1, \delta_3, \delta_5, \delta_6, \delta_7, \delta_9, \delta_{10}, \delta_{11}, \delta_{13}, \delta_{14}\}$.

Let $\delta : \Sigma_3^* \to \Sigma_5^*$ be the morphism that is defined by
0 → 0102211003011210020331100201121002011312200102211003011210020
3311002011210020113122001022110030112100203311002011210020113122001
022110030112100203311002011210020113144 0,

1 → 0102211003011210020331100201121002011312200102211003011210020
3311002011210020113122001022110030112100203311002011210020113122001
022110030112100203311002011210020113144 0,

2 → 0102211003011210020331100201121002011312200102211003011210020
3311002011210020113122001022110030112100203311002011210020113122001
022110030112100203311002011210020113300201121002011220030112100201
122003011210020112200301121002011220211002011210020113122 0

Lemma 7. Consider the infinite word:
$\mathbf{h}_\delta = \delta(\mathbf{h}) =$ 0102211003011210020331100201121002011312200102211 0 0...
If $\mathbf{h}_\delta$ contains an instance of the pattern $x\pi^i(x)\pi^j(x)\pi^k(x)$ then this instance can not be modelled by any tuple of the set $\{\delta_1, \delta_2, \delta_4, \delta_6, \delta_8, \delta_9, \delta_{11}, \delta_{12}, \delta_{14}\}$.

Let $\delta : \Sigma_3^* \to \Sigma_4^*$ be the morphism that is defined by
0 → 0001222100012220111022201110222011102220111200012221000122201 1

```
10222011102220111022201112000122210001222011102220111022201110222011102220
11120002111022201110222011102220111022210001222011102220111022201110222011
1022201112,

1 → 00012221000122201110222011102220111022201112000122210001222011
10222011102220111022201112000122210001222011102220111022201110222011102220
1113,

2 → 00012221000122201110222011102220111022201112000122210001222011
10222011102220111022201112000122210001222011102220111022201110222011102220
11120001222100012220111022201110222011102220111
```

Lemma 8. Consider the infinite word:

$\mathbf{h}_\delta = \delta(\mathbf{h}) =$ `000122210001222011102220111022201110222011120001222`$\ldots$

If $\mathbf{h}_\delta$ contains an instance of the pattern $x\pi^i(x)\pi^j(x)\pi^k(x)$ then this instance can not be modelled by any tuple of the set $\{\delta_1, \delta_3, \delta_4, \delta_7, \delta_8, \delta_{10}, \delta_{12}, \delta_{13}, \delta_{14}\}$.

Let $\delta : \Sigma_3^* \to \Sigma_5^*$ be the morphism that is defined by

```
0 →  01120221011202231330011202210112022313300112022101120223133
01120221011202233001211002122001130331011202210112200313300114044
10112022101122003133001140441011202210112200313300114044101120221
01122003133001142440011404 41,

1 → 01120221011220031330011404410112022101122003133001140441011202
21011220031330011404410112022101122003233001130331011202210112200 3
13300114044101120221011220031330011404410112022101122003133001140 4
41011202210112200313300114244001140441,

2 → 01120221011220031330011404410112022101122003133001140441011202
21011220031330011404410112022101122003233001130331011202210112200
31330011404410112022101122003133001140441011202210112200313300114
04410112022101122003133001142440011404 41
```

Lemma 9. Consider the infinite word:

$\mathbf{h}_\delta = \delta(\mathbf{h}) =$ 011202210112022313300112022101120223133001120221011...
If $\mathbf{h}_\delta$ contains an instance of the pattern $x\pi^i(x)\pi^j(x)\pi^k(x)$ then this instance can not be modelled by any tuple of the set $\{\delta_1, \delta_3, \delta_5, \delta_6, \delta_7, \delta_9, \delta_{11}, \delta_{12}, \delta_{14}\}$.

Let $\delta : \Sigma_3^* \to \Sigma_5^*$ be the morphism that is defined by
0 → 00100100200100100200100 1003, 1 → 00100100200100100200100 1003,
2 → 00100100200100100200100 1004

Lemma 10. Consider the infinite word:
$\mathbf{h}_\delta = \delta(\mathbf{h}) =$ 00100100200100100200100100300100100200100100200100 1...
If $\mathbf{h}_\delta$ contains an instance of the pattern $x\pi^i(x)\pi^j(x)\pi^k(x)$ then this instance can not be modelled by any tuple of the set $\{\delta_1, \delta_2, \delta_3, \delta_4, \delta_5, \delta_6, \delta_9, \delta_{10}, \delta_{11}, \delta_{14}\}$.

Let $\delta : \Sigma_3^* \to \Sigma_2^*$ be the morphism that is defined by
0 → 00100101001001011011010010010100100101001001011011010010010100
10010100100101101101001001010010010110110100100101001001010010100100101
10110100100101001001010010010110110100100101001001010010100100101101101
01101101011011010010010100100101101101001001010010010100100101101101
011,

1 → 00100101001001011011010010010100100101001001011011010010010100
10010100100101101101001001010010010110110100100101001001010010100100101
10110100100101001001010010010110110100100101001001010010100100101101101
01011010010010100110010100100101101101001001010010010100100101101101
01001001010010010100100101101101001001010010010110110100100101001001011011
01010010010110110100100101001001010010010110110100100101001001010010100
10010110110101101101001001010010010100100101101101001001010010100100101
00100101101101001001010010010100100101101101001001010010100100101101101
00100101001001010010010110110100100101001001010010010110110100100101001011011010,

2 → 01001010010010100100101101101011011010010010100100101001010010100100101
10110100100101001001010010010110110100100101001001010010100100101101101

Lemma 11. Consider the infinite word:
$\mathbf{t}_\delta = \delta(\mathbf{t}) =$ 00100101001001011011010010010100100101001001011011010010...
If $\mathbf{h}_\delta$ contains an instance of the pattern $x\pi^i(x)\pi^j(x)\pi^k(x)$ then this instance

can not be modelled by any tuple of the set $\{\delta_1, \delta_2, \delta_3, \delta_4, \delta_5, \delta_6, \delta_9, \delta_{10}, \delta_{13}, \delta_{14}\}$.

Let $\delta : \Sigma_3^* \to \Sigma_4^*$ be the morphism that is defined by

$0 \to$ 0221220010020033133001002001121100200100221220010020011211002
0010022322001002001121100200100221220010020011211002001002232200100
2001121100200100221220010020011211002001002232200100200112110020010
0221220010020011211002001003313300200100221220010020011211002001
0022122001002001131100200100221220010020011211002001002212200100200
0113110020010022122001002001121100200100221220010020011311002001002
2212200100200112110020010,

$1 \to$ 0221220010020033233001002001121100200100221220010020011211002
0010022322001002001121100200100221220010020011211002001002232200100
2001121100200100221220010020011211002001002232200100200112110020010
0221220010020011211002001003313300200100221220010020011211002001
0022122001002001131100200100221220010020011211002001002212200100200
0113110020010022122001002001121100200100221220010020011311002001002
2212200100200112110020010,

$2 \to$ 0221220010020033133001002001121100200100221220010020011211002
0123220010020011211002001002212200100200112110020010022322001002001
1211002001002212200100200112110020010022322001002001121110020010022
2200100200112110020010033133002001002212200100200112110020010022122
0010020011311002001002212200100200112110020010022122001002001131100
2001002212200100200112110020010022122001002001131100200100221220010
0200112110020010

Lemma 12. Consider the infinite word:

$\mathbf{h}_\delta = \delta(\mathbf{h}) =$ 022122001002003313300100200112110020010022122001002...
If $\mathbf{h}_\delta$ contains an instance of the pattern $x\pi^i(x)\pi^j(x)\pi^k(x)$ then this instance can not be modelled by any tuple of the set $\{\delta_1, \delta_2, \delta_3, \delta_4, \delta_5, \delta_6, \delta_9, \delta_{11}, \delta_{12}, \delta_{14}\}$.

Let $\delta : \Sigma_3^* \to \Sigma_2^*$ be the morphism that is defined by

$0 \to$ 0010011011011001001101101100100100110110011001001101100100100110
1101100100110110110010010011011001001001101100100100110110110010010011
1101100100110110110010010011011001001001101100100100110

```
1101100100100110110010010011011001,
1  → 0010011011011001001101101100100100110110010010011011001001001101100100110
1101100100110110110010010011011001001001101100100100110110110010010011011001
1101100100110110110010010011011001001001101101100100110110110010010011011001
1101100100100110110010010011011001,
2  → 0010011011011001001101101100100100110110010010011011001001001101100100110
0110110010011011011001001001101100100100110110010010011011011001001101100110
0110110010011011011001001001101100100100110110010010011011011001001101100110
0110110010010011011001001001101100
```

Lemma 13. If $\mathbf{h}_\delta = \delta(\mathbf{h})$ contains an instance of the pattern $x\pi^i(x)\pi^j(x)\pi^k(x)$ then this instance can not be modelled by any tuple of the set $\{\delta_1, \delta_2, \delta_3, \delta_4, \delta_5, \delta_6, \delta_9, \delta_{11}, \delta_{13}, \delta_{14}\}$.

Let $\delta : \Sigma_3^* \to \Sigma_4^*$ be the morphism that is defined by

```
0 → 0222112220001100011000110002200022000110001100011000220002200011
1000110001100022000220001100011000110002220022200011000110001100022
0002200011000110001100022000220001100011000110002200022000110001100
0110002220022200011000110001100022000220001100011000110002200022000
1100011000110002200022000110001100011000222002220001100011000110002
2000220001100011000110002200022000110001100011000220002200011000110
001100,
1 → 0222332220001100011000110002200022000110001100011000220002200011
1000110001100022000220001100011000110002220022200011000110001100022
0002200011000110001100022000220001100011000110002200022000110001100
0110002220022200011000110001100022000220001100011000110002200022000
1100011000110002200022000110001100011000222002220001100011000110002
2000220001100011000110002200022000110001100011000220002200011000110
001100,
2 → 0222112220001100011000110002200022000110001100011000220002200011
1000110001100022000220001100011000110002220022200011000110001100022
0002200011000110001100022000220001100011000110002200022000110001100
0110002220022200011000110001100022000220001100011000110002200022000
1100011000110002200022000110001100011000222002220001100011000110002
2000220001100011000110002200022000110001100011000220002200011000110
001100
```

Lemma 14. If $\mathbf{h}_\delta = \delta(\mathbf{h})$ contains an instance of the pattern $x\pi^i(x)\pi^j(x)\pi^k(x)$ then this instance can not be modelled by any tuple of the set $\{\delta_1, \delta_2, \delta_3, \delta_4, \delta_5, \delta_7, \delta_8, \delta_{11}, \delta_{13}, \delta_{14}\}$.

Let $\delta : \Sigma_3^* \to \Sigma_5^*$ be the morphism that is defined by

$0 \to 000100010001000200010001000100030001000100010004000100010001000$
$200010001000100040001000100010003,$
$1 \to 000100010001000200010001000100030001000100010004000100010001000$
$300010001000100020001000100010004,$
$2 \to 000100010001000200010001000100030001000100010004000100010001000$
$300010001000100020001000100010003$

Lemma 15. Consider the infinite word:
$\mathbf{h}_\delta = \delta(\mathbf{h}) = 000100010001000200010001000100030001000100010004000\ldots$
If $\mathbf{h}_\delta$ contains an instance of the pattern $x\pi^i(x)\pi^j(x)\pi^k(x)$ then this instance can not be modelled by any tuple of the set $\{\delta_1, \delta_2, \delta_3, \delta_4, \delta_5, \delta_{10}, \delta_{11}, \delta_{12}, \delta_{13}, \delta_{14}\}$.

Let $\delta : \Sigma_3^* \to \Sigma_5^*$ be the morphism that is defined by

$0 \to 010401030102010301040102010401030102010301040103010201040102010 3,$
$1 \to 010401030102010301040102010401030104010201030102010401020103010 4,$
$2 \to 010301020103010401020104010301020103010401030102010401020103010$

Lemma 16. Consider the infinite word:
$\mathbf{h}_\delta = \delta(\mathbf{h}) = 010201030102010401020103010201050102010301020104010\ldots$
If $\mathbf{h}_\delta$ contains an instance of the pattern $x\pi^i(x)\pi^j(x)\pi^k(x)$ then this instance can not be modelled by any tuple of the sets $\{\delta_1, \delta_2, \delta_5, \delta_6, \delta_7, \delta_8, \delta_9, \delta_{10}, \delta_{11}, \delta_{12}, \delta_{13}, \delta_{14}\}$,

Let $\delta : \Sigma_3^* \to \Sigma_4^*$ be the morphism that is defined by

$0 \to 200233211200230220110210012210211200230220110210012210211200 23$
$022011021001221021120023022011021001331221,$

$1 \to 001301102201200211201221001301102201200211201221001301102201 20$
$021120122100130110220 1,$

$2 \to 200233211200230220110210012210211200230220110210012210211200 23$
$022011021001221021123320021022011021001221021120021022011021001221$
$021120021022011021001221021120021022011021001331221$

Lemma 17. Consider the infinite word:

$\mathbf{h}_\delta = \delta(\mathbf{h}) =$ 20023321120023022011021001221021120023022011021001 22...

If $\mathbf{h}_\delta$ contains an instance of the pattern $x\pi^i(x)\pi^j(x)\pi^k(x)$ then this instance can not be modelled by any tuple of the set $\{\delta_1, \delta_3, \delta_4, \delta_6, \delta_7, \delta_8, \delta_9, \delta_{10}, \delta_{11}, \delta_{13}\}$.

Let $\delta : \Sigma_3^* \to \Sigma_5^*$ be the morphism that is defined by

0 → 22002112002201100122100110220021123322300311331001102201100122
10011022002112002201100122100110220021123322300311331001103301100122100110,

1 → 22002112002201100122100110220021123322300311331001102201100122
10011022002112002201100122100110220021123322300311331001102201100122100110,

2 → 22002112002201100122100110220021123322300311331001102201100122
10011022002112002201100122100110220021123322300311331001104401100122100110

Lemma 18. Consider the infinite word:

$\mathbf{h}_\delta = \delta(\mathbf{h}) =$ 2200211200220110012210011022002112332230031133100110...

If $\mathbf{h}_\delta$ contains an instance of the pattern $x\pi^i(x)\pi^j(x)\pi^k(x)$ then this instance can not be modelled by any tuple of the set $\{\delta_1, \delta_3, \delta_4, \delta_6, \delta_7, \delta_8, \delta_9, \delta_{11}, \delta_{13}, \delta_{14}\}$.

Let $\delta : \Sigma_3^* \to \Sigma_3^*$ be the morphism that is defined by

0 → 2102112102220210010210010211210211210211210222021001021001021001021121021121021121022202100102
1001021001021121021121021121022202102220210010210010210010211210211210211210222021001021001021001021121021121021121022202100102100102
10010211210211210211210222021001021001021001021121021121021121022202100102100102100102112102 11,

1 → 2102112102220210010210010211210211210211210222021001021001021001021121021121021121022202100102100102100102112102112102112102220210222021001021001021001021121021121021121022202100102100102100102112102112102112102220210010210010210010211210211210222021001021001021001021121021121021121022202100102 1

0010210010211210211,

2 → 2102112102202100102100102112102112102112102202100102100102100102
1121021121021121022021001021001021001021121021121021121022021001021
0102100102112102112102202100102100102100102112102112102112102202100
0210010210010211210211210211210220210010210010210010211210211.

Lemma 19. Consider the infinite word:
$\mathbf{h}_\delta = \delta(\mathbf{h}) =$ 2102112102202100102100102112102112102112102202100102...
If $\mathbf{h}_\delta$ contains an instance of the pattern $x\pi^i(x)\pi^j(x)\pi^k(x)$ then this instance can not be modelled by any tuple of the set $\{\delta_1, \delta_2, \delta_4, \delta_6, \delta_8, \delta_9, \delta_{10}, \delta_{11}, \delta_{13}\}$.

Let $\delta : \Sigma_3^* \to \Sigma_5^*$ be the morphism that is defined by

0 → 00100122133133122100100122 1, 1 → 00100122133133122144144122 1,
2 → 44144122133133122100100122 1

Lemma 20. If $\mathbf{h}_\delta = \delta(\mathbf{h})$ contains an instance of the pattern $x\pi^i(x)\pi^j(x)\pi^k(x)$ then this instance can not be modelled by any tuple of the sets $\{\delta_1, \delta_3, \delta_4, \delta_6, \delta_9, \delta_{10}, \delta_{11}, \delta_{13}, \delta_{14}\}$.

Let $\delta : \Sigma_3^* \to \Sigma_4^*$ be the morphism that is defined by
0 → 1200012012012000, 1 → 1200012012012000, 2 → 1200013013331000

Lemma 21. Consider the infinite word:
$\mathbf{h}_\delta = \delta(\mathbf{h}) =$ 120001201201200012000120120120001200013013331000120...
If $\mathbf{h}_\delta$ contains an instance of the pattern $x\pi^i(x)\pi^j(x)\pi^k(x)$ then this instance can not be modelled by any tuple of the set $\{\delta_1, \delta_3, \delta_4, \delta_7, \delta_8, \delta_{10}, \delta_{11}, \delta_{12}, \delta_{13}\}$.

Let $\delta : \Sigma_3^* \to \Sigma_4^*$ be the morphism that is defined by

0 → 0001221000122100012210002111022011022201102201110220110222011022
0110220111022011022201102201112,

1 → 0001221000122100012210001222011022011102201102220110220111022011
1022011022201102201110220110221,

2 → 0001221000122100012210002111022011022201102201110220110222011022
0110220111022011022201102201113

Lemma 22. If $\mathbf{h}_\delta = \delta(\mathbf{h})$ contains an instance of the pattern $x\pi^i(x)\pi^j(x)\pi^k(x)$ then this instance can not be modelled by any tuple of the set $\{\delta_1, \delta_3, \delta_4, \delta_7, \delta_8,$ $\delta_{10}, \delta_{11},$
$\delta_{13}, \delta_{14}\}$.

Let $\delta : \Sigma_3^* \to \Sigma_3^*$ be the morphism that is defined by
$0 \to 00122211122210001110$, $1 \to 00122200011120002220$,
$2 \to 00122211122210002220$

Lemma 23. Consider the infinite word:
$\mathbf{h}_\delta = \delta(\mathbf{h}) = 00122211122210001110001222000111200022200012221111222\ldots$
If $\mathbf{h}_\delta$ contains an instance of the pattern $x\pi^i(x)\pi^j(x)\pi^k(x)$ then this instance can not be modelled by any tuple of the set $\{\delta_1, \delta_3, \delta_4, \delta_7, \delta_8, \delta_{11}, \delta_{12}, \delta_{13}, \delta_{14}\}$.

Let $\delta : \Sigma_3^* \to \Sigma_5^*$ be the morphism that is defined by
$0 \to 0120010211414001$, $1 \to 0120010211414002$, $2 \to 0210020322424001$

Lemma 24. Consider the infinite word:
$\mathbf{h}_\delta = \delta(\mathbf{h}) = 012001021141400101200102114140020210020322424002021\ldots$
If $\mathbf{h}_\delta$ contains an instance of the pattern $x\pi^i(x)\pi^j(x)\pi^k(x)$ then this instance can not be modelled by any tuple of the set $\{\delta_1, \delta_2, \delta_4, \delta_6, \delta_8, \delta_9, \delta_{10}, \delta_{12}\}$.

Let $\delta : \Sigma_3^* \to \Sigma_4^*$ be the morphism that is defined by

$0 \to 200100300200100300201220210020010020122021002001002012202100200$
$100201220210020030010020030010020030010020030133031002001002012202$
$002001002012202100200100201220210020010020122320331330310020010020$
$220210020010020122021002001002012202100200100201223203313303100200$
$220210020010020122021002001002012202100200100201223203313303100200$
$002012202100200100201220210020010020122021002001002012232033133031$
$002012202100200100201220210020010020122021002001002012232033133031$
$020010020122021002001002012202100200100201220210020010020122320331$
$020010020122021002001002012202100200100201220210020010020122322021$
$020010020122021002001002012202100200100201220210020010020322021002$
$010020122021002001002012202100200100201220210020010020322021002001$
$020122021002001002012202100200100201220210020010020322021002001002$
$122021002001002012202100200100201220210$

$1 \to 200100300200100300201220210020010020122021002001002012202100200$
$100201220210020030010020030010020030010020030133031002001002012202$
$002001002012202100200100201220210020010020122320331330310020010020$
$220210020010020122021002001002012202100200100201223203313303100200$
$220210020010020122021002001002012202100200100201223203313303100200$
$002012202100200100201220210020010020122021002001002012232033133031$
$002012202100200100201220210020010020122021002001002012232033133031$
$020010020122021002001002012202100200100201220210020010020122320331$

```
0200100201220210020010020122021002001002012202100200100201223220210
0200100201220210020010020122021002001002012202100200100203220210020
0100201220210020010020122021002001002012202100200100203220210020010
0201220210020010020122021002001002012202100200100203220210020010020
1220210020010020122021002001002012202100,

2 → 2001003002001003002012202100200100201220210020010020122021002002
1002012202100200300100200300100200300102110112102212011210223220210
0200100201220210020010020122021002001002012202100200100203220210020
0100201220210020010020122021002001002012202100200100203220210020010
0201220210020010020122021002001002012202100200100203220210020010020
1220210020010020122021002001002012202100
```

Lemma 25. If $\mathbf{h}_\delta = \delta(\mathbf{h})$ contains an instance of the pattern $x\pi^i(x)\pi^j(x)\pi^k(x)$ then this instance can not be modelled by any tuple of the set $\{\delta_1, \delta_2, \delta_4, \delta_6, \delta_9, \delta_{10}, \delta_{11}, \delta_{12}\}$.

Let $\delta : \Sigma_3^* \to \Sigma_4^*$ be the morphism that is defined by
`0` → `1021110122212000,` `1` → `1021110122212000,` `2` → `1021112133323000`

Lemma 26. Consider the infinite word:
$\mathbf{h}_\delta = \delta(\mathbf{h}) =$ `10211101222120001021112133323000102111213333230001021...`
If $\mathbf{h}_\delta$ contains an instance of the pattern $x\pi^i(x)\pi^j(x)\pi^k(x)$ then this instance can not be modelled by any tuple of the set $\{\delta_1, \delta_2, \delta_4, \delta_8, \delta_{10}, \delta_{11}, \delta_{12}, \delta_{13}\}$.

Let $\delta : \Sigma_3^* \to \Sigma_5^*$ be the morphism that is defined by
`0` → `2121200010103334,` `1` → `3434300010101022,` `2` → `2121200010101022`

Lemma 27. Consider the infinite word:
$\mathbf{h}_\delta = \delta(\mathbf{h}) =$ `212120001010333434343000101033343434300010101022212...`
If $\mathbf{h}_\delta$ contains an instance of the pattern $x\pi^i(x)\pi^j(x)\pi^k(x)$ then this instance can not be modelled by any tuple of the set $\{\delta_1, \delta_2, \delta_4, \delta_8, \delta_{10}, \delta_{12}, \delta_{13}, \delta_{14}\}$.

Let $\delta : \Sigma_3^* \to \Sigma_4^*$ be the morphism that is defined by
`0` → `1022200011121000,` `1` → `1022200011121000,` `2` → `1022200033323000`

Lemma 28. Consider the infinite word:
$\mathbf{h}_\delta = \delta(\mathbf{h}) =$ `10222000111210001022200011121000102220003332300010022...`
If $\mathbf{h}_\delta$ contains an instance of the pattern $x\pi^i(x)\pi^j(x)\pi^k(x)$ then this instance can not be modelled by any tuple of the set $\{\delta_1, \delta_2, \delta_4, \delta_8, \delta_{11}, \delta_{12}, \delta_{13}, \delta_{14}\}$.

Let $\delta : \Sigma_3^* \to \Sigma_5^*$ be the morphism that is defined by
$0 \to 0323224020012020$, $1 \to 0424221020012020$, $2 \to 0323224020012020$

Lemma 29. Consider the infinite word:
$\mathbf{h}_\delta = \delta(\mathbf{h}) = 032322402001202004242210200120200323224020012020032\ldots$
If $\mathbf{h}_\delta$ contains an instance of the pattern $x\pi^i(x)\pi^j(x)\pi^k(x)$ then this instance can not be modelled by any tuple of the set $\{\delta_1, \delta_3, \delta_5, \delta_6, \delta_7, \delta_9, \delta_{10}, \delta_{12}\}$.

Let $\delta : \Sigma_3^* \to \Sigma_4^*$ be the morphism that is defined by

$0 \to 0010021011201002001002101120100200100210112010020010031011201002$
$0010021011201002001002101120100200100210112010020010031011201002001002$
$1011201002001002101120100200100210112010020010031011201002001002101120$
$1002001202210200100200120221220121101120100210112010020010031011201002$
$1002001202210200100200120221220121101120100200120221020010020012022102$
$00,$

$1 \to 0010021011201002001002101120100200100210112010020010031011201002$
$0010021011201002001002101120100200100210112010020010031011201002001002$
$1011201002001002101120100200100210112010020010031011201002001002101120$
$1002001202210200100200120221220121101120100200100210112010020010031011$
$201002,$

$2 \to 0010021011201002001002101120100200100210112010020010021011301002$
$0010021011201002001002101120100200100210112010020010021011301002001002$
$1011201002001002101120100200100210112010020010021011301002001002101120$
$1002001202210200100200120221220121101120100200100210112010020010021011$
30100200100310
11201002

Lemma 30. Consider the infinite word:
$\mathbf{h}_\delta = \delta(\mathbf{h}) = 0010021011201002001002101120100200100210112010020010031011\ldots$
If $\mathbf{h}_\delta$ contains an instance of the pattern $x\pi^i(x)\pi^j(x)\pi^k(x)$ then this instance can not be modelled by any tuple of the set $\{\delta_1, \delta_3, \delta_5, \delta_6, \delta_9, \delta_{10}, \delta_{11}, \delta_{12}\}$.

Let $\delta : \Sigma_3^* \to \Sigma_4^*$ be the morphism that is defined by
$0 \to 1202221011102000$, $1 \to 1202221011102000$, $2 \to 1202223133301000$

Lemma 31. Consider the infinite word:
$\mathbf{h}_\delta = \delta(\mathbf{h}) = 120222101110200012022210111020001202223133301000120 2\ \ldots$
If $\mathbf{h}_\delta$ contains an instance of the pattern $x\pi^i(x)\pi^j(x)\pi^k(x)$ then this instance can not be modelled by any tuple of the set $\{\delta_1, \delta_3, \delta_5, \delta_7, \delta_{10}, \delta_{11}, \delta_{12}, \delta_{13}\}$.

Let $\delta : \Sigma_3^* \to \Sigma_4^*$ be the morphism that is defined by
$0 \to 1211102022201000, \quad 1 \to 1211102022201000, \quad 2 \to 1211132322201000$

Lemma 32. Consider the infinite word:
$\mathbf{h}_\delta = \delta(\mathbf{h}) = 12111020222010001211102022201000121113232220100012 11\ldots$
If $\mathbf{h}_\delta$ contains an instance of the pattern $x\pi^i(x)\pi^j(x)\pi^k(x)$ then this instance can not be modelled by any tuple of the set $\{\delta_1, \delta_3, \delta_5, \delta_7, \delta_{10}, \delta_{12}, \delta_{13}, \delta_{14}\}$.

Let $\delta : \Sigma_3^* \to \Sigma_4^*$ be the morphism that is defined by
$0 \to 1110001211102000, \quad 1 \to 1110001211102000, \quad 2 \to 1110002322201000$

Lemma 33. Consider the infinite word:
$\mathbf{h}_\delta = \delta(\mathbf{h}) = 11100012111020001110001211102000111000232220100011100\ldots$
If $\mathbf{h}_\delta$ contains an instance of the pattern $x\pi^i(x)\pi^j(x)\pi^k(x)$ then this instance can not be modelled by any tuple of the set $\{\delta_1, \delta_3, \delta_5, \delta_7, \delta_{11}, \delta_{12}, \delta_{13}, \delta_{14}\}$.

Let $\delta : \Sigma_3^* \to \Sigma_4^*$ be the morphism that is defined by
$0 \to 0001211102000122201000121110200012220100012111020001222010001211$
$1020001222030001211102220 1,$

$1 \to 0001211102000122201000121110200012220100012111020001222010001211$
$1022201000121110200012220100012111020001222010001211102000122201000$
$1211102220100012111020001222010001211102000122201000121110200012220100$
$0012111022201,$

$2 \to 0001211102000122201000121110200012220100012111020001222030001211$
1022201

Lemma 34. If $\mathbf{h}_\delta = \delta(\mathbf{h})$ contains an instance of the pattern $x\pi^i(x)\pi^j(x)\pi^k(x)$ then this instance can not be modelled by any tuple of the set $\{\delta_1, \delta_3, \delta_7, \delta_{10}, \delta_{11}, \delta_{12}, \delta_{13}, \delta_{14}\}$.

Let $\delta : \Sigma_3^* \to \Sigma_4^*$ be the morphism that is defined by
$0 \to 0101022100202011200101022100202011200101022100202011200101022$
$1021101012201120010102210020201120010102210020201120010102210020201$
$1200101022100202011200101022100233030112 0,$

1 → 0101022100202011200101022100202011200101022100202011200101022100
0211010122011200101022100202011200101022100202011200101022100202011
2
0010102210021101012201120010102210020201120010102210020201120010102
2
1002020112001010221002110101220112001010221002020112001010221002020
1
1200101022100202011200101022100211010133011200101022100202011200101
0
2210020201120010102210020201120010102210021101012201120010102210020
2
0112001010221002020112001010221002020112001010221002110101220112001
0
1022100202011200101022100202011200101022100202011200101022100211010
1
2201120010102210020201120010102210020201120010102210020201120010102
2
1002110101330112001010221002020112001010221002020112001010221002020
1
1200101022100211010122011200101022100202011200101022100202011200101
0
2210020201120010102210021101012201120010102210020201120010102210020
2
0112001010221002020112001010221002110101220112001010221002020112001
0
1022100202011200101022100202011200101022100211010133011 20,

2 → 0101022100202011200101022100202011200101022100202011200101022100
0211010122011200101022100202011200101022100202011200101022100202011
2
0010102210021101012201120010102210020201120010102210020201120010102
2
1002020112001010221002110101220112001010221002020112001010221002020
1
1200101022100202011200101022100211010221002020112001010221002020112
0
0101022100202011200101022100202011212001010221002020112001010221002
0
2011200101022100202011200101022100202033100202011200101022100202011
2
0010102210020201120010102210020201120010103310020201130010102210020
2
0112001010221002110101220112 0.

Lemma 35. If $\mathbf{h}_\delta = \delta(\mathbf{h})$ contains an instance of the pattern $x\pi^i(x)\pi^j(x)\pi^k(x)$ then this instance can not be modelled by any tuple of the set $\{\delta_1, \delta_4, \delta_6, \delta_8, \delta_9, \delta_{10}, \delta_{12}, \delta_{14}\}$.

Let $\delta : \Sigma_3^* \to \Sigma_5^*$ be the morphism that is defined by
0 → 0122100101331001, 1 → 0122100101331001, 2 → 0144100133131001

Lemma 36. Consider the infinite word:
$\mathbf{h}_\delta = \delta(\mathbf{h}) =$ 0122100101331001012210010133100101441001331310010 12 . . .
If $\mathbf{h}_\delta$ contains an instance of the pattern $x\pi^i(x)\pi^j(x)\pi^k(x)$ then this instance can not be modelled by any tuple of the set $\{\delta_1, \delta_4, \delta_6, \delta_8, \delta_9, \delta_{10}, \delta_{13}, \delta_{14}\}$.

Let $\delta : \Sigma_3^* \to \Sigma_4^*$ be the morphism that is defined by
0 → 0001022201110122201112000102220111012220111012220111012220111022
0
2111012220111012220111012220111012221000102220111012220111022202111
0
1222011101222011101222011101222100010222011101222011102220211101221
0
1

```
1101222011101222011101222100010222011101222011102220211101222011101222
2011102220111012220111012220111012220111012220111012222011101222100010
0222011101222011102220211101222011101222011101222011101222100010222011
1101222011102220211101222011101222011101222011101222100010222011101222
2011102220211101222011101222011101222011012221,

1 → 0001022201110122201112000102220111012220111012220111012220111012220222
0211101222011101222011101222011101222100010222011101222011102220211101
1222011101222011101222011101222100010222011101222011102220211101222010
1101222011101222011101222100010222011101222011102220211101222011101222
2011102220111012220111012220111012220111012220111012222011101222100010
0222011101222011102220211101222011101222011101222011101222100010222011
1101222011102220211101222011101222011101222011101222100010222011101222
2011102220211101222011101222011101222011012221,

2 → 0001022201110122201113000102220111012220111012220111012220111012220222
0211101222011101222011101222011101222100010222011101222011102220211101
1222011101222011101222011101222100010222011101222011102220211101222010
1101222011101222011101222100010222011101222011102220211101222011101222
2011102220111012220111012220111012220111012222011101222100010222011101222
0222011101222011102220211101222011101222011101222011101222100010222011
1101222011102220211101222011101222011101222011101222100010222011101222
2011102220211101222011101222011101222011012221
```

Lemma 37. If $\mathbf{h}_\delta = \delta(\mathbf{h})$ contains an instance of the pattern $x\pi^i(x)\pi^j(x)\pi^k(x)$ then this instance can not be modelled by any tuple of the set $\{\delta_1, \delta_4, \delta_8, \delta_{10}, \delta_{11}, \delta_{12}, \delta_{13}, \delta_{14}\}$.

Appendix B

Code listings

This chapter contains source code listings for the computer programs that were used in some of the proofs of this thesis.

The following code try to construct a word as long as possible by always adding a letter to the current word it constructed by backtracking. This program show that there exists no infinite word that contains instances of patterns of the set $\{\delta_1, \delta_2, \delta_3, \delta_6, \delta_7\}$ which is an unavoidable set of patterns mentioned in the table A.1. All other unavoidable sets in this table can be checked by similar codes. Computer programs related to checking these sets can be found at `http://media.informatik.uni-kiel.de/zs/patterns.zip`. They are implemented in Ruby

Checking whether $\{\delta_1, \delta_2, \delta_3, \delta_6, \delta_7\}$ is unavoidable set of pattern

```
require "set"

def permutation2(x1,x2,x3,x4)
    t = x1.length
    arr = Array.new(t,0)
    total = 0
    cycles = t
    while cycles > 0 do
        j = 0
        while (arr[j] != 0 && j<t) do
            j += 1
        end

        if j<t
            arr[j]= t + 1 - cycles
            total += 1
        end

        s_1 = Set.new([x1[j],x2[j],x3[j],x4[j]])
        temp = true
        while (j<t && temp == true) do
```

```
22              temp = false
23              1.upto(t-1) { |i|
24                  if (arr[i]==0)
25                      s_2 = Set.new([x1[i],x2[i],x3[i],x4[i]])
26                      s_3 = Set.new
27                      s_3 = s_1 & s_2
28                      if (s_3.empty? == false)
29                          temp = true
30                          s_1 = s_1 | s_2
31                          arr[i] = t + 1 - cycles
32                      end
33                  end
34              }
35          end
36          cycles = cycles-1
37      end
38
39      if (total>1)
40          puts "more than 1 cycle"
41      end
42  end
43
44  def permutation(x1,x2,x3,x4)
45      t = x1.length
46      arr = Array.new(t,0)
47      total = 0
48      cycles = t
49      while cycles > 0 do
50          j = 0
51          while (arr[j] != 0 && j<t) do
52              j += 1
53          end
54
55          if j<t
56              arr[j]= t + 1 - cycles
57              total += 1
58          end
59
60          s_1 = Set.new([x1[j],x2[j],x3[j],x4[j]])
61          temp = true
62          while (j<t && temp == true) do
63              temp = false
64              1.upto(t-1) { |i|
65                  if (arr[i]==0)
66                      s_2 = Set.new([x1[i],x2[i],x3[i],x4[i]])
```

```
                        s_3 = Set.new
                        s_3 = s_1 & s_2
                        if (s_3.empty? == false)
                            temp = true
                            s_1 = s_1 | s_2
                            arr[i] = t + 1 - cycles
                        end
                    end
                }
            end
            cycles = cycles-1
        end

        if (total>1)
        end

        map_cycles = Hash.new
        0.upto(t-1) {|i|
            if (!(map_cycles.key?(x1[i])))
                map_cycles[x1[i]] = arr[i]
            end
        }

        0.upto(t-1) {|i|
            if (!(map_cycles.key?(x2[i])))
                map_cycles[x2[i]] = arr[i]
            end
        }
        0.upto(t-1) {|i|
            if (!(map_cycles.key?(x3[i])))
                map_cycles[x3[i]] = arr[i]
            end
        }
        0.upto(t-1) {|i|
            if (!(map_cycles.key?(x4[i])))
                map_cycles[x4[i]] = arr[i]
            end
        }

#defining the mappings
    maps = []

    mapping12 = Hash.new
    0.upto(t-1) { |pos|
        #case1: the letter at from[pos] isn't mapped already
```

```
112         if (!( mapping12.key?(x1[pos])))
113             mapping12[x1[pos]] = x2[pos]
114         end
115     }
116     maps.push(mapping12)
117
118     mapping13 = Hash.new
119     0.upto(t-1) { |pos|
120         #case1: the letter at from[pos] isn't mapped already
121         if (!( mapping13.key?(x1[pos])))
122             mapping13[x1[pos]] = x3[pos]
123         end
124     }
125     maps.push(mapping13)
126
127     mapping14 = Hash.new
128     0.upto(t-1) { |pos|
129         #case1: the letter at from[pos] isn't mapped already
130         if (!( mapping14.key?(x1[pos])))
131             mapping14[x1[pos]] = x4[pos]
132         end
133     }
134     maps.push(mapping14)
135
136     mapping21 = Hash.new
137     0.upto(t-1) { |pos|
138         #case1: the letter at from[pos] isn't mapped already
139         if (!( mapping21.key?(x2[pos])))
140             mapping21[x2[pos]] = x1[pos]
141         end
142     }
143     maps.push(mapping21)
144
145     mapping23 = Hash.new
146     0.upto(t-1) { |pos|
147         #case1: the letter at from[pos] isn't mapped already
148         if (!( mapping23.key?(x2[pos])))
149             mapping23[x2[pos]] = x3[pos]
150         end
151     }
152     maps.push(mapping23)
153
154     mapping24 = Hash.new
155     0.upto(t-1) { |pos|
156         #case1: the letter at from[pos] isn't mapped already
```

```
157         if (!( mapping24.key?(x2[pos])))
158             mapping24[x2[pos]] = x4[pos]
159         end
160     }
161     maps.push(mapping24)
162 
163     mapping31 = Hash.new
164     0.upto(t-1) { |pos|
165         #case1: the letter at from[pos] isn't mapped already
166         if (!( mapping31.key?(x3[pos])))
167             mapping31[x3[pos]] = x1[pos]
168         end
169     }
170     maps.push(mapping31)
171 
172     mapping32 = Hash.new
173     0.upto(t-1) { |pos|
174         #case1: the letter at from[pos] isn't mapped already
175         if (!( mapping32.key?(x3[pos])))
176             mapping32[x3[pos]] = x2[pos]
177         end
178     }
179     maps.push(mapping32)
180 
181     mapping34 = Hash.new
182     0.upto(t-1) { |pos|
183         #case1: the letter at from[pos] isn't mapped already
184         if (!( mapping34.key?(x3[pos])))
185             mapping34[x3[pos]] = x4[pos]
186         end
187     }
188     maps.push(mapping34)
189 
190     mapping41 = Hash.new
191     0.upto(t-1) { |pos|
192         #case1: the letter at from[pos] isn't mapped already
193         if (!( mapping41.key?(x4[pos])))
194             mapping41[x4[pos]] = x1[pos]
195         end
196     }
197     maps.push(mapping41)
198 
199     mapping42 = Hash.new
200     0.upto(t-1) { |pos|
201         #case1: the letter at from[pos] isn't mapped already
```

```
202         if (!( mapping42.key?(x4[pos])))
203             mapping42[x4[pos]] = x2[pos]
204         end
205     }
206     maps.push(mapping42)
207
208     mapping43 = Hash.new
209     0.upto(t-1) { |pos|
210         #case1: the letter at from[pos] isn't mapped already
211         if (!( mapping43.key?(x4[pos])))
212             mapping43[x4[pos]] = x3[pos]
213         end
214     }
215     maps.push(mapping43)
216
217     strings = []
218     strings.push(x1)
219     strings.push(x2)
220     strings.push(x3)
221     strings.push(x4)
222
223     alph = Set.new
224     0.upto(t-1){ |i|
225         0.upto(3){ |j|
226             alph.add(strings[j][i])
227         }
228     }
229
230     temp=true
231     0.upto(11){ |i|
232         0.upto(11) {|j|
233             if i!=j
234                 alph.each do |x|
235                     if maps[i].key?(x) && maps[j].key?(x)
236                         if maps[i][x]==maps[j][x]
237                             alph.each do |y|
238                                 if maps[i].key?(y) && maps[j].key?(y) && map_cycles[y]==
    map_cycles[x]
239                                     if maps[i][y]!=maps[j][y]
240                                         temp=false
241                                         return temp
242                                     end
243                                 end
244                             end
245                         end
```

```
                    end
                end
            end
        }
    }
return temp
end

#counting different symbols in a word
def count_letters(str)
    count = Hash.new(0)
    str.delete(" ").each_char { |letter| count[letter]+=1}
    Hash[count.sort_by {|k,v| v}.reverse]
end

#factorizes w into 4 parts of equal length for w of length 4k
def factorize(w)
    fourth = w.length/4
    return w[0, fourth], w[fourth, fourth], w[(-2)*fourth, fourth], w[-fourth, fourth]
end

#checks whether a morphism exists that maps from to to
def morphism_exists?(from, to)
    #lengths have to be equal
    return false if (from.length != to.length)
    mapping = Hash.new
    0.upto(from.length-1) { |pos|
        #case1: the letter at from[pos] isn't mapped already
        if (!(mapping.key?(from[pos])))
            #return false if something else is mapped to the target letter already
            #(it's not injective then)
            return false if (mapping.has_value?(to[pos]))
            #otherwise map from[pos] -> to[pos]
            mapping[from[pos]] = to[pos]
        #case2: the letter is mapped already. check if it's mapped consistently
        else
            return false if (mapping[from[pos]] != to[pos])
        end
    }
    #if no inconsistency is found
    return true
end

#short function that checks if x1=x2 and x1 can be mapped to f by a bijective morphism
def pattern?(x1, x2, f)
```

```
291     if (x1 != x2 || !morphism_exists?(x1, f))
292         return false
293     else
294         return true
295     end
296 end
297
298 #check whether variables are equal to each others or not (requiring not equal variables)
299 def equal?(x1, x2, x3, x4)
300     if ((x1 == x2) && (x1 == x3) && (x1 == x4) && (x2 == x3) && (x2 == x4) && (x3 == x4
        ))
301         return false
302     else
303         return true
304     end
305 end
306
307 #check whether variables are equal to each others or not (requiring equal variables)
308 def equal3?(x1, x2, x3)
309     if ((x1 == x2) && (x1 == x3) && (x2 == x3))
310         return true
311     else
312         return false
313     end
314 end
315
316 #check whether variables are equal to each others or not (requiring equal variables)
317 def equal2?(x1, x2)
318     if ((x1 == x2))
319         return true
320     else
321         return false
322     end
323 end
324
325 #check whether variables are equal to each others or not (requiring not equal variables)
326 def morphisms?(x1, x2, x3, x4)
327     if (morphism_exists?(x1, x2) && morphism_exists?(x1, x3) && morphism_exists?(x1,
        x4) && morphism_exists?(x2, x3) && morphism_exists?(x2, x4) && morphism_exists
        ?(x3, x4))
328         return false
329     else
330         return true
331     end
332 end
```

```
333
334 #we assume that |w|=4k for some integer k
335 def checkxxxx(w)
336     x1, x2, x3, x4 = factorize(w)
337     (!equal?(x1, x2, x3, x4) && !morphisms?(x1, x2, x3, x4))
338 end
339
340 def checkxxfx(w)
341     x1, x2, f1, x3 = factorize(w)
342     (pattern?(x1, x2, f1) && pattern?(x1, x3, f1) && pattern?(x2, x3, f1) && !equal2?(x1,
        f1) && !morphisms?(x1, x2, f1, x3))
343 end
344 def checkxfxf(w)
345     x1, f1, x2, f2 = factorize(w)
346     (pattern?(x1, x2, f1) && pattern?(x1, x2, f2) && pattern?(f1, f2, x1) && !equal2?(x1,
        f1))
347 end
348 def checkxf1xf2(w)
349     x1, f1, x2, f2 = factorize(w)
350     (pattern?(x1, x2, f1) && pattern?(x1, x2, f2) && !equal2?(x1, f1) && !equal2?(f1,f2)
        && !equal2?(x1, f2) && !morphisms?(x1, f1, x2, f2))
351 end
352
353 def checkfxxx(w)
354     f, x1, x2, x3 = factorize(w)
355     (pattern?(x1, x2, f) && pattern?(x1, x3, f) && pattern?(x2, x3, f) && !equal2?(x1,f)
        && !morphisms?(f, x1, x2, x3))
356 end
357
358 def checkxf1f2f3(w)
359     x, f1, f2, f3 = factorize(w)
360     (!equal2?(x, f1) && !equal2?(x, f2) && !equal2?(x, f3) && !equal2?(f1, f2) && !equal2
        ?(f1, f3) && !equal2?(f2, f3) && !morphisms?(x, f1, f2, f3))
361 end
362 def checkxxff(w)
363     x1, x2, f1, f2 = factorize(w)
364     (pattern?(x1, x2, f1) && pattern?(x1, x2, f2) && equal2?(f1,f2) && !equal2?(x1,f1))
365 end
366 def checkxxf1f2(w)
367     x1, x2, f1, f2 = factorize(w)
368     (pattern?(x1, x2, f1) && pattern?(x1, x2, f2) && !equal2?(x1, f1) && !equal2?(x1, f2)
        && !equal2?(f1,f2) && !morphisms?(x1, x2, f1, f2))
369 end
370
371 def checkxffx(w)
```

```
372     x1, f1, f2, x2 = factorize(w)
373     (pattern?(x1, x2, f1) && pattern?(x1, x2, f2) && pattern?(f1, f2, x1) && !equal2?(x1,
          f1))
374 end
375
376 def checkxf1f2f1(w)
377     x1, f1, f2, f3 = factorize(w)
378     (pattern?(f1, f3, x1) && pattern?(f1, f3, f2) && !equal2?(x1, f1) && !equal2?(x1,f2)
          && !equal2?(f1, f2) && !morphisms?(x1, f1, f2, f3))
379 end
380
381 def checkxf1f1f2(w)
382     x1, x2, f1, f2 = factorize(w)
383     (pattern?(x2, f1, x1) && morphism_exists?(x1, f1) && morphism_exists?(x1, f2) && !
          equal2?(x1,f1) && !equal2?(x1, f2) && !equal2?(f1, f2))
384 end
385
386 def checkx1x2ff(w)
387     x1, x2, f1, f2 = factorize(w)
388     (pattern?(f1, f2, x1) && pattern?(f1, f2, x2) && !equal2?(x1, f1) && !equal2?(x2, f1)
          && !equal2?(x1,x2) && !morphisms?(x1, x2, f1, f2))
389 end
390
391 def checkxf1f2f2(w)
392     x1, f1, f2, f3 = factorize(w)
393     (pattern?(f2, f3, x1) && pattern?(f2, f3, f1) && !equal2?(x1, f1)  && !equal2?(x1, f2)
          && !equal2?(f1, f2) && !morphisms?(x1, f1, f2, f3))
394 end
395
396 def checkxxxf(w)
397     x1, x2, x3, f = factorize(w)
398     (pattern?(x1, x2, f) && pattern?(x1, x3, f) && pattern?(x2, x3, f) && !equal2?(x1,f)
          && !morphisms?(x1, x2, x3, f))
399 end
400
401 #checks if the word w contains a factor of the form xf1f2f3, xxf1f2, xf1xf2, xxxf, xxfx
        simultaneously at any position.
402 def checkfxxx_xxfx_xf1f2f1_x1x2ff(w)
403     #factorlength
404     (1..(w.length/4).ceil).each { |d|
405         #starting position
406         (0..w.length - 4*d).each { |i|
407             u = w[i..i + d - 1]
408             fu = w[i+d..i+2*d-1]
409             gu = w[i+2*d..i+3*d-1]
```

```
410                 hu = w[i+3*d..i+4*d-1]
411                 if checkxxxx(u+fu+gu+hu)
412                     #if permutation(u,fu,gu,hu)
413                         puts "The word #{w} contains a factor of the form xxxx here: #{u}|#{
       fu}|#{gu}|#{hu}"
414                      #end
415                     return true
416                 end
417                 if checkxf1f2f3(u+fu+gu+hu)
418                     if permutation(u,fu,gu,hu)
419                     puts "The word #{w} contains a factor of the form xf1f2f3 here: #{u}|#{fu
       }|#{gu}|#{hu}"
420                         permutation2(u,fu,gu,hu)
421
422 return true
423             end
424                 end
425                 if checkxxf1f2(u+fu+gu+hu)
426                     if permutation(u,fu,gu,hu)
427                     puts "The word #{w} contains a factor of the form xxf1f2 here: #{u}|#{fu
       }|#{gu}|#{hu}"
428                         permutation2(u,fu,gu,hu)
429
430 return true
431             end
432                 end
433                 if checkxf1xf2(u+fu+gu+hu)
434                     if permutation(u,fu,gu,hu)
435                     puts "The word #{w} contains a factor of the form xf1xf2 here: #{u}|#{fu
       }|#{gu}|#{hu}"
436                         permutation2(u,fu,gu,hu)
437
438 return true
439             end
440                 end
441                 if checkxxxf(u+fu+gu+hu)
442                     if permutation(u,fu,gu,hu)
443                     puts "The word #{w} contains a factor of the form xxxf here: #{u}|#{fu}|#{
       gu}|#{hu}"
444                         permutation2(u,fu,gu,hu)
445
446 return true
447             end
448                 end
449                 if checkxxfx(u+fu+gu+hu)
```

```
                    if permutation(u,fu,gu,hu)
                    puts "The word #{w} contains a factor of the form xxfx here: #{u}|#{fu}|#{
    gu}|#{hu}"
                        permutation2(u,fu,gu,hu)

return true
          end
             end

      }
    }
    return false
end

#try to produce an infinite word that avoids patterns above by backtracking
puts "Trying to generate a word that avoids patterns of size 4:"
def backtrack1(w)
if (!checkfxxx_xxfx_xf1f2f1_x1x2ff(w))
    $let = count_letters(w).keys.count
    0.upto($let){ |temp|
        backtrack1(w + temp.to_s)
    }
    end
end

#without loss of generality the word starts with 0
backtrack1("0")
```

Codes for Chapter 4

The following code perform checks on a uniform word used in Theorem 4.2.

Checking whether $\{\delta_2, \delta_3, \delta_4, \delta_5, \delta_6, \delta_7, \delta_8, \delta_9, \delta_{10}, \delta_{11}, \delta_{12}, \delta_{13}, \delta_{14}\}$ is avoidable

```
require "set"

def permutation(x1,x2,x3,x4)
    t = x1.length
    arr = Array.new(t,0)
    total = 0
    cycles = t
    while cycles > 0 do
        j = 0
        while (arr[j] != 0 && j<t) do
            j += 1
        end
```

```
13
14          if  j<t
15              arr[j]= t + 1 - cycles
16              total += 1
17          end
18
19          s_1 = Set.new([x1[j],x2[j],x3[j],x4[j]])
20          temp = true
21          while (j<t && temp == true) do
22              temp = false
23              1.upto(t-1) { |i|
24                  if (arr[i]==0)
25                      s_2 = Set.new([x1[i],x2[i],x3[i],x4[i]])
26                      s_3 = Set.new
27                      s_3 = s_1 & s_2
28                      if (s_3.empty? == false)
29                          temp = true
30                          s_1 = s_1 | s_2
31                          arr[i] = t + 1 - cycles
32                      end
33                  end
34              }
35          end
36          cycles = cycles-1
37      end
38
39      if (total>1)
40          return false
41      end
42
43      map_cycles = Hash.new
44      0.upto(t-1) {|i|
45          if (!(map_cycles.key?(x1[i])))
46              map_cycles[x1[i]] = arr[i]
47          end
48      }
49
50      0.upto(t-1) {|i|
51          if (!(map_cycles.key?(x2[i])))
52              map_cycles[x2[i]] = arr[i]
53          end
54      }
55      0.upto(t-1) {|i|
56          if (!(map_cycles.key?(x3[i])))
57              map_cycles[x3[i]] = arr[i]
```

```
58          end
59      }
60      0.upto(t-1) {|i|
61          if (!( map_cycles.key?(x4[i])))
62              map_cycles[x4[i]] = arr[i]
63          end
64      }
65
66  #defining the mappings
67      maps = []
68
69      mapping12 = Hash.new
70      0.upto(t-1) { |pos|
71          #case1: the letter at from[pos] isn't mapped already
72          if (!( mapping12.key?(x1[pos])))
73              mapping12[x1[pos]] = x2[pos]
74          end
75       }
76       maps.push(mapping12)
77
78      mapping13 = Hash.new
79      0.upto(t-1) { |pos|
80          #case1: the letter at from[pos] isn't mapped already
81          if (!( mapping13.key?(x1[pos])))
82              mapping13[x1[pos]] = x3[pos]
83          end
84       }
85      maps.push(mapping13)
86
87      mapping14 = Hash.new
88      0.upto(t-1) { |pos|
89          #case1: the letter at from[pos] isn't mapped already
90          if (!( mapping14.key?(x1[pos])))
91              mapping14[x1[pos]] = x4[pos]
92          end
93       }
94       maps.push(mapping14)
95
96      mapping21 = Hash.new
97      0.upto(t-1) { |pos|
98          #case1: the letter at from[pos] isn't mapped already
99          if (!( mapping21.key?(x2[pos])))
100             mapping21[x2[pos]] = x1[pos]
101         end
102      }
```

```
maps.push(mapping21)

mapping23 = Hash.new
0.upto(t-1) { |pos|
      #case1: the letter at from[pos] isn't mapped already
      if (!(mapping23.key?(x2[pos])))
          mapping23[x2[pos]] = x3[pos]
      end
 }
maps.push(mapping23)

mapping24 = Hash.new
0.upto(t-1) { |pos|
      #case1: the letter at from[pos] isn't mapped already
      if (!(mapping24.key?(x2[pos])))
          mapping24[x2[pos]] = x4[pos]
      end
 }
maps.push(mapping24)

mapping31 = Hash.new
0.upto(t-1) { |pos|
      #case1: the letter at from[pos] isn't mapped already
      if (!(mapping31.key?(x3[pos])))
          mapping31[x3[pos]] = x1[pos]
      end
 }
maps.push(mapping31)

mapping32 = Hash.new
0.upto(t-1) { |pos|
      #case1: the letter at from[pos] isn't mapped already
      if (!(mapping32.key?(x3[pos])))
          mapping32[x3[pos]] = x2[pos]
      end
 }
maps.push(mapping32)

mapping34 = Hash.new
0.upto(t-1) { |pos|
      #case1: the letter at from[pos] isn't mapped already
      if (!(mapping34.key?(x3[pos])))
          mapping34[x3[pos]] = x4[pos]
      end
 }
```

```
148   maps.push(mapping34)
149
150   mapping41 = Hash.new
151   0.upto(t-1) { |pos|
152         #case1: the letter at from[pos] isn't mapped already
153         if (!( mapping41.key?(x4[pos])))
154               mapping41[x4[pos]] = x1[pos]
155         end
156    }
157   maps.push(mapping41)
158
159   mapping42 = Hash.new
160   0.upto(t-1) { |pos|
161         #case1: the letter at from[pos] isn't mapped already
162         if (!( mapping42.key?(x4[pos])))
163               mapping42[x4[pos]] = x2[pos]
164         end
165    }
166   maps.push(mapping42)
167
168   mapping43 = Hash.new
169   0.upto(t-1) { |pos|
170         #case1: the letter at from[pos] isn't mapped already
171         if (!( mapping43.key?(x4[pos])))
172               mapping43[x4[pos]] = x3[pos]
173         end
174    }
175   maps.push(mapping43)
176
177   strings = []
178   strings.push(x1)
179   strings.push(x2)
180   strings.push(x3)
181   strings.push(x4)
182
183   alph = Set.new
184   0.upto(t-1){ |i|
185        0.upto(3){ |j|
186             alph.add(strings[j][i])
187        }
188   }
189
190   temp=true
191   0.upto(11){ |i|
192      0.upto(11) {|j|
```

```
193             if i!=j
194                 alph.each do |x|
195                     if maps[i].key?(x) && maps[j].key?(x)
196                         if maps[i][x]==maps[j][x]
197                             alph.each do |y|
198                                 if maps[i].key?(y) && maps[j].key?(y) && map_cycles[y]==
        map_cycles[x]
199                                     if maps[i][y]!=maps[j][y]
200                                         temp=false
201                                         return temp
202                                     end
203                                 end
204                             end
205                         end
206                     end
207                 end
208             end
209         }
210     }
211
212 return temp
213 end
214
215 #factorizes w into 4 parts of equal length for w of length 4k
216 def factorize(w)
217     fourth = w.length/4
218     return w[0, fourth], w[fourth, fourth], w[(-2)*fourth, fourth], w[-fourth, fourth]
219 end
220
221 #checks whether a morphism exists that maps from to to
222 def morphism_exists?(from, to)
223     #lengths have to be equal
224     return false if (from.length != to.length)
225     mapping = Hash.new
226     0.upto(from.length-1) { |pos|
227         #case1: the letter at from[pos] isn't mapped already
228         if (!(mapping.key?(from[pos])))
229             #return false if something else is mapped to the target letter already
230             #(it's not injective then)
231             return false if (mapping.has_value?(to[pos]))
232             #otherwise map from[pos] -> to[pos]
233             mapping[from[pos]] = to[pos]
234         #case2: the letter is mapped already. check if it's mapped consistently
235         else
236             return false if (mapping[from[pos]] != to[pos])
```

```
237             end
238         }
239         #if no inconsistency is found
240         return true
241     end
242
243     #short function that checks if x1=x2 and x1 can be mapped to f by a bijective morphism
244     def pattern?(x1, x2, f)
245         if (x1 != x2 || !morphism_exists?(x1, f))
246             return false
247         else
248             return true
249         end
250     end
251
252     #check whether variables are equal to each others or not (requiring not equal variables)
253     def equal?(x1, x2, x3, x4)
254         if ((x1 == x2) && (x1 == x3) && (x1 == x4) && (x2 == x3) && (x2 == x4) && (x3 == x4
              ))
255             return false
256         else
257             return true
258         end
259     end
260
261     #check whether variables are equal to each others or not (requiring equal variables)
262     def equal3?(x1, x2, x3)
263         if ((x1 == x2) && (x1 == x3) && (x2 == x3))
264             return true
265         else
266             return false
267         end
268     end
269
270     #check whether variables are equal to each others or not (requiring equal variables)
271     def equal2?(x1, x2)
272         if ((x1 == x2))
273             return true
274         else
275             return false
276         end
277     end
278
279
280     #check whether variables are equal to each others or not (requiring not equal variables)
```

```
281 def morphisms?(x1, x2, x3, x4)
282     if (morphism_exists?(x1, x2) && morphism_exists?(x1, x3) && morphism_exists?(x1,
          x4) && morphism_exists?(x2, x3) && morphism_exists?(x2, x4) && morphism_exists
          ?(x3, x4))
283         return false
284     else
285         return true
286     end
287 end
288
289 #we assume that |w|=4k for some integer k
290 def checkxxxx(w)
291     x1, x2, x3, x4 = factorize(w)
292     (!equal?(x1, x2, x3, x4) && !morphisms?(x1, x2, x3, x4))
293 end
294
295 def checkxxfx(w)
296     x1, x2, f1, x3 = factorize(w)
297     (pattern?(x1, x2, f1) && pattern?(x1, x3, f1) && pattern?(x2, x3, f1) && !equal2?(x1,
          f1) && !morphisms?(x1, x2, f1, x3))
298 end
299 def checkxfxf(w)
300     x1, f1, x2, f2 = factorize(w)
301     (pattern?(x1, x2, f1) && pattern?(x1, x2, f2) && pattern?(f1, f2, x1) && !equal2?(x1,
          f1) && !morphisms?(x1, f1, x2, f2))
302 end
303 def checkxf1xf2(w)
304     x1, f1, x2, f2 = factorize(w)
305     (pattern?(x1, x2, f1) && pattern?(x1, x2, f2) && !equal2?(x1, f1) && !equal2?(f1,f2)
          && !equal2?(x1, f2) && !morphisms?(x1, f1, x2, f2))
306 end
307
308 def checkfxxx(w)
309     f, x1, x2, x3 = factorize(w)
310     (pattern?(x1, x2, f) && pattern?(x1, x3, f) && pattern?(x2, x3, f) && !equal2?(x1,f)
          && !morphisms?(f, x1, x2, x3))
311 end
312
313 def checkxf1f2f3(w)
314     x, f1, f2, f3 = factorize(w)
315     (!equal2?(x, f1) && !equal2?(x, f2) && !equal2?(x, f3) && !equal2?(f1, f2) && !equal2
          ?(f1, f3) && !equal2?(f2, f3) && !morphisms?(x, f1, f2, f3))
316 end
317 def checkxxff(w)
318     x1, x2, f1, f2 = factorize(w)
```

```
319     (pattern?(x1, x2, f1) && pattern?(x1, x2, f2) && equal2?(f1,f2) && !equal2?(x1,f1) &&
          !morphisms?(x1, x2, f1, f2))
320 end
321 def checkxxf1f2(w)
322     x1, x2, f1, f2 = factorize(w)
323     (pattern?(x1, x2, f1) && pattern?(x1, x2, f2) && !equal2?(x1, f1) && !equal2?(x1, f2)
          && !equal2?(f1,f2) && !morphisms?(x1, x2, f1, f2))
324 end
325
326 def checkxffx(w)
327     x1, f1, f2, x2 = factorize(w)
328     (pattern?(x1, x2, f1) && pattern?(x1, x2, f2) && pattern?(f1, f2, x1) && !equal2?(x1,
          f1) && !morphisms?(x1, f1, f2, x2))
329 end
330
331 def checkxf1f2f1(w)
332     x1, f1, f2, f3 = factorize(w)
333     (pattern?(f1, f3, x1) && pattern?(f1, f3, f2) && !equal2?(x1, f1) && !equal2?(x1,f2)
          && !equal2?(f1, f2) && !morphisms?(x1, f1, f2, f3))
334 end
335
336 def checkxf1f1f2(w)
337     x, f1, f2, f3 = factorize(w)
338     (pattern?(f1, f2, x) && pattern?(f1, f2, f3) && !equal2?(x, f1) && !equal2?(x, f3) &&
          !equal2?(f1, f3) && !morphisms?(x, f1, f2, f3))
339 end
340
341 def checkx1x2ff(w)
342     x1, x2, f1, f2 = factorize(w)
343     (pattern?(f1, f2, x1) && pattern?(f1, f2, x2) && !equal2?(x1, f1) && !equal2?(x2, f1)
          && !equal2?(x1,x2) && !morphisms?(x1, x2, f1, f2))
344 end
345
346 def checkxf1f2f2(w)
347     x1, f1, f2, f3 = factorize(w)
348     (pattern?(f2, f3, x1) && pattern?(f2, f3, f1) && !equal2?(x1, f1)  && !equal2?(x1, f2)
          && !equal2?(f1, f2) && !morphisms?(x1, f1, f2, f3))
349 end
350
351 def checkxxxf(w)
352     x1, x2, x3, f = factorize(w)
353     (pattern?(x1, x2, f) && pattern?(x1, x3, f) && pattern?(x2, x3, f) && !equal2?(x1,f)
          && !morphisms?(x1, x2, x3, f))
354 end
355
```

```
356 def checkxfxx(w)
357     x1, f, x2, x3 = factorize(w)
358     (pattern?(x1, x2, f) && pattern?(x1, x3, f) && pattern?(x2, x3, f) && !equal2?(x1,f)
          && !morphisms?(x1, f, x2, x3))
359 end
360
361 def checkxf1f2x(w)
362     x1, f1, f2, x2 = factorize(w)
363     (pattern?(x1, x2, f1) && pattern?(x1, x2, f2) && !equal2?(x1,f1) && !equal2?(x1,f2)
          && !equal2?(f1,f2) && !morphisms?(x1, f1, f2, x2))
364 end
365
366 #checks if the word w contains a factor of the form xxxx or xxf1f2 or xf1xf2 or xf1f2f1 or
        xf1f2f2 or xxxf or xxfx or fxxx or xxff or xfxf or xffx or xf1f1f2 or xf1f2x.
367 def check_Functions(w)
368     #factorlength
369     (1..( w.length/4).ceil).each { |d|
370         #starting position
371         (0.. w.length − 4∗d).each { |i|
372             u = w[i .. i + d − 1]
373             fu = w[i+d..i+2∗d−1]
374             gu = w[i+2∗d..i+3∗d−1]
375             hu = w[i+3∗d..i+4∗d−1]
376             if checkxxxx(u+fu+gu+hu)
377                 puts "The word #{w} contains a factor of the form xxxx here: #{u}|#{fu}|#{
          gu}|#{hu}"
378
379                 return true
380             end
381             if checkxxf1f2(u+fu+gu+hu)
382                 if permutation(u,fu,gu,hu)
383
384                 puts "The word #{w} contains a factor of the form xxf1f2 here: #{u}|#{fu
          }|#{gu}|#{hu}"
385             return true
386         end
387             end
388             if checkxf1xf2(u+fu+gu+hu)
389                 if permutation(u,fu,gu,hu)
390
391                 puts "The word #{w} contains a factor of the form xf1xf2 here: #{u}|#{fu
          }|#{gu}|#{hu}"
392             return true
393         end
394             end
```

```
395             if checkxf1f2f1(u+fu+gu+hu)
396                 if permutation(u,fu,gu,hu)
397
398                 puts "The word #{w} contains a factor of the form xf1f2f1 here: #{u}|#{fu
    }|#{gu}|#{hu}"
399             return true
400         end
401             end
402          if checkxf1f2f2(u+fu+gu+hu)
403                 if permutation(u,fu,gu,hu)
404
405                 puts "The word #{w} contains a factor of the form xf1f2f2 here: #{u}|#{fu
    }|#{gu}|#{hu}"
406             return true
407         end
408             end
409          if checkxxxf(u+fu+gu+hu)
410                 if permutation(u,fu,gu,hu)
411
412                 puts "The word #{w} contains a factor of the form xxxf here: #{u}|#{fu}|#{
    gu}|#{hu}"
413             return true
414         end
415             end
416             if checkxxfx(u+fu+gu+hu)
417                 if permutation(u,fu,gu,hu)
418
419                 puts "The word #{w} contains a factor of the form xxfx here: #{u}|#{fu}|#{
    gu}|#{hu}"
420             return true
421         end
422             end
423             if checkxfxx(u+fu+gu+hu)
424                 if permutation(u,fu,gu,hu)
425
426                 puts "The word #{w} contains a factor of the form xfxx here: #{u}|#{fu}|#{
    gu}|#{hu}"
427             return true
428         end
429             end
430             if checkfxxx(u+fu+gu+hu)
431                 if permutation(u,fu,gu,hu)
432
433                 puts "The word #{w} contains a factor of the form fxxx here: #{u}|#{fu}|#{
    gu}|#{hu}"
```

```
434             return true
435         end
436             end
437             if checkxxff(u+fu+gu+hu)
438                 if permutation(u,fu,gu,hu)
439
440                 puts "The word #{w} contains a factor of the form xxff here: #{u}|#{fu}|#{
    gu}|#{hu}"
441             return true
442         end
443             end
444             if checkxfxf(u+fu+gu+hu)
445                 if permutation(u,fu,gu,hu)
446
447                 puts "The word #{w} contains a factor of the form xfxf here: #{u}|#{fu}|#{
    gu}|#{hu}"
448             return true
449         end
450             end
451             if checkxffx(u+fu+gu+hu)
452                 if permutation(u,fu,gu,hu)
453
454                 puts "The word #{w} contains a factor of the form xffx here: #{u}|#{fu}|#{
    gu}|#{hu}"
455           return true
456         end
457             end
458          if checkxf1f1f2(u+fu+gu+hu)
459                 if permutation(u,fu,gu,hu)
460
461                 puts "The word #{w} contains a factor of the xf1f1f2 here: #{u}|#{fu}|#{
    gu}|#{hu}"
462             return true
463         end
464             end
465           if checkxf1f2x(u+fu+gu+hu)
466                 if permutation(u,fu,gu,hu)
467
468       puts "The word #{w} contains a factor of the xf1f2x here: #{u}|#{fu}|#{gu}|#{hu}"
469           return true
470         end
471             end
472       }
473     }
474   return false
```

```
end

puts "checking if the word contains any patterns"
check_Functions("0123041203410234013240312430213401234021342013240124012304120341")
```

Codes for Chapter 4

The following codes are MiniZinc input and model files described in the chapter 5. The generation of the data files is done by executing a Java program.

Input file for Checking the Avoidability of Formulas with Reversal problem

```
sigma = 5;
wordLength = 5;
nrPatterns = 1;
maxPatternLength = 4;

patterns = array2d(1..nrPatterns, 1..maxPatternLength, [
  % x1x2x2x1r
  1, 2, 2, -1,
]);

nrVarsInPattern = array1d(1..nrPatterns, [
  2,
]);

morphicWordLength = 10;
morphicWord = array1d(1..morphicWordLength, [
  0, 1, 1, 0, 1, 0, 0, 1, 1, 0,
]);

numberOfMorphicWordImages = 2;
morphicWordImagesLengths = array1d(1..numberOfMorphicWordImages, [
  3, 2,
]);
```

Codes for Chapter 5

The following code is a .mzn model file for the Checking the Avoidability of Formulas with Reversal problem mentioned in the chapter 5.

Model file for Checking the Avoidability of Formulas with Reversal problem

```
% input parameters
int: sigma;
int: wordLength;
int: nrPatterns;
int: maxPatternLength;
array [1..nrPatterns, 1..maxPatternLength] of int: patterns;
array [1..nrPatterns] of int: nrVarsInPattern;
int: morphicWordLength;
array [1..morphicWordLength] of int: morphicWord;
int: numberOfMorphicWordImages;
array [1..numberOfMorphicWordImages] of int: morphicWordImagesLengths;

% output decision variables
array [1..wordLength] of var 1..sigma: word;
array [1..(sum(i in 1..morphicWordLength)(morphicWordImagesLengths[morphicWord[i]
    + 1]))] of var 1..sigma: finalWord;

% returns lenght of variable indicated by 'variableIndex' based on 'labelIndex'
function int: length(int: variableIndex, int: wordLength, int: nrVars, int: label) =
  if (ceil(label / pow(wordLength, (nrVars - variableIndex + 1) - 1)) mod wordLength
      == 0) then
    wordLength
  else
    ceil(label / pow(wordLength, (nrVars - variableIndex + 1) - 1)) mod wordLength
  endif;

predicate avoidPatterns(array[int] of var int: word, int: length) =
  % for each pattern
  forall(p in 1..nrPatterns)
  (
    % for each position in word
    forall(start in 1..length)
    (
      let {
        % maximum number of labels based on number of variables
        int: maxNumberOfLabels = pow(length, nrVarsInPattern[p]);
      } in
      % move through all labels
      forall(i in 1..maxNumberOfLabels)
      (
```

```
39        % if lengths assigned to variables make sense
40        if (start + sum(k in 1..maxPatternLength where patterns[p, k] != 0) (length(abs(
      patterns[p, k]), length, nrVarsInPattern[p], i)) - 1 <= length) then
41        (
42          exists(varOcc in 1..maxPatternLength where patterns[p, varOcc] != 0)
43          (
44            exists(nextOcc in (varOcc + 1)..maxPatternLength where abs(patterns[p,
      varOcc]) == abs(patterns[p, nextOcc]))
45            (
46              let {
47                var int: varLength = length(abs(patterns[p, varOcc]), length,
      nrVarsInPattern[p], i),
48              } in
49              exists(l in 1..varLength)
50              (
51                let {
52                  var int: occInWord = start + (if varOcc > 1 then
53                    (sum(k in 1..(varOcc - 1))(length(abs(patterns[p, k]), length,
      nrVarsInPattern[p], i))) else 0 endif),
54                  var int: nextOccInWord = occInWord +
55                    sum(k in varOcc..(nextOcc - 1))
56                      (length(abs(patterns[p, k]), length, nrVarsInPattern[p], i)),
57                  var int: posFirst = occInWord + l - 1,
58                  var int: posSecond =
59                    (if (patterns[p, varOcc] == patterns[p, nextOcc]) then
60                      (nextOccInWord + l - 1)
61                    else (nextOccInWord + varLength - (l - 1) - 1) endif)
62                } in
63                word[posFirst] != word[posSecond]
64              )
65            )
66          )
67        )
68        else 1=1
69        endif
70      )
71    )
72  )
73;
74
75constraint avoidPatterns(word, wordLength);
76constraint avoidPatterns(finalWord, (sum(i in 1..morphicWordLength) (
      morphicWordImagesLengths[morphicWord[i] + 1])));
77constraint
78  forall (i in 1..morphicWordLength)
```

```
  (
    let {
      var int: importantIndexInWord = (sum (k in 1..(morphicWord[i])) (
      morphicWordImagesLengths[k])) + 1,
      var int: importantIndexInFinalWord = (sum (k in 1..(i − 1)) (
      morphicWordImagesLengths[morphicWord[k] + 1])) + 1,
    } in
    forall (j in 1.. morphicWordImagesLengths[morphicWord[i] + 1])
    (
      word[importantIndexInWord + j − 1] = finalWord[importantIndexInFinalWord + j −
      1]
    )
  );

solve::int_search(finalWord, input_order, indomain_min, complete) satisfy;

output[show(finalWord)];
```

Codes for Chapter 5

The following code is a .dzn data file for the Avoidability of Patterns in the Abelian Sense problem mentioned in the chapter 5.

Input file for Checking the Avoidability of Patterns in the Abelian Sense

```
sigma = 4;
wordLength = 10;
nrPatterns = 1;
maxPatternLength = 2;

patterns = array2d(1..nrPatterns, 1.. maxPatternLength, [
  % xx
  1, 1,
]);

nrVarsInPattern = array1d(1..nrPatterns, [
  1,
]);

morphicWordLength = 2;
morphicWord = array1d(1..morphicWordLength, [ 0, 1, ]);

numberOfMorphicWordImages = 2;
morphicWordImagesLengths = array1d(1..numberOfMorphicWordImages, [5, 5, ]);
```

Codes for Chapter 5

The following code is a .mzn model file for the Avoidability of Patterns in the Abelian Sense problem mentioned in the chapter 5.

Model file for Checking the Avoidability of Patterns in the Abelian Sense

```
% input parameters
int: wordLength;
int: sigma;
int: nrPatterns;
int: maxPatternLength;
int: morphicWordLength;
int: numberOfMorphicWordImages;
array [1..morphicWordLength] of int: morphicWord;
array [1..numberOfMorphicWordImages] of int: morphicWordImagesLengths;
array [1..nrPatterns, 1..maxPatternLength] of int: patterns;
array [1..nrPatterns] of int: nrVarsInPattern;

% output decision variables
array [1..wordLength] of var 1..sigma: word;
array [1..( sum(i in 1..morphicWordLength) (morphicWordImagesLengths[morphicWord[i]
    + 1]))] of var 1..sigma: finalWord;

% returns lenght of variable indicated by 'variableIndex' based on 'labelIndex'
function int: length(int: variableIndex, int: wordLength, int: nrVars, int: label) =
  if (ceil(label / pow(wordLength, (nrVars - variableIndex + 1) - 1)) mod wordLength
      == 0) then
    wordLength
  else
    ceil(label / pow(wordLength, (nrVars - variableIndex + 1) - 1)) mod wordLength
  endif;

predicate avoidPatterns(array[int] of var int: word, int: length) =
  % for each pattern
  forall(p in 1..nrPatterns)
  (
    % for each position in word
    forall(start in 1..length)
    (
      let {
        % maximum number of labels based on number of variables
        int: maxNumberOfLabels = pow(length, nrVarsInPattern[p]);
      } in
      % move through all labels
      forall (i in 1..maxNumberOfLabels)
      (
```

```
39          % if lengths assigned to variables make sense
40          if (start + sum(k in 1..maxPatternLength where patterns[p, k] != 0) (length(abs(
        patterns[p, k]), length, nrVarsInPattern[p], i)) - 1 <= length) then
41          (
42            exists(varOcc in 1..maxPatternLength where patterns[p, varOcc] != 0)
43            (
44              exists(nextOcc in (varOcc + 1)..maxPatternLength where abs(patterns[p,
        varOcc]) == abs(patterns[p, nextOcc]))
45              (
46                let {
47                  var int: varLength = length(abs(patterns[p, varOcc]), length,
        nrVarsInPattern[p], i),
48                } in
49                exists(l in 1..varLength)
50                (
51                  let {
52                    var int: occInWord = start + (if varOcc > 1 then
53                      (sum(k in 1..(varOcc - 1))(length(abs(patterns[p, k]), length,
        nrVarsInPattern[p], i))) else 0 endif),
54                    var int: nextOccInWord = occInWord +
55                      sum(k in varOcc..(nextOcc - 1))
56                        (length(abs(patterns[p, k]), length, nrVarsInPattern[p], i)),
57                    var int: firstPos = occInWord + l - 1,
58                  } in
59                  sum(k in occInWord..(occInWord + length(abs(patterns[p, varOcc]), length,
         nrVarsInPattern[p], i) - 1) where word[k] == word[firstPos]) (word[k]) !=
60                  sum(k in nextOccInWord..(nextOccInWord + length(abs(patterns[p,
        nextOcc]), length, nrVarsInPattern[p], i) - 1) where word[k] == word[firstPos]) (
        word[k])
61                )
62              )
63            )
64          )
65          else 1=1
66          endif
67        )
68      )
69    )
70  ;
71
72  constraint avoidPatterns(word, wordLength);
73  constraint avoidPatterns(finalWord, (sum(i in 1..morphicWordLength) (
        morphicWordImagesLengths[morphicWord[i] + 1])));
74  constraint
75    forall (i in 1..morphicWordLength)
```

```
  (
    let {
      var int: firstPos = (sum (k in 1..( morphicWord[i])) (morphicWordImagesLengths[
      k])) + 1,
      var int: importantIndexInFinalWord = (sum (k in 1..(i − 1)) (
      morphicWordImagesLengths[morphicWord[k] + 1])) + 1,
    } in
    forall (j in 1.. morphicWordImagesLengths[morphicWord[i] + 1])
    (
      word[firstPos + j − 1] = finalWord[importantIndexInFinalWord + j − 1]
    )
  );

solve::int_search(finalWord, input_order, indomain_min, complete) satisfy;

output[show(finalWord)];
```

Codes for Chapter 5

The following code is a .dzn data file for the Avoidability of Patterns under Permutations problem mentioned in the chapter 5.

Input file for Checking Avoidability of Patterns under Permutations

```
sigma = 3;
wordLength = 10;
nrPatterns = 1;
nrPermutations = 6;
maxPatternLength = 3;

repetitions = array3d(1..nrPatterns, 1.. maxPatternLength, 1..4, [
  % x1p1(x1)x1
   1,0,1,0,      1,1,1,0,      1,0,1,0,
]);

permutations = array4d(1..nrPatterns, 1.. nrPermutations, 1..maxPatternLength, 1..sigma, [
  % x1p1(x1)x1
   1,2,3,      1,2,3,      1,2,3,
   1,2,3,      1,3,2,      1,2,3,
   1,2,3,      2,1,3,      1,2,3,
   1,2,3,      2,3,1,      1,2,3,
   1,2,3,      3,1,2,      1,2,3,
   1,2,3,      3,2,1,      1,2,3,
]);

nrVarsInPattern = array1d(1..nrPatterns, [
```

```
  1,
]);

nrPermsInPattern = array1d(1..nrPatterns, [
  1,
]);

morphicWordLength = 2;
morphicWord = array1d(1..morphicWordLength, [
  0, 1,
]);

numberOfMorphicWordImages = 2;
morphicWordImagesLengths = array1d(1..numberOfMorphicWordImages, [
  5, 5,
]);
```

Codes for Chapter 5

The following code is a .mzn model file for the Avoidability of Patterns under Permutations problem mentioned in the chapter 5.

Model file for Checking Avoidability of Patterns under Permutations

```
% input parameters
int: wordLength;
int: sigma;
int: nrPatterns;
int: maxPatternLength;
int: nrPermutations;
int: morphicWordLength;
int: numberOfMorphicWordImages;
array [1..morphicWordLength] of int: morphicWord;
array [1..numberOfMorphicWordImages] of int: morphicWordImagesLengths;
array [1..nrPatterns, 1..maxPatternLength, 1..4] of int: repetitions;
array [1..nrPatterns, 1..nrPermutations, 1..maxPatternLength, 1..sigma] of int:
    permutations;
array [1..nrPatterns] of int: nrVarsInPattern;
array [1..nrPatterns] of int: nrPermsInPattern;

% output decision variables
array [1..wordLength] of var 1..sigma: word;
array [1..( sum(i in 1..morphicWordLength) (morphicWordImagesLengths[morphicWord[i]
    + 1]))] of var 1..sigma: finalWord;

% returns lenght of variable indicated by 'variableIndex' based on 'labelIndex'
```

```
function int: length(int: variableIndex, int: wordLength, int: nrVars, int: label) =
  if (ceil(label / pow(wordLength, (nrVars − variableIndex + 1) − 1)) mod wordLength
      == 0) then
    wordLength
  else
    ceil(label / pow(wordLength, (nrVars − variableIndex + 1) − 1)) mod wordLength
  endif;

predicate avoidPatterns(array[int] of var int: word, int: length) =
  % for each pattern
  forall(p in 1..nrPatterns)
  (
    % for each position in word
    forall(start in 1..length)
    (
      let {
        % maximum number of labels based on number of variables
        int: maxNumberOfLabels = pow(length, nrVarsInPattern[p]);
      } in
      % move through all labels
      forall (i in 1..maxNumberOfLabels)
      (
        % if lengths assigned to variables make sense
        if (start + sum(k in 1..maxPatternLength where repetitions[p, k, 3] != 0) (length(
      repetitions[p, k, 3], length, nrVarsInPattern[p], i)) − 1 <= length) then
        (
          exists(m in 1..nrPermsInPattern[p])
          (
            forall(z in 1..nrPermutations)
            (
              exists(varOcc in 1..maxPatternLength where
                       repetitions[p, varOcc, 3] != 0 /\ repetitions[p, varOcc, 1] == m)
              (
                exists(nextOcc in 1..maxPatternLength where
                         (nextOcc != varOcc /\
                         (repetitions[p, varOcc, 3] == repetitions[p, nextOcc, 3]) /\
                         (repetitions[p, nextOcc, 2] == 0 \/
                         repetitions[p, varOcc, 1] == repetitions[p, nextOcc, 1])))
                (
                  let {
                    var int: varLength = length(abs(repetitions[p, varOcc, 3]), length,
      nrVarsInPattern[p], i),
                  } in
                  exists(l in 1..varLength)
                  (
```

```
                        let {
                          var int: occInWord = start + (if varOcc > 1 then
                            (sum(k in 1..(varOcc − 1))(length(repetitions[p, k, 3], length,
    nrVarsInPattern[p], i))) else 0 endif),
                          var int: nextOccInWord = start + (if nextOcc > 1 then
                            (sum(k in 1..(nextOcc − 1))(length(repetitions[p, k, 3], length,
    nrVarsInPattern[p], i))) else 0 endif),
                          var int: posFirst = occInWord + l − 1,
                          var int: posSecond =
                            (if (repetitions[p, varOcc, 4] == repetitions[p, nextOcc, 4])
    then
                              (nextOccInWord + l − 1)
                            else (nextOccInWord + varLength − (l − 1) − 1) endif)
                        } in
                        permutations[p, z, varOcc, word[posFirst]] !=
                        permutations[p, z, nextOcc, word[posSecond]]
                      )
                    )
                  )
                )
              )
            )
          )
          else 1=1
          endif
        )
      )
    );

constraint avoidPatterns(word, wordLength);
constraint avoidPatterns(finalWord, (sum(i in 1..morphicWordLength) (
      morphicWordImagesLengths[morphicWord[i] + 1])));
constraint
  forall (i in 1..morphicWordLength)
  (
    let {
      var int: importantIndexInWord = (sum (k in 1..(morphicWord[i])) (
      morphicWordImagesLengths[k])) + 1,
      var int: importantIndexInFinalWord = (sum (k in 1..(i − 1)) (
      morphicWordImagesLengths[morphicWord[k] + 1])) + 1,
    } in
    forall (j in 1..morphicWordImagesLengths[morphicWord[i] + 1])
    (
      word[importantIndexInWord + j − 1] = finalWord[importantIndexInFinalWord + j −
      1]
    )
```

```
  );

solve::int_search(finalWord, input_order, indomain_min, complete) satisfy;

output[show(finalWord)];
```

Codes for Chapter 5

Bibliography

[12] *Developments in language theory - 16th international conference, dlt 2012, taipei, taiwan, august 14-17, 2012. proceedings*. Vol. 7410. LNCS. Springer, 2012. ISBN: 978-3-642-31652-4.

[BCN12] Bastian Bischoff, James Currie, and Dirk Nowotka. "Unary patterns with involution". In: *Int. J. Found. Comput. Sci.* 23.8 (2012), pp. 1641–1652. DOI: 10.1142/S0129054112400679.

[Ber71] J. Bernoulli. *"Sur une nouvelle espece de calcul". in: recueil pour les as- tronomes 1*. 1771.

[BO15] Golnaz Badkobeh and Pascal Ochem. "Characterization of some binary words with few squares". In: *Theor. Comput. Sci.* 588 (2015), pp. 73–80. DOI: 10.1016/j.tcs.2015.03.044. URL: https://doi.org/10.1016/j.tcs.2015.03.044.

[BP07] Jean Berstel and Dominique Perrin. "The origins of combinatorics on words". In: *Eur. J. Comb.* 28.3 (Apr. 2007), pp. 996–1022. ISSN: 0195-6698. DOI: 10.1016/j.ejc.2005.07.019. URL: http://dx.doi.org/10.1016/j.ejc.2005.07.019.

[Car07] Arturo Carpi. "On dejean's conjecture over large alphabets". In: *Theoretical Computer Science* 385.1-3 (2007), pp. 137–151.

[Cas94] Julien Cassiagne. "Motifs évitables et régularité dans les mots". PhD Thesis. Université Paris VI, 1994.

[CBE11] Alkan C, Coe BP, and Eichler EE. "Genome structural variation discovery and genotyping." In: *Nature Rev Genet* 12 (2011), pp. 363–376. URL: https://doi.org/10.1038/nrg2958.

[CMN+18] James D. Currie, Florin Manea, Dirk Nowotka, and Kamellia Reshadi. "Unary patterns under permutations". In: *Theor. Comput. Sci.* 743 (2018), pp. 72–82. DOI: 10.1016/j.tcs.2018.05.033. URL: https://doi.org/10.1016/j.tcs.2018.05.033.

[CMN15] James D. Currie, Florin Manea, and Dirk Nowotka. “Unary patterns with permutations”. In: *Developments in Language Theory*. Vol. 9168. Lecture Notes in Computer Science. Springer, 2015, pp. 191–202.

[CP75] Schmid CW and Deininger PL. “Sequence organization of the human genome”. In: *Cell* 6 (1975), pp. 345–358. URL: https://doi.org/10.1038/nrg2958.

[Cur05] J. Currie. “Pattern avoidance: themes and variations”. In: *Theoret. Comput. Sci.* 339.1 (2005), pp. 7–18. DOI: 10.1016/j.tcs.2005.01.004.

[Dej72] Françoise Dejean. “Sur un théorème de Thue”. In: *J. Comb. Theory, Ser. A* 13.1 (1972), pp. 90–99.

[DMR+17] Chen Fei Du, Hamoon Mousavi, Eric Rowland, Luke Schaeffer, and Jeffrey Shallit. “Decision algorithms for fibonacci-automatic words, II: related sequences and avoidability”. In: *Theor. Comput. Sci.* 657 (2017), pp. 146–162. DOI: 10.1016/j.tcs.2016.10.005. URL: https://doi.org/10.1016/j.tcs.2016.10.005.

[FBF+03] N. Pytheas Fogg, Valerie Berthe, Sebastien Ferenczi, Christian Mauduit, and Anne Siegel. *Substitutions in dynamics, arithmetics and combinatorics*. Lecture Notes in Mathematics. Springer Berlin Heidelberg, 2003. ISBN: 9783540457145.

[Gau00] C. F. Gauß. *Werke. vol. xiii. teubner, leipzig*. 1900.

[GC90] JW Gaubatz and RG Cutler. “Mouse satellite DNA is transcribed in senescent cardiac muscle”. In: *J Biol Chem* 265 (1990), pp. 17753–17758.

[GMM+13] Paweł Gawrychowski, Florin Manea, Robert Mercaş, Dirk Nowotka, and Catalin Tiseanu. “Finding pseudo-repetitions”. In: *30th International Symposium on Theoretical Aspects of Computer Science, STACS 2013, February 27 - March 2, 2013, Kiel, Germany*. Ed. by Natacha Portier and Thomas Wilke. Vol. 20. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2013, pp. 257–268.

[GMN13] Pawel Gawrychowski, Florin Manea, and Dirk Nowotka. "Discovering hidden repetitions in words". In: *The Nature of Computation. Logic, Algorithms, Applications - 9th Conference on Computability in Europe, CiE 2013, Milan, Italy, July 1-5, 2013. Proceedings*. Ed. by Paola Bonizzoni, Vasco Brattka, and Benedikt Löwe. Vol. 7921. Lecture Notes in Computer Science. Springer, 2013, pp. 210–219. ISBN: 978-3-642-39052-4. DOI: 10.1007/978-3-642-39053-1_24. URL: https://doi.org/10.1007/978-3-642-39053-1%5C_24.

[GMN14] Paweł Gawrychowski, Florin Manea, and Dirk Nowotka. "Testing generalised freeness of words". In: *31st International Symposium on Theoretical Aspects of Computer Science (STACS), STACS 2014, March 5-8, 2014, Lyon, France*. Ed. by Ernst W. Mayr and Natacha Portier. Vol. 25. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2014, pp. 337–349.

[Gus97] Dan Gusfield. *Algorithms on strings, trees, and sequences - computer science and computational biology*. Cambridge University Press, 1997. ISBN: 0-521-58519-8.

[Hal64] M. Hall. "Lectures on modern mathematics". In: vol. 2. Wiley, New York, 1964. Chap. Generators and relations in groups – The Burnside problem, pp. 42–92.

[HV17] Stepan Holub and Robert Veroff. "Formalizing a fragment of combinatorics on words". In: *Unveiling Dynamics and Complexity - 13th Conference on Computability in Europe, CiE 2017, Turku, Finland, June 12-16, 2017, Proceedings*. Ed. by Jarkko Kari, Florin Manea, and Ion Petre. Vol. 10307. Lecture Notes in Computer Science. Springer, 2017, pp. 24–31. ISBN: 978-3-319-58740-0. DOI: 10.1007/978-3-319-58741-7_3. URL: https://doi.org/10.1007/978-3-319-58741-7%5C_3.

[JVO+07] Jurka J, Kapitonov VV, Kohany O, and Jurka MV. "Repetitive sequences in complex genomes: structure and evolution". In: *Annu Rev Genomics Hum Genet* 8 (2007), pp. 241–259. URL: https://doi.org/10.1146/annurev.genom.8.080706.092416.

[Kri06] Dalia Krieger. "On critical exponents in fixed points of binary k-uniform morphisms". In: *STACS 2006, 23rd Annual Symposium on Theoretical Aspects of Computer Science, Marseille, France, February 23-25, 2006, Proceedings*. 2006, pp. 104–114. DOI: 10.1007/11672142_7. URL: https://doi.org/10.1007/11672142_7.

[Lot02] M Lothaire. *Algebraic combinatorics on words*. Cambridge University Press, 2002.

[Lot05] M. Lothaire. *Applied combinatorics on words*. Cambridge University Press, 2005.

[Lot97] M. Lothaire. *Combinatorics on words*. Cambridge University Press, 1997.

[MMN12a] Florin Manea, Mike Müller, and Dirk Nowotka. "The avoidability of cubes under permutations". In: *Developments in Language Theory*. Vol. 7410. LNCS. Springer, 2012, pp. 416–427. ISBN: 978-3-642-31652-4.

[MMN12b] Florin Manea, Mike Müller, and Dirk Nowotka. "The avoidability of cubes under permutations". In: *Developments in Language Theory*. Vol. 7410. LNCS. Extended version to appear in J. Comput. Syst. Sci. Springer, 2012, pp. 416–427. ISBN: 978-3-642-31652-4.

[Mou92] Jean Moulin-Ollagnier. "Proof of Dejean's conjecture for alphabets with 5, 6, 7, 8, 9, 10 and 11 letters". In: *Theoretical Computer Science* 95.2 (30 3 1992), pp. 187–205.

[MP02] Batzer MA and Deininger PL. "Alu repeats and human genomic diversity". In: *Nature Rev Genet* 3 (2002), pp. 370–379. URL: https://doi.org/10.1038/nrg798.

[MP92] Filippo Mignosi and Giuseppe Pirillo. "Repetitions in the fibonacci infinite word". In: *ITA* 26 (1992), pp. 199–204.

[MS13] Hamoon Mousavi and Jeffrey Shallit. "Repetition avoidance in circular factors". In: *Developments in Language Theory - 17th International Conference, DLT 2013, Marne-la-Vallée, France, June 18-21, 2013. Proceedings*. 2013, pp. 384–395. DOI: 10.1007/978-3-642-38771-5_34. URL: https://doi.org/10.1007/978-3-642-38771-5%5C_34.

[MSS16] Hamoon Mousavi, Luke Schaeffer, and Jeffrey Shallit. "Decision algorithms for fibonacci-automatic words, I: basic results". In: *RAIRO - Theor. Inf. and Applic.* 50.1 (2016), pp. 39–66. DOI: 10.1051/ita/2016010. URL: https://doi.org/10.1051/ita/2016010.

[Och06] Pascal Ochem. "A generator of morphisms for infinite words". In: *ITA* 40.3 (2006), pp. 427–441. DOI: 10.1051/ita:2006020. URL: https://doi.org/10.1051/ita:2006020.

[Pan84] Jean-Jacques Pansiot. "A propos d'une conjecture de f. dejean sur les répétitions dans les mots". In: *Discrete Applied Mathematics* 7.3 (1984), pp. 297–311.

[Rao11] Michaël Rao. "Last cases of dejean's conjecture". In: *Theoretical Computer Science* 412.27 (2011), pp. 3010–3018.

[Res19] Kamellia Reshadi. "Unary Patterns of Size Four with Morphic Permutations". In: *arXiv:1902.02333* (2019). Published in Proc. Student Research Forum (SRF), SOFSEM 2019.

[RJ10] Britten RJ. "Transposable element insertions have strongly affected human evolution". In: *Proc Natl Acad Sci USA* 107 (2010), pp. 19945–19948. URL: https://doi.org/10.1073/pnas.1014330107.

[Ros17] Matthieu Rosenfeld. "Avoidability of Abelian Repetitions in Words". Theses. Université de Lyon, June 2017. URL: https://tel.archives-ouvertes.fr/tel-01563118.

[RR16] Michaël Rao and Matthieu Rosenfeld. "Avoidability of long k-abelian repetitions". In: *Math. Comput.* 85.302 (2016), pp. 3051–3060. DOI: 10.1090/mcom/3085. URL: https://doi.org/10.1090/mcom/3085.

[Thu06] Axel Thue. "Über unendliche Zeichenreihen". In: *Kra. Vidensk. Selsk. Skrifter, I. Mat. Nat. Kl.* 1906.7 (1906), pp. 1–22.