

# 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:8:1-zs-00000355-a5

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](mailto: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},  
  doi       = {10.21941/kcss/2019/3},  
  series    = {Kiel Computer Science Series},  
  note      = {Dissertation, Faculty of Engineering, Kiel University.}  
}
```

© 2019 by Kamellia Reshadi

# 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  $i, j, k \geq 0$ , 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  $\Sigma_m$ , für  $m \geq 6$ . Später versuchen wir stärkere Resultate zu finden indem wir ein Intervall  $\sigma$  finden, welches für ist  $i, j, k \geq 0$ , 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 einem 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  $w$ ,  $\text{pexp}_i(w)$  ist das Supremum von der Anzahl der Vorkommen jener  $i$  Faktoren, so dass ein Faktor in  $w$  erzeugt wird, und  $\text{RT}_i(k)$  ist dann das Infimum von  $\text{pexp}_i(w)$  über allen Wörtern  $w \in \Sigma_k^\omega$ . Indem wir unendliche ternäre Wörter prüfen, die einige Eigenschaften erfüllen, zeigen wir, dass  $\text{RT}_i(3) = \frac{3i}{2} + \frac{1}{4}$  wenn  $i$  gerade ist und  $\text{RT}_i(3) = \frac{3i}{2} + \frac{1}{6}$  wenn  $i$  ungerade und  $i \geq 3$  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  $i, j, k \geq 0$ , 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  $\Sigma_m$ , for  $m \geq 6$ . Later, we try to find stronger results by finding an interval  $\sigma$  such that given  $i, j, k \geq 0$ , 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  $w$ ,  $\text{pexp}_i(w)$  is the supremum of powers that can be constructed by concatenation of such factors, and  $\text{RT}_i(k)$  is then the infimum of  $\text{pexp}_i(w)$  over all words  $w \in \Sigma_k^\omega$ . Again, by checking infinite ternary words that satisfies some properties, we show that  $\text{RT}_i(3) = \frac{3i}{2} + \frac{1}{4}$  if  $i$  is even and  $\text{RT}_i(3) = \frac{3i}{2} + \frac{1}{6}$  if  $i$  is odd and  $i \geq 3$ .

## 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

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Preliminaries</b>	<b>5</b>
2.1	Patterns and avoidability . . . . .	5
2.2	Morphisms . . . . .	7
2.3	Powers . . . . .	8
<b>3</b>	<b>Unary patterns under permutations</b>	<b>9</b>
3.1	Introduction . . . . .	9
3.2	A general result . . . . .	9
3.3	Avoidability for small alphabets . . . . .	14
3.3.1	Ternary alphabets . . . . .	14
3.3.2	Four and five letter alphabets: the morphic case . . .	18
<b>4</b>	<b>Unary Patterns of Size Four with Morphic Permutations</b>	<b>33</b>
4.1	Introduction . . . . .	33
4.2	Avoidability of patterns under permutations . . . . .	34
4.3	Algorithm to generate avoidable cases . . . . .	42
4.4	Conclusions . . . . .	47
<b>5</b>	<b>On Modelling the Avoidability of Patterns as CSP</b>	<b>49</b>
5.1	Introduction . . . . .	49
5.1.1	The MiniZinc Language . . . . .	49
5.1.2	Code . . . . .	52
5.2	Checking the Avoidability of Formulas with Reversal . . . .	53
5.3	Checking the Avoidability of Patterns in the Abelian Sense .	57
5.4	Checking Avoidability of Patterns under Permutations . . .	58
5.5	Generating Morphic Words . . . . .	62
5.6	Conclusions . . . . .	65

## Contents

<b>6 Repetition Avoidance in Products of Factors</b>	<b>67</b>
6.1 Introduction . . . . .	67
6.2 Preliminaries . . . . .	70
6.3 Main results . . . . .	72
6.4 Concluding remarks . . . . .	77
<b>A Unary patterns under permutations</b>	<b>79</b>
<b>B Code listings</b>	<b>95</b>
<b>Bibliography</b>	<b>129</b>

# List of Tables

3.1	Definition of the values $\delta_a$ , with $1 \leq a \leq 14$ . . . . .	20
4.1	Definition of the values $\delta_a$ , with $1 \leq a \leq 14$ . . . . .	35
4.2	Longest words avoiding factors modelled by some of the sets in the statement of Theorem 4.1. . . . .	42
A.1	All unavoidable sets of $\delta_i$ s . . . . .	79



# 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  $x\pi^i(x)\pi^j(x)\dots$  in which  $x$  is

## 1. Introduction

a word variable as per the original setting, while  $\pi$  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  $\Sigma$  by exchanging  $x$  for a concrete word and substituting  $\pi$  by a function mapping  $x$  to words over  $\Sigma$ . 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 ([IMMN12a; 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  $n \geq 4$  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  $\Sigma_6$  when  $\pi$  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  $\pi$  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  $\text{pexp}_i(w)$  be the supremum of the exponent over the products (concatenation) of  $i$  factors of the word  $w$ . The repetition threshold  $\text{RT}_i(k)$  is then the infimum of  $\text{pexp}_i(w)$  over all words  $w \in \Sigma_k^\omega$ . Mousavi and Shallit obtained that  $\text{RT}_i(2) = 2i$  and  $\text{RT}_2(3) = \frac{13}{4}$ . We show that  $\text{RT}_i(3) = \frac{3i}{2} + \frac{1}{4}$  if  $i$  is even and  $\text{RT}_i(3) = \frac{3i}{2} + \frac{1}{6}$  if  $i$  is odd and  $i \geq 3$ .





# 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  $\Sigma_k = \{0, \dots, k-1\}$  and  $\Sigma'_k = \{1, 2, \dots, k\}$ . We use  $w^R$ , to denote the reversal of word  $w$ . The empty word is denoted by  $\varepsilon$ ,  $w[i]$  denotes the  $i^{\text{th}}$  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  $f : \Sigma_k \rightarrow \Sigma_k$  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  $\pi$ . An instance of a pattern  $p$  in  $\Sigma_k$  is the result of substituting uniformly every variable by a word in  $\Sigma_k^+$  and every function variable by a function over  $\Sigma_k^*$ . A *substitution* (for a word  $\delta$ ) is a mapping  $h : X \rightarrow \Sigma^+$ . For every variable  $x$  occurring in  $\delta$ , we say that  $x$  is *substituted by*  $h(x)$ . The word obtained by substituting every occurrence of a variable  $x$  in  $\delta$  by  $h(x)$  is denoted by  $h(\delta)$ . For instance, we consider the pattern  $\beta = xyy$  and the words  $u = bcaca$ ,  $v = aaaaa$  and have  $h(\beta) = u$ , for  $h(x) = b$  and  $h(y) = ca$ , and  $g(\beta) = v$ , for  $g(x) = a$  and  $g(y) = aa$ . If there exists a substitution  $h$  such that  $h(\delta) = w$ , we say

## 2. Preliminaries

that  $\delta$  matches  $w$ . A pattern is avoidable in  $\Sigma_k$  if there is an infinite word over  $\Sigma_k$  that does not contain any instance of the pattern. The *avoidability problem* for a pattern asks whether, for a given pattern  $\delta$  and an alphabet of terminals  $\Sigma$ , there exists an infinite word (stream)  $u$  over  $\Sigma$ , such that  $\delta$  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  $\Sigma_k$  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  $\phi$ , 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  $\Sigma_k$  if there exists an infinite word over  $\Sigma_k$  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  $\{xx, xy y z x^r, x y y^r x\}$  is a formula. In order to show that this formula is avoidable, an infinite word avoiding each of the patterns  $xx, xy y z x^r, x y y^r x$  should be constructed.

For a word  $w$  over an alphabet  $\Sigma_k$ , the Parikh vector of  $w$  is an array  $\Psi_w$  indexed by the letters of  $\Sigma_k$ , 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  $w \in \Sigma_k^*$  realizes (or matches) in the Abelian sense the pattern  $\delta \in X^*$  if there are  $u_1, \dots, u_{|\delta|} \in \Sigma_k^+$  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  $\Sigma_k^*$  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  $\Sigma_k$  is a permutation of  $\Sigma_k$ , 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  $|f(a)| = n$  holds for all letters  $a \in \Sigma$ .

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  $\pi$ . An instance of a pattern  $p$  in  $\Sigma_k$  is the result of substituting every variable by a word in  $\Sigma_k^+$  and every function variable by a function over  $\Sigma_k^*$ . A pattern is avoidable in  $\Sigma_k$  if there is an infinite word over  $\Sigma_k$  that does not contain any instance of the pattern.

The infinite Thue-Morse word  $t$  is defined as  $t = \lim_{n \rightarrow \infty} \phi_t^n(0)$ , for the morphism  $\phi_t : \Sigma_2^* \rightarrow \Sigma_2^*$  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  $h = \lim_{n \rightarrow \infty} \phi_h^n(0)$ , for the morphism  $\phi_h : \Sigma_3^* \rightarrow \Sigma_3^*$  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  $\pi_{i_j}$  a morphic or antimorphic permutation for  $1 \leq j \leq r$ . Replacing  $x$  by  $\pi_{i_1}^{-1}(x)$  and  $\pi_{i_j}(x)$  by  $\pi_{i_j}(\pi_{i_1}^{-1}(x))$  for  $1 \leq j \leq r$ , 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  $\pi_{j_\ell}$  a morphic or

## 2. Preliminaries

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

### 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  $n \in \mathbb{Z}_{\geq 1}$ ,  $x \in \Sigma^+$ , and  $y$  is a proper prefix of  $x$ . Equivalently,  $z$  has a  $|x|$ -period and  $|y| = |z| \bmod |x|$ . If  $|z| = p$  and  $|x| = q$ , we say that  $z$  is a  $p/q$ -power, or  $z = x^{p/q}$ . For example, the word *sen* 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  $\Sigma^*$  *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  $n \geq 1$ . If  $w$  cannot be expressed as a power of another word, then  $w$  is said to be *primitive*.

# 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  $\Sigma_3$ . Similarly, all such patterns under anti-/morphic permutations are also avoidable in  $\Sigma_4$ , but not in  $\Sigma_6$ . The case of  $\Sigma_5$  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  $\Sigma'_3$  as  $\mathbf{v} = \prod_{i=0}^{\infty} v_i$ , with  $v_i = h_i + 1$ , where  $h$  the infinite 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  $|x| \geq 2$  and  $\pi_i, \pi_j$  and  $\pi_k$  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  $|x| \geq 2$ , where each  $\pi_r$  is a morphic permutation.

### 3. Unary patterns under permutations

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

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

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

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

and letters  $\pi_r(a_\ell)$  and  $\pi_r(a_{\ell+1})$  are distinct, since  $\pi_r$  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  $|x| \geq 2$ , we have  $k_1 \geq 2$ . 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  $2k_1 \geq 4$ , this is impossible; the block lengths in  $\mathbf{u}$  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  $\mathbf{u}$ ; however, this implies that  $k_1 k_1$  is a factor of  $\mathbf{v}$ . Since  $\mathbf{v}$  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  $\mathbf{u}$ , and  $\mathbf{v}$  contains the factor  $(k_1 k_2 \dots k_n)^2$ . This is impossible. Similarly, one argues that  $k_n = 3$  gives a contradiction.

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

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

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

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

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

### 3.2. A general result

and then  $v$  contains the factor  $(k_2 k_3 \cdots k_1)^2$ , 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:**  $k_1, k_n \leq 2$ , and  $n \geq 3$ . 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  $\mathbf{u}$ , we see that  $k_n k_1$  is a factor of  $\mathbf{v}$ , whence  $k_n \neq k_1$ . Since we have already reasoned that  $k_1, k_n \leq 2$ , 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  $\mathbf{u}$ , 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  $\mathbf{u}$ , and  $\pi_i(a_n) \neq \pi_j(a_1)$ , we conclude that  $k_{n-1} k_n$  is a factor of  $\mathbf{v}$ ; 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  $\mathbf{u}$ , so that  $k_{n-1} k_n k_1 k_2 = (k_1 k_n)^2$  is a factor of  $\mathbf{v}$ . This is impossible.

**Case 2bii:**  $k_1, k_n \leq 2$ , 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,  $\mathbf{u}$  contains the word

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

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

### 3. Unary patterns under permutations

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

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

so that  $\mathbf{v}$  contains a word  $\alpha 3123\beta$ ,  $\alpha, \beta \in \{1, 2, 3\}$ ,  $\alpha \geq 2$ . 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

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

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

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

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  $|x| \geq 2$  where each  $\pi_r$  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  $|x| \geq 2$ , where one of the  $\pi_r$  is an antimorphic permutation.

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

We consider the block structure of  $x$ :

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



### 3.2. A general result

where the  $a_i \in \Sigma_3$ , with  $a_\ell \neq a_{\ell+1}$ ,  $k_\ell \geq 1$ ,  $1 \leq \ell \leq n$ . Since  $\pi_i$  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  $|x| \geq 2$ , we have  $n \geq 2$ . 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  $|x| \geq 3$ . If  $k_{n-1} = 1$ ,  $w$  contains  $a_{n-2}^{k_n-2} a_{n-1}^1 a_n^2 \pi_i(a_{n-1}^1) \pi_i(a_{n-2})$ , so that  $\mathbf{v}$  has the factor 121. This is impossible. Thus  $k_{n-1} > 1$ . This forces  $\mathbf{u}$  to contain a block  $a_{n-1}^y a_n^z \pi_i(a_{n-1}^z)$ , where  $y, z \geq 2$  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  $w = a_1 a_2 b_1 b_2 c_1 c_2 d_1 d_2$ , 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  $\pi_i$  is antimorphic. If, instead,  $\pi_i$  is morphic and  $\pi_j$  is antimorphic, (resp.,  $\pi_i$  and  $\pi_j$  are morphic,  $\pi_k$  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.  $\square$

Consequently,  $\mathbf{u}$  has no factor of the form  $\pi_\ell(x) \pi_i(x) \pi_j(x) \pi_k(x)$  with  $|x| \geq 2$  and  $\pi_i, \pi_j$  and  $\pi_k$  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  $|x| \geq 2$  and  $\pi_i, \pi_j$  and  $\pi_k$  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  $\pi_{i_1}(x) \pi_{i_2}(x) \cdots \pi_{i_r}(x)$  with  $|x| \geq 2$ ,  $r \geq 4$ , and  $\pi_{i_j}$  is a morphic or antimorphic permutation for  $1 \leq j \leq r$ .*

### 3. Unary patterns under permutations

## 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  $\Sigma_3$ . 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  $\pi$  an anti-/morphic permutation of  $\Sigma_3$ .

Let us note from the beginning that the permutations of  $\Sigma_3$  are either cycles (i.e.,  $\mathbf{ord}(a) = 3$  for all  $a \in \Sigma_3$ ) or involutions (i.e.,  $\mathbf{ord}(a) \leq 2$  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  $\pi$  is an anti-/morphic permutation of  $\Sigma_3$ .*

*Proof.* By Theorem 3.1 we know that  $\mathbf{u}$  has no factor  $x\pi^i(x)\pi^j(x)x$  with  $|x| \geq 2$ , by just taking in the statement of the theorem  $\pi_i = \pi^i$ ,  $\pi_j = \pi^j$ , and  $\pi_k$  the identity on  $\Sigma_3$ . 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  $\Sigma_3$  (that is, both letters from  $\Sigma_3 \setminus \{a\} = \{b, c\}$ ). 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  $\Sigma_3$  must occur. But this would mean that  $\mathbf{u}$  contains the factor  $abca$ , so  $\mathbf{h}$  should contain the factor 11, a contradiction.  $\square$

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  $\pi$  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  $\Sigma_3$  that has no factor of the form  $x\pi(x)x$ , when  $\pi$  is replaced by a morphic (respectively, antimorphic) permutation.*

### 3.3. Avoidability for small alphabets

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  $\pi$  is replaced by a morphic (respectively, antimorphic) involution of  $\Sigma_2$ .*

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  $\pi$  a morphic permutation of  $\Sigma_3$ .*

*Proof.* Clearly, each morphic permutation  $\pi$  of  $\Sigma_3$  is either a cycle or an involution. In all cases,  $\pi^6$  is the identity morphism on  $\Sigma_3^*$ . 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  $\pi$  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  $\{1, 2\} \subseteq \{i(\bmod 3), j(\bmod 3), k(\bmod 3)\}$ , we get that when replacing  $\pi$  with a cycle of  $\Sigma_3$  (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  $\pi$  is replaced by an involution of  $\Sigma_2$  the result follows from the definition of  $\mathbf{u}_{\mathcal{P}}$ , while when  $\pi$  is replaced by any other permutation of  $\Sigma_3$ , 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  $\{i(\bmod 3), j(\bmod 3), k(\bmod 3)\}$  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.

### 3. Unary patterns under permutations

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  $\pi$  is a cycle of  $\Sigma_3$  we can only obtain words which have the form  $xf(x)xf(x)$  for some morphic permutation  $f$  of  $\Sigma_3$ , while for  $\pi$  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.  $\square$

**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  $\pi$  an antimorphic permutation of  $\Sigma_3$ .*

*Proof.* Just like in the previous proof, for each antimorphic permutation  $\pi$  of  $\Sigma_3$ , we have that  $\pi^6$  is the identity morphism on  $\Sigma_3^*$ . 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  $\pi$  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  $\{1, 2\} \subseteq \{i \pmod 3, j \pmod 3, k \pmod 3\}$ , we get that when replacing  $\pi$  with a cycle of  $\Sigma_3$  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  $\{i \pmod 3, j \pmod 3, k \pmod 3\}$  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  $\pi$  with a cycle of  $\Sigma_3$ , the pattern  $x\pi^3(x)\pi^j(x)\pi^k(x)$  equals  $xx^R\pi^j(x)\pi^k(x)$ . But the instances

### 3.3. Avoidability for small alphabets

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  $\pi$  with an involution of  $\Sigma_3$ , 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  $\pi$  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  $\pi$  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  $\pi$  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  $\pi$  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  $|x| \geq 2$ . We show that  $\mathbf{u}$  does not contain instances of this pattern with  $|x| = 1$ . Assume that  $x = a \in \Sigma_3$ . If  $\pi$  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  $\pi$  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  $\pi$  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  $f(x)$

### 3. Unary patterns under permutations

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  $|x| \geq 2$ . 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  $|x| \geq 3$ . 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  $\pi$  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  $\pi$  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  $\pi$  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  $\pi$  is substituted by an anti-/morphic permutation, are avoidable over  $\Sigma_3$ .*

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  $r \geq 4$ , the  $i_j$  non-negative integers, and  $\pi$  an anti-/morphic permutation, are avoidable over  $\Sigma_3$ .*

## 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  $i, j, k \geq 0$ . For simplicity, the factors  $x$ ,  $\pi^i(x)$ ,  $\pi^j$ , or  $\pi^k(x)$  of the pattern are called  $x$ -items in the following. Our analysis is based on the relation between the possible

### 3.3. Avoidability for small alphabets

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  $\delta_i$ ,  $1 \leq i \leq 14$ . Intuitively, they allow us to define, for a pattern  $x\pi^i(x)\pi^j(x)\pi^k(x)$ , which are the alphabets  $\Sigma_m$  in which we can model certain (non-)equality relationships between the images of the  $x$ -items. For example, in alphabets  $\Sigma_m$  with  $m \geq \delta_1$  we can assign values to  $x$  and  $\pi$  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  $\delta_1$  letters), while for  $\Sigma_m$  with  $m \geq \delta_2$  we can assign values to  $x$  and  $\pi$  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  $\delta_1$  and 0012 for the case defined by  $\delta_2$ . In general, when considering a  $\delta_i$ , 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 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  $\inf \emptyset = \infty$ , so some of the  $\delta_i$ s 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  $\delta_1$  is always non-empty, and also that  $\delta_1 > 3$ . Indeed, at least two of  $i, j, k$  have the same parity, so  $\delta_1$  should not divide 2. Similarly, out of 0,  $i, j, k$  at least two have the same remainder modulo 3, so  $\delta_1$  should also not divide 3. Let  $K = \{\delta_1, \delta_2, \dots, \delta_{14}\}$ .

For a pattern  $x\pi^i(x)\pi^j(x)\pi^k(x)$ , we say that some  $\delta_i$  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

### 3. Unary patterns under permutations

**Table 3.1.** Definition of the values  $\delta_a$ , with  $1 \leq a \leq 14$ .

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

$\delta_i$  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  $\Sigma_m$ , for  $m \geq \max\{\delta_1, \delta_3, \delta_6, \delta_{12}, \delta_{13}\}$ .*

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

Because  $m \geq \delta_1$ , 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  $\delta_1$ , 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  $m \geq \delta_3$  (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  $\text{ord}_f(u[1]) = \delta_3$ . We can derive similar observations for  $\delta_6, \delta_{12}$ , and  $\delta_{13}$ , as well as for all the  $\delta_i$  values we defined.

One can check with the aid of a computer, by a straightforward backtracking algorithm, that if  $m \geq \max\{\delta_1, \delta_3, \delta_6, \delta_{12}, \delta_{13}\}$  then the longest word over  $\Sigma_m$  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



### 3.3. Avoidability for small alphabets

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  $\delta_1 \geq 3$ , is 00120120120111 (adding new letters to this word does not lead to a longer one). This concludes our proof.  $\square$

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

$$0 \rightarrow 01220112, \quad 1 \rightarrow 0, \quad 2 \rightarrow 03110223$$

**Lemma 3.12.** *Consider the infinite word:*

$$\mathbf{h}_\delta = \delta(\mathbf{h}) = 0122011200311022301220112031102230012201120031102230 \dots$$

*If  $\mathbf{h}_\delta$  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  $\delta$  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  $\delta_i$ s 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  $\delta_i$  are equal. Furthermore, if  $u$  is a word of length  $\geq 25$ , 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  $|u| \geq 25$ . Looking at the occurrence of  $u^4$  in  $\mathbf{h}_\delta$  we get that the  $i^{\text{th}}$  occurrence of  $u$  in this repetition can be written

### 3. Unary patterns under permutations

as  $u'_i h(x_{1,i}) h(x_{2,i}) \dots h(x_{i,i}) u''_i$  where  $1 \leq i \leq 4$ ,  $x_{1,i}, \dots, x_{k,i} \in \{0, 1, 2\}$  and, for  $1 \leq j \leq 3$  we have  $u''_j u'_{j+1} = h(x_j)$  for some letter  $x_j \in \{0, 1, 2\}$ . 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  $1 \leq j \leq k$ . Accordingly,  $\mathbf{h}$  should contain a square, which is a contradiction.  $\square$

**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  $\delta_i$ s, and we check the avoidability of each combination.

To begin with, assume that  $\delta_{i_1}, \delta_{i_2}, \delta_{i_3}$  are all  $\delta_i$ s whose representations contain at least three different digits. Then if the word  $\mathbf{t}$  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  $\delta_i$  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  $\delta_i$ s whose representations contain at most two different digits (e.g.,  $\delta_6, \delta_7$ , etc.). Then if the word  $\mathbf{h}$  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  $\mathbf{h}$  does not contain any square, but these  $\delta_i$ s 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  $\delta_i$ s whose representation has at least three different digits, and the other one of them is a  $\delta_i$  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  $\delta_i$ . 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  $\pi$  is replaced by an involution), and the remaining two  $\delta_i$ s 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  $\delta_i$ s whose representation has at most two different digits, and one of them is any  $\delta_i$  except  $\delta_3, \delta_4, \delta_{14}$ , which is represented using at least three different digits. Then  $\mathbf{h}$  does not

### 3.3. Avoidability for small alphabets

contain instances of patterns that can be modelled by any  $\delta_i \in K'$ , because all such  $\delta_i$ s model only instances of the pattern that contain squares or have 4 different letters (e.g., the instances modeled by  $\delta_1$ ).

Assume two of  $\delta_{i_1}, \delta_{i_2}, \delta_{i_3}$  are among  $\delta_i$ s whose representation has at most two different digits, and one of them is  $\delta_3$  or  $\delta_4$  or  $\delta_{14}$ , the word defined in Lemma 3.12 can avoid them.  $\square$

**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  $w_p$  that does not contain any instance of this pattern with  $\pi$  a morphic permutation of  $\Sigma_4$ .*

*Proof.* In this proof, we do a case analysis depending on the possible permutations of  $\Sigma_4$  that  $\pi$  can be assigned to.

If  $\pi$  is assigned to the identical permutation  $1_{\Sigma_4}$ , then the image obtained is a 4-power. If  $\pi$  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  $\delta_i$ , called  $\delta_{i_1}$  in the following. If  $\pi$  is assigned a permutation  $f$  that permutes in a cycle three elements of  $\Sigma_4$  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  $\delta_i$ , called  $\delta_{i_2}$  in the following, and a 4-power. Assume  $\pi$  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  $\delta_i$ , called  $\delta_{i_3}$ , 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  $\{\delta_{i_1}, \delta_{i_2}, \delta_{i_3}\}$ .  $\square$

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

### 3. Unary patterns under permutations

4-powers or are modelled by 0012, and if  $\pi$  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  $\pi$  is substituted by a morphic permutation, are avoidable over  $\Sigma_4$ .*

**Theorem 3.16.** *All patterns  $\pi^{i_1}(x)\pi^{i_2}(x)\dots\pi^{i_r}(x)$  with  $r \geq 4$ , the  $i_j$  non-negative integers, and  $\pi$  a morphic permutation, are avoidable over  $\Sigma_4$ .*

#### 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  $\pi$  functions.

We now show a series of similar results for the case when the function variable  $\pi$  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  $\delta_i$  as in the respective table. The difference is that when we define the digit-representation of  $\delta_i$  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  $\delta_i$  a canonical digit-representation  $0d_1d_2d_3$ . Then, we define the

### 3.3. Avoidability for small alphabets

derived digit-representations of  $\delta_i$ :  $\{0e_1e_2e_3 \mid e_\ell = d_\ell \text{ or } e_\ell = d'_\ell, 1 \leq \ell \leq 3\}$ . The set of digit-representations of a value  $\delta_i$  consists in its canonical digit-representation ( $d_\ell$ ) and its derived digit-representation ( $d'_\ell$ ) which denotes that the parity of the exponent is odd and we apply the permutation on  $x$  in a reverse order). As such, each  $\delta_i$  has 8 digit-representations. For instance  $\delta_2$  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  $\delta_i$  associated to that pattern and  $e_\ell$  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,  $\delta_i$  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  $\delta_i$ .

**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  $\Sigma_m$ , for  $m \geq \max\{\delta_1, \delta_3, \delta_6, \delta_{12}, \delta_{13}\}$ .*

*Proof.* Once more, by a backtracking algorithm, we get that if  $m \geq \max\{\delta_1, \delta_3, \delta_6, \delta_{12}, \delta_{13}\}$ , 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'.  $\square$

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

### 3. Unary patterns under permutations

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

$$0 \rightarrow 0012201123, \quad 1 \rightarrow 0021300123, \quad 2 \rightarrow 0023302113$$

**Lemma 3.18.** *Consider the infinite word:*

$$\mathbf{h}_\delta = \delta(\mathbf{h}) = 00122011230021300123002330211300122011230023302113 \dots$$

*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  $\delta$  is 48. Using a computer program we checked that, there is no factor of the form  $uf(u)g(u)h(u)$  with  $|u| \leq 48$  which can be modelled by the any of the  $\delta_i$ s 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  $|u| \leq 48$  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  $\delta_i$  are equal. Further, if  $u$  is a word of length  $\geq 49$ , 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  $\delta$ .

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  $\mathbf{h}_\delta$  contains the square  $uf^i(u)$  (or, respectively,  $f^j(u)f^k(u)$ ). Looking at the form of the morphism  $\delta$  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  $|u| \geq 49$ , no matter how  $i, j, k$  are chosen. However, it contains factors

### 3.3. Avoidability for small alphabets

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

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

**Lemma 3.19.** *Consider the infinite word:*

$$\mathbf{h}_\delta = \delta(\mathbf{h}) = 0001000100010101000101010001000100010001010100010101000 \dots$$

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  $\{\delta_1, \delta_2, \delta_3, \delta_4, \delta_5, \delta_{10}, \delta_{12}, \delta_{13}, \delta_{14}\}$ .

*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  $\delta$  is 38. Using a computer program we checked that, there is no factor of the form  $uf(u)g(u)h(u)$  with  $|u| \leq 38$  which can be modelled by the any of the  $\delta_i$ s 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  $\geq 38$ , 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  $\delta$ .

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  $\mathbf{h}_\delta$  contains the square  $uf^i(u)$  (or, respectively,  $f^j(u)f^k(u)$ ). Looking at the form of the morphism  $\delta$  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.

### 3. Unary patterns under permutations

It is worth noting that  $\mathbf{h}_\delta$  does not contain any factor  $uf^i(u)f^j(u)f^k(u)$  with  $|u| \geq 39$ , 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  $|u| \leq 38$ , only that, in that case,  $i, j, k$  are not modelled by the  $\delta_i$ s from the statement.  $\square$

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

**Lemma 3.20.** *Consider the infinite word:*

$\mathbf{h}_\delta = \delta(\mathbf{h}) = 000100010001001100010011000100010001000100110001000110001 \dots$

*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  $\{\delta_1, \delta_2, \delta_3, \delta_4, \delta_5, \delta_{11}, \delta_{13}, \delta_{14}\}$ .*

*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  $\delta$  is 38. Using a computer program we checked that, there is no factor of the form  $uf(u)g(u)h(u)$  with  $|u| \leq 38$  which can be modelled by the any of the  $\delta_i$ s 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  $\geq 38$ , 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  $\delta$ .

So, let us assume  $\mathbf{h}_\delta$  contains a factor  $uf^i(u)f^j(u)f^k(u)$ , with  $f$  anti-morphic 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  $\mathbf{h}_\delta$  contains the square  $uf^i(u)$  (or, respectively,  $f^j(u)f^k(u)$ ). Looking at the form of the morphism  $\delta$  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.



### 3.3. Avoidability for small alphabets

It is worth noting that  $\mathbf{h}_\delta$  does not contain any factor  $uf^i(u)f^j(u)f^k(u)$  with  $|u| \geq 39$ , 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  $|u| \leq 38$ , only that, in that case,  $i, j, k$  are not modelled by the  $\delta_i$ s from the statement.  $\square$

**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  $\delta_i$ s, we check the avoidability of each combination.

To begin with, assume that  $\delta_{i_1}, \delta_{i_2}, \delta_{i_3}$  are any  $\delta_i$ s 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  $\delta_i$  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  $\delta_i$ s 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  $\delta_i$ s whose representation has at least three different digits, and the other one of them is a  $\delta_i$  whose representation has at most two different digits, then if the  $\delta_i$  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  $\delta_i$  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.

### 3. Unary patterns under permutations

Assume two of  $\delta_{i_1}, \delta_{i_2}, \delta_{i_3}$  are any of  $\delta_i$ s whose representation has at most two different digits, and one of them is any  $\delta_i$  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  $\delta_i$ s whose representation has at most two different digits, and one of them is  $\delta_3$  or  $\delta_4$  or  $\delta_{14}$ , then the word defined in Lemma 3.18 can avoid them.  $\square$

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  $w_p$  that does not contain any instance of this pattern with  $\pi$  an antimorphic permutation of  $\Sigma_4$ .*

*Proof.* If  $\pi$  is assigned the identical permutation  $1_{\Sigma_4}$ , 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  $\pi$  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  $\delta_i$ .

If  $\pi$  is assigned a permutation  $f$  that permutes cyclicly three elements of  $\Sigma_4$  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  $\delta_i$  and a 4-power.

Assume  $\pi$  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

### 3.3. Avoidability for small alphabets

instances are modelled by exactly one  $\delta_i$  and a 4-power. Our statement follows now from Lemma 3.21.  $\square$

As an example, for  $i = 3, j = 16, k = 2$ , if  $\pi$  is mapped to a 4-cycle, we can obtain instances which are like  $00'00$  or are modelled by  $01'02$ . If  $\pi$  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  $\pi$  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  $\Sigma_6$ . This is shown by the next lemma. The case of  $\Sigma_5$  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  $\Sigma_m$ , for  $m \geq 6$  with  $\pi$  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  $\Sigma_m$  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  $\Sigma_6$ . Indeed, if  $\pi$  is mapped to a 6-cycle, the instances we obtain are modelled by 0112. If  $\pi$  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  $\pi$  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  $\pi$  is mapped to a permutation composed of a 4-cycle and a 2-cycle, the instances we obtain are again modelled by 0112. If  $\pi$  is mapped to an involution, the instances we obtain are modelled by 0001. If  $\pi$  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  $\pi$  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  $\pi$ .

### 3. Unary patterns under permutations

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.  $\square$

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

# 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  $\sigma$  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  $\Sigma_m$ , for  $m \geq k$ , and avoidable in  $\Sigma_m$ , for  $4 \leq m < k$ . The avoidability of  $x\pi^i(x)\pi^j(x)$  in  $\Sigma_2$  and  $\Sigma_3$  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. Unary Patterns of Size Four with Morphic Permutations

### 4.2 Avoidability of patterns under permutations

In this section we try to identify an upper bound on the size of the alphabets  $\Sigma_m$  in which a pattern  $x\pi^i(x)\pi^j(x)\pi^k(x)$ , with  $i, j, k \geq 0$  is unavoidable, when  $\pi$  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  $\delta_a$ , with  $1 \leq a \leq 14$ . Intuitively, they allow us to define, for a pattern  $x\pi^i(x)\pi^j(x)\pi^k(x)$ , which are the alphabets  $\Sigma_m$  in which we can model certain (in-)equality relationships between the images of the  $x$ -items. For example, in alphabets  $\Sigma_m$  with  $m \geq \delta_1$  we can assign values to  $x$  and  $\pi$  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  $\delta_1$  letters). Also, in  $\Sigma_m$  with  $m \geq \delta_2$  we can assign values to  $x$  and  $\pi$  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  $\delta_a$ , we assign different digits to the  $x$ -items that can be mapped to different words in alphabets of size at least  $\delta_a$ . For example, we use the representation 0123 for the case defined through  $\delta_1$  and 0012 for the case defined by  $\delta_2$ . In general, when considering an  $\delta_a$ , 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  $\delta_a$  divides  $t$ . We then put a 1 on 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  $\delta_a$  divides  $(r - s)$ , and so on.

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

## 4.2. Avoidability of patterns under permutations

**Table 4.1.** Definition of the values  $\delta_a$ , with  $1 \leq a \leq 14$ .

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

Thus, for different patterns we will have different parameters.

Recall that  $\inf \emptyset = \infty$ , so the value of some  $\delta_a$ s 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  $\delta_1$  is always non-empty, and also that  $\delta_1 > 3$ . Indeed, at least two of  $i, j, k$  have the same parity, so  $\delta_1$  should not divide 2. Similarly, out of  $0, i, j, k$  at least two have the same remainder modulo 3, so  $\delta_1$  should also not divide 3. Let  $K = \{\delta_1, \delta_2, \dots, \delta_{14}\}$ .

For a pattern  $x\pi^i(x)\pi^j(x)\pi^k(x)$ , we say that one of the numbers  $\delta_a$  (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  $\delta_a$  are equal. An infinite word  $w$  over some alphabet  $\Sigma$  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  $\delta_\ell$ , and  $\delta_p$  respectively, with  $\ell, p \geq 1$ , we say that  $w_1$  is a swapped form of  $w_2$  if there exists a position  $i \leq 4$  such that  $w_1[i] = w_2[i+1]$ , and  $w_1[i+1] = w_2[i]$ , and

#### 4. Unary Patterns of Size Four with Morphic Permutations

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

Let  $\delta$  be the digit representation of  $x\pi^i(x)\pi^j(x)\pi^k(x)$ . We say that  $\delta$  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  $\delta$  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  $\delta$  contains a cube if it is 0001 or 0111. We say  $\delta$  has two squares if it is 0011. Finally,  $\delta$  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  $\delta_n$ s 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  $\delta_1$  and:

- ▷ one of the  $\delta_n$ s whose representation has a prefix or a suffix square, but no gapped cube. That is:  $\delta_2$  or  $\delta_5$ .
- ▷ one of the  $\delta_n$ s that has a gapped square, but does not have two gapped squares. These are  $\delta_3$  or  $\delta_4$ .
- ▷ one of the  $\delta_n$ s that contains cubes or two squares:  $\delta_6$  or  $\delta_9$  or  $\delta_{10}$ .
- ▷ one of the  $\delta_n$ s that contains gapped cubes:  $\delta_7$  or  $\delta_8$ .

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.



## 4.2. Avoidability of patterns under permutations

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  $\delta_1$  and:

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

Moreover, we have the restriction that if we choose  $\delta_2$  then  $\delta_7$  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  $\delta_1$  and  $\delta_{10}$  (the only  $\delta_n$  that has two square factors) as well as:

- ▷ one of the  $\delta_n$ s whose representation has a prefix or a suffix square, but no gapped cube. That is:  $\delta_2$  or  $\delta_5$ .
- ▷ one of the  $\delta_n$ s whose representation has a gapped square, but does not have two gapped squares. That is:  $\delta_3$  or  $\delta_4$ .
- ▷ one of the  $\delta_n$ s whose representation contains gapped cubes:  $\delta_7$  or  $\delta_8$ .

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  $\delta_1, \delta_2, \delta_7$ , and:

- ▷ one of the  $\delta_n$ s whose representations contain cubes ( $\delta_6$  or  $\delta_9$ ), or two square ( $\delta_{10}$ ) and

#### 4. Unary Patterns of Size Four with Morphic Permutations

▷ one of the  $\delta_a$ s for whose representation only the first and last digits are equal, and they are different from all other digits ( $\delta_{14}$ ).

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  $\delta_1$ ,  $\delta_{12}$ ,  $\delta_{13}$ , and  $\delta_{14}$ , as well as one of the  $\delta_a$ s whose representation contains cubes ( $\delta_6$  or  $\delta_9$ ). One example is  $\{\delta_1, \delta_9, \delta_{12}, \delta_{13}, \delta_{14}\}$ .

Let  $\mathcal{S}_6$  be the collection of sets that contain  $\delta_1$ ,  $\delta_{10}$ ,  $\delta_{13}$ ,  $\delta_{14}$ , and

- ▷ one of the  $\delta_a$ s whose representation contains a cube ( $\delta_6$  or  $\delta_9$ ), and
- ▷ one of the  $\delta_a$ s whose representation contains a gapped cube ( $\delta_7$  or  $\delta_8$ ), and
- ▷ one of the  $\delta_a$ s whose representation contains a gapped square ( $\delta_3$  or  $\delta_4$ ).

Here we have the restriction that if we have in a set of  $\mathcal{S}_6$  the element  $\delta_7$ , whose representation has on the first, the second and the last positions the same digit, we should add  $\delta_4$  whose second and last digit are the same. Furthermore, the presence of both  $\delta_4$  and  $\delta_8$  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  $\delta_1$ ,  $\delta_{12}$ ,  $\delta_{13}$ , and one of the  $\delta_a$ s of the set  $\{\delta_2, \delta_5\}$ , and one of the  $\delta_a$ s of the set  $\{\delta_3, \delta_4\}$ , and one of the  $\delta_a$ s 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  $\delta_a$ s 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  $\delta_a$ s 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  $\delta_1$  and all elements of the set  $\{\delta_3, \delta_5, \delta_7, \delta_{14}\}$ , and one of the  $\delta_a$ s 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  $\delta_1$  and ,  $\delta_{10}$ :

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

## 4.2. Avoidability of patterns under permutations

- ▷ 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  $\delta_a$ s of the set  $\{\delta_3, \delta_4\}$ , and
- ▷ one of the  $\delta_a$ s of the set  $\{\delta_6, \delta_9\}$ , and
- ▷ one of the  $\delta_a$ s whose representation has two gapped squares  $\delta_{11}$ .

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  $\delta_2$  or  $\delta_5$  or  $\delta_{10}$  as the  $\delta_a$ s with squares in a set, then  $\delta_3, \delta_4$ , and  $\delta_{14}$  should be selected as gapped squares, respectively, and in the first two conditions,  $\delta_{10}$  and  $(\delta_{12}$  or  $\delta_{13})$ , and in the last condition,  $(\delta_3$  or  $\delta_4)$  and  $(\delta_7$  or  $\delta_8$  or  $\delta_{12}$  or  $\delta_{13})$  should be added to the set. The other restriction is that, if we have the numbers  $\delta_7, \delta_{14}$ , then  $\delta_3$ , and if we have the numbers  $\delta_4, \delta_{14}$ , then  $(\delta_8$  or  $\delta_{12})$  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  $\delta_1$  and (only) one of the following sets:

- ▷  $\delta_3$  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  $\delta_a$  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

#### 4. Unary Patterns of Size Four with Morphic Permutations

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  $4 \leq i \leq 14$ . 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  $5 \leq i \leq 14$ . 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  $6 \leq i \leq 14$ . 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  $10 \leq i \leq 13$ , and  $i + 1 \leq j \leq 14$ , and the set  $\{\delta_1, \delta_4, \delta_i, \delta_j\}$  for all  $12 \leq i \leq 13$ , and  $i + 1 \leq j \leq 14$ . 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  $4 \leq i \leq 5$ , and  $i + 1 \leq j \leq 14$ . 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  $6 \leq i \leq 9$ , and  $i + 1 \leq j \leq 14$ , and the set  $\{\delta_1, \delta_4, \delta_i, \delta_j\}$  for all  $6 \leq i \leq 11$ , and  $i + 1 \leq j \leq 14$ . 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  $5 \leq i \leq 13$ ,  $i + 1 \leq j \leq 14$ , and the set  $\{\delta_1, \delta_i, \delta_j, \delta_k\}$  for all  $5 \leq i \leq 12$ ,  $i + 1 \leq j \leq 13$ , and  $j + 1 \leq k \leq 14$ . 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

## 4.2. Avoidability of patterns under permutations

far as the sets not containing  $\delta_1$  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  $\sigma = \min\{\max(S) \mid S = S_\ell \text{ for some } \ell = 1, \dots, 10\}$ . Then  $\sigma \geq 4$  and  $p$  is unavoidable in  $\Sigma_m$ , for all  $m > \sigma$ .*

*Proof.* Because  $m \geq \delta_1$ , 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  $\delta_1$ , 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  $m \geq \delta_2$  (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  $\text{ord}_f(u[1]) = \delta_2$ . We can derive similar observations for all the  $\delta_a$  parameters involved in the definition of  $\sigma$ .

One can check with the aid of a computer, by a backtracking algorithm, that if  $m \geq \max(S) + 1$ , when  $S = S_\ell$  for some  $\ell = 1, \dots, 10$ , then  $p$  is unavoidable in  $\Sigma_m$ . 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  $\pi$ . But, in most of the cases we need to check, we got the result even when we explicitly allowed  $\pi$  to be only a cycle. In the remaining cases, we needed to allow  $\pi$  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  $\sigma$ .

For instance, looking at  $S_1$ , using a computer program that explores all the possibilities by backtracking we get that if  $m \geq \max\{\delta_1, \delta_2, \delta_4, \delta_6, \delta_7\}$ ,

#### 4. Unary Patterns of Size Four with Morphic Permutations

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

$\{\delta_1, \delta_2, \delta_4, \delta_6, \delta_7\}$	010211002211002211002211000
$\{\delta_1, \delta_2, \delta_7, \delta_{12}, \delta_{13}\}$	01010102020001112220001112220001112220001112
$\{\delta_1, \delta_2, \delta_4, \delta_7, \delta_{10}\}$	012012012011100
$\{\delta_1, \delta_2, \delta_7, \delta_{10}, \delta_{14}\}$	0102010201020103010101020001
$\{\delta_1, \delta_3, \delta_5, \delta_8, \delta_{12}, \delta_{13}\}$	0001112220001112220001112220001112012012012
$\{\delta_1, \delta_2, \delta_3, \delta_6, \delta_{10}, \delta_{11}, \delta_{13}\}$	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  $\pi$  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  $\pi$  to be replaced only by cycles. However, if we allow  $\pi$  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  $m \geq \max\{\delta_1, \delta_2, \delta_3, \delta_6, \delta_{10}, \delta_{11}, \delta_{12}\} + 1$  the pattern is unavoidable.

All the sets of  $\alpha_i$ s that are used to define  $\sigma$  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  $\sigma + 1 \geq 5$ .  $\square$

### 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

### 4.3. Algorithm to generate avoidable cases

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  $\delta_a$  parameters  $S'$  such that  $S'$  does not include any set  $S = S_\ell$  for some  $\ell = 1, \dots, 10$ . In other words,  $K_a$  contains all nonempty strict subsets of the sets  $S = S_\ell$  for some  $\ell = 1, \dots, 10$  as well as any other sets of parameters that do not include any of the sets  $S = S_\ell$  for some  $\ell = 1, \dots, 10$ . We already know that all subsets of the sets  $S = S_\ell$  for some  $\ell = 1, \dots, 10$  are avoidable. Also, all supersets of the sets  $S = S_\ell$  for some  $\ell = 1, \dots, 10$  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,  $K \setminus \{\delta_1\}$  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  $S_i$ , ( $1 \leq i \leq 10$ ), generate all sets of  $\delta_a$ s 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  $\delta_a$ s 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  $\delta_a$ s 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*

#### 4. Unary Patterns of Size Four with Morphic Permutations

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.

$$\begin{aligned}
 & \{\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}\}, \quad \{\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}\}, \quad \{\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}\}, \quad \{\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}\}, \quad \{\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}\}, \quad \{\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}\}, \quad \{\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}\}, \quad \{\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}\}, \quad \{\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}\}, \quad \{\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}\}, \quad \{\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}\}, \quad \{\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}\}, \quad \{\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_{11}, \delta_{13}, \delta_{14}\}, \quad \{\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}\}, \quad \{\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}\}, \quad \{\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}\}, \quad \{\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}\}, \quad \{\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}\}, \quad \{\delta_1, \delta_4, \delta_8, \delta_{10}, \delta_{11}, \delta_{12}, \delta_{13}, \delta_{14}\}
 \end{aligned}$$

*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  $\delta: \Sigma_3^* \rightarrow \Sigma_5^*$  is the morphism defined by  
 $0 \rightarrow 0123041203410234, 1 \rightarrow 0132403124302134, 2 \rightarrow 0123402134201324$

We show that if  $h_\delta$  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  $T$ . Assume, for the sake of contradiction, that  $h_\delta$  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  $f$



### 4.3. Algorithm to generate avoidable cases

morphic permutation). The maximum length of a factor of  $h_\delta$  that does not contain a full image of any letter of the ternary Thue word under  $\delta$  is 30. Using a computer program we checked that  $h_\delta$  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  $\geq 31$ , 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  $\delta$  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  $\delta$ , or the image of 2 under  $\delta$ , 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); 01230412034102340132403124302 aligned with 23402134201324012304120341023 (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-

#### 4. Unary Patterns of Size Four with Morphic Permutations

tion of the letters 2 in the image of 2, and get again only contradictions. Therefore, no instance of the pattern is contained in  $h_\delta$ . This concludes our proof.

We were in the case when  $u$  is a word of length  $\geq 31$ . So each of the factors  $u, f^i(u), f^j(u), f^k(u)$  contains a full image of a letter of  $\mathbf{h}$ . 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  $\mathbf{h}$  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  $\mathbf{h}$ ). 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  $\mathbf{h}$ , 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  $\sigma$ , such that  $p$  is avoidable in  $\Sigma_m$  for  $m \leq \sigma - 1$  and unavoidable in  $\Sigma_m$  for  $m \geq \sigma + 1$ .*

*Proof.* By [CMN15; CMN+18], we get that all the unary patterns of size 4 with permutations are avoidable in  $\Sigma_m$  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  $\Sigma_m$  for all  $m \geq 3$ . If  $i = k$ , then the pattern is avoidable in  $\Sigma_m$ , for all  $m \geq 3$ , 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  $\delta_a$ , with  $1 \leq a \leq 14$ , for the given pattern. Then, we consider the sets  $S_i$ , with  $1 \leq i \leq 10$ , and compute  $\sigma = \min\{\max(S) \mid S = S_\ell \text{ for some } \ell = 1, \dots, 10\}$ . By Theorem 4.1 we get that  $x\pi^i(x)\pi^j(x)\pi^k(x)$  is unavoidable in  $\Sigma_m$ , for  $m \geq \sigma + 1$ . Let now  $S' = S_\ell$  for some  $\ell = 1, \dots, 10$  be a set such that  $\max(S') = \sigma$ . Assume that there exists  $\ell \geq 5$  such that  $x\pi^i(x)\pi^j(x)\pi^k(x)$  is unavoidable in  $\Sigma_\ell$  and  $\ell < \sigma$ . Let  $A_0$  be the set containing all  $\delta_a$  parameters which are at most  $\ell$ , 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  $\delta \leq \ell$ . Clearly,  $A_0$  is either a strict subset of a set  $S = S_\ell$  for some  $\ell = 1, \dots, 10$  or  $A_0$  is incomparable to any of the sets of  $S$ . It cannot include any set  $S'' = S_\ell$  for some  $\ell = 1, \dots, 10$  as then  $\max(S'') < \max(S') = \min\{\max(S) \mid S = S_\ell \text{ for some } \ell = 1, \dots, 10\}$ , 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  $\Sigma_m$  such that the instances of  $p$  over  $\Sigma_m$  correspond only to  $\delta_{a^5}$  contained in  $A_0$ . This means that  $w$  avoids  $A_0$  in  $\Sigma_m$  with  $5 \leq m \leq \ell$ . So,  $p$  is avoidable in  $\Sigma_\ell$ , a contradiction.

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

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  $\Sigma_\sigma$ , or  $\beta = \sigma - 1$ , otherwise.

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

## 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  $\pi$  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

#### 4. Unary Patterns of Size Four with Morphic Permutations

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.

# 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

## 5. On Modelling the Avoidability of Patterns as CSP

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,

## 5.1. Introduction

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.

## 5. On Modelling the Avoidability of Patterns as CSP

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 minimzinc`
- `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, 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:



## 5.2. Checking the Avoidability of Formulas with Reversal

<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  $\Sigma_\ell$  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  $\ell$  of the alphabet  $\Sigma_\ell$ , 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  $x_1, \dots, x_k$  is encoded as a word over  $\{1, 2, \dots, k\} \cup \{-1, -2, \dots, -k\}$ , 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  
   inputFileNames pattern
```

## 5. On Modelling the Avoidability of Patterns as CSP

```
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 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

## 5.2. Checking the Avoidability of Formulas with Reversal

variable `label` ranges from 1 to `wordLengthnrVarsInPattern[p]`. Now, the length of the variable  $x_i$  is encoded in `label` using the formula

$$|x_i| = \left\lfloor \frac{\text{label}}{\text{wordLength}^{\text{nrVarsInPattern}[p]} - i} \right\rfloor \bmod \text{wordLength}.$$

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.

```

1 function int: length(int: var, int: wordLength, int: nrVars, int: label) =
2   if (ceil(label / pow(wordLength,nrVars - var) mod wordLength == 0) then
3     wordLength
4   else
5     ceil(label / pow(wordLength,nrVars - var) mod wordLength
6   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.

```

1 sum(k in 1..maxPatternLength where patterns[p, k] != 0)
2   (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

## 5. On Modelling the Avoidability of Patterns as CSP

`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.

```

1 exists (varOcc in 1..maxPatternLength where patterns[p,varOcc] != 0)(
2   exists (nextOcc in (varOcc + 1)..maxPatternLength where
3     abs(patterns[p,varOcc]) == abs(patterns[p, nextOcc]))(
4     let {
5       var int: varLength = length(abs(patterns[p,varOcc]), wordLength,nrVarsInPattern[
6         p], i),
7     } in
8     exists (l in 1..varLength)(
9       let {
10        var int: occInWord = start +
11          (if varOcc > 1 then
12            (sum(k in 1..(varOcc - 1))
13              (length(abs(patterns[p, k]), wordLength, nrVarsInPattern[p], i)))
14            else 0 endif),
15        var int: nextOccInWord = occInWord +
16          sum(k in varOcc..(nextOcc - 1))
17            (length(abs(patterns[p, k]), wordLength, nrVarsInPattern[p], i)),
18        var int: posFirst = occInWord + l - 1,
19        var int: posSecond =
20          (if (patterns[p, varOcc] == patterns[p, nextOcc]) then (nextOccInWord + l -
21            1)
22            else (nextOccInWord + varLength - l) endif)
23        } in
24        word[posFirst] != word[posSecond]
25      )
26    )
27  )

```

**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  $l^{\text{th}}$  symbol of the first image of

### 5.3. Checking the Avoidability of Patterns in the Abelian Sense

the variable (the one starting on `occInWord`), which is found on position `posFirst` in `word`, is different from the  $\iota^{th}$  symbol of the second image of the variable (the one starting on `next0ccInWord`), 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 `next0cc` 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  $x_1x_2x_2x_1r$  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  $\iota$  in the string that is supposed to be the image of the variable  $x$ ) whose number of occurrences in the

## 5. On Modelling the Avoidability of Patterns as CSP

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 `next0ccInWord`. 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  $1 \leq i \leq m$ ,  $x_i$  is a word variables and  $\pi_i$  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  $x, y$  are word variables and  $\pi$  and  $\sigma$  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  $\pi(\sigma(x))$ .

## 5.4. Checking Avoidability of Patterns under Permutations

Now, given a formula consisting of patterns under permutations (i.e., a set of patterns under permutations) and an alphabet  $\Sigma_k$ , 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  $\Sigma_k$ . 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  $\{x_1, x_2, x_3, \dots\}$  and the function variables are symbols from the set  $\{p_1, p_2, \dots\}$ . 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 antimorphism  $p_i$ , iterated  $k$  times, is applied to the word variable  $x$ . Let us exemplify this. Consider the pattern  $x_1 x_2 p_1^5(x_2) p_2^3(x_1)$  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  $i^{\text{th}}$  word variable occurring in the pattern (when read left to right) is mapped to the number  $i$ ; also, the  $i^{\text{th}}$  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

## 5. On Modelling the Avoidability of Patterns as CSP

$x$  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  $i^{\text{th}}$  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 (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[i][j]` gives us all the possible ways in which the  $j^{\text{th}}$  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  $j^{\text{th}}$  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 reshaped to the correct number by our program).

```
1 (1,2,3) , (1,2,3) , (1,2,3) ,  
2 (1,2,3) , (1,2,3) , (1,3,2) ,  
3 (1,2,3) , (1,2,3) , (2,1,3) ,  
4 (1,2,3) , (1,2,3) , (2,3,1) ,  
5 (1,2,3) , (1,2,3) , (3,1,2) ,  
6 (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



## 5.4. Checking Avoidability of Patterns under Permutations

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 `var0cc`, with `repetitions[p, var0cc, 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  $x_1x_2p_2(x_1)$  and  $x_1p_1(x_1)$

Then, for the variable occurring on `var0cc` in the considered pattern we find its next occurrence on position `next0cc` (i.e., `repetitions [p, var0cc, 3] == repetitions [p, next0cc, 3]`) under the same permutation (so we have `repetitions[p,var0cc,1]= repetitions[p,next0cc,1]`) or without any permutation applied on it (i.e., `repetitions [p, next0cc, 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

## 5. On Modelling the Avoidability of Patterns as CSP

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 + 1 - 1,  
8     var int: posSecond =  
9       (if (repetitions[p, varOcc, 4] == repetitions[p, nextOcc, 4]) then  
10        (nextOccInWord + 1 - 1)  
11        else (nextOccInWord + varLength - (l - 1) - 1) endif)  
12   } in  
13   permutations[p, z, varOcc, word[posFirst]] != permutations[p, z, nextOcc, word[  
14   posSecond]]  
)
```

**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

## 5.5. Generating Morphic Words

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 `morphicWord` 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

## 5. On Modelling the Avoidability of Patterns as CSP

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

```
1 constraint avoidPatterns(word, wordLength);
2 constraint avoidPatterns(finalWord, (sum(i in 1..morphicWordLength)
3     (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.

```
1 constraint
2   forall (i in 1..morphicWordLength)(
3     let {
4       var int : morphCharPos = morphicWord[i] + 1;
5       var int : posInWord = (sum (k in 1..(morphCharPos - 1)) (
6         morphicWordImagesLengths[k])) + 1,
7       var int : posInFinalWord = (sum (k in 1..(i - 1)) (morphicWordImagesLengths[
8         morphicWord[k] + 1])) + 1,
9     } in
10    forall (j in 1..morphicWordImagesLengths[morphCharPos])(
11      word[posInWord + j - 1] = finalWord[posInFinalWord + j - 1]
12    ));
```

**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 avoidability 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

## 5. On Modelling the Avoidability of Patterns as CSP

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.

# 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  $b(an)^2a$  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 ( $abbcacaca = ab^2(ca)^3$ ), the fractional powers, as the word indicates, may be rational numbers -  $banana = b(an)^{\frac{5}{2}}$ , 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  $b(anan)^{\frac{5}{4}}$  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  $\delta$ -power for a rational  $\delta$  ( $w$  is  $\delta$ -power free), since, for instance  $abcd a$  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

## 6. Repetition Avoidance in Products of Factors

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

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

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  $u = a$  and  $v = b$ ). 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  $i \geq 2$ , 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  $RT(2) = 2$  follows. Dejean [Dej72] proved  $RT(3) = \frac{7}{4}$  and conjectured  $RT(4) = \frac{7}{5}$  and  $RT(k) = \frac{k}{k-1}$  for  $k > 4$ . Mousavi and Shallit [MS13] extended this notion to  $RTC$  - the circular repetition threshold - and they proved  $RTC(2) = 4$  and  $RTC(3) = \frac{13}{4}$ . Moreover they established in [MS13] the notion  $RT_i(k)$  describing the repetition threshold over a  $k$ -letter alphabet and concatenating  $i \in \mathbb{N}$  factors. They proved  $RT_2(k) = RTC(k)$  for  $k = 2, 3$  and conjectured  $RT_2(k) = RTC(k)$  for  $k > 3$ .

In this work we act on this suggestions by investigating both  $RT_i(3)$  for  $i > 2$  and  $RT_2(k)$  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.

## 6. Repetition Avoidance in Products of Factors

**Theorem.** *For every  $k \geq 2$ , i.e. every alphabet with at least two letters, the repetition threshold with two factors and the circular repetition threshold are identical,  $RT_2(k) = RTC(k)$ .*

Mousavi and Shallit [MS13] have considered the binary alphabet and obtained that  $RT_i(2) = 2i$  for every  $i \geq 1$ . Our second result considers the ternary alphabet and gives the value of  $RT_i(3)$  for every  $i \geq 1$ . This extends the result of Dejean [Dej72] that  $RT_1(3) = \frac{7}{4}$  and the result of Mousavi and Shallit [MS13] that  $RT_2(3) = \frac{13}{4}$ .

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

$$\triangleright RT_i(3) = \frac{3i}{2} + \frac{1}{4} \text{ if } i = 1 \text{ or } i \text{ is even.}$$

$$\triangleright RT_i(3) = \frac{3i}{2} + \frac{1}{6} \text{ if } i \text{ is odd and } i \geq 3.$$

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  $(anan, a)$  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  $(an, a)$ .

**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  $\beta$  such that  $\beta > \delta$  (resp.  $\beta \geq \delta$ ). Moreover it is called  $(\delta^+, n)$ -free if it does not contain

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

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

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

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

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

▷  $\text{RT}(2) = 2$  (see [Dej72]);

▷  $\text{RT}(3) = \frac{7}{4}$  (see [Dej72])

▷  $\text{RT}(4) = \frac{7}{5}$  (see [Pan84])

▷  $\text{RT}(k) = \frac{k}{k-1}$ , for  $k \geq 5$  (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  $w \in \Sigma^\omega$  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  $\text{cexp}(w)$ . Let  $\text{RTC}(k)$  be the minimum of  $\text{cexp}(w)$  over every  $w \in \Sigma_k^\omega$ .

## 6. Repetition Avoidance in Products of Factors

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

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

### 6.3 Main results

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

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

*Proof.* The language of words in  $\Sigma_k^*$  avoiding circular repetitions of exponent at least  $\ell$  (or strictly greater than  $\ell$ ) 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],  $\text{pexp}_2(w) = \text{cexp}(w)$ . This implies that  $\text{RT}_2(k) = \text{RTC}(k)$ .  $\square$

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

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

*Proof.* Mousavi and Shallit [MS13] have proven that  $\text{RT}_2(3) = \frac{13}{4}$ , which settles the case  $i = 2$ . We have double checked their computation of the lower bound  $\text{RT}_2(3) \geq \frac{13}{4}$ . 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  $e \geq \frac{13}{4}$ . 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

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

This is the desired lower bound.  $\square$

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

*Proof.* Suppose that  $i \geq 3$  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

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

since this would produce the  $i$ -terms product  $(uv)^j u$  which is a repetition of the form  $t^x$  with exponent  $x = 3j + e \geq 3j + \frac{5}{3}$ . 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.  $\square$

*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  $\text{pexp}_i(w)$  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  $\{a, b, c\} = \{0, 1, 2\}$

## 6. Repetition Avoidance in Products of Factors

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  $i$ ,  $RT_i(3) \leq \frac{3i}{2} + \frac{1}{4}$  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  $\text{pexp}_i(w) \leq \frac{3i}{2} + \frac{1}{4}$ . The ternary morphic word used in [MS13] to obtain  $RT_2(3) \leq \frac{13}{4}$  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  $\Sigma_4$  by the following 45-uniform morphism satisfies  $\text{pexp}_i(u) \leq \frac{3i}{2} + \frac{1}{4}$ . Define the morphism by

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

Recall that a word is  $(\alpha^+, n)$ -free if it does not contain a repetition with period at least  $n$  and exponent strictly greater than  $\alpha$ . First, we check that such ternary images are  $(\frac{202}{135}^+, 36)$ -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  $\Sigma_4$  of length smaller than  $\frac{2 \cdot \frac{202}{135}}{\frac{202}{135} - 7} < 32$ . 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
- ▷  $9 \leq |t| \leq 35$ .

### 6.3. Main results

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  $\text{pexp}_i \leq \frac{3i}{2} + \frac{1}{4}$  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  $\text{pexp}_{i,t}$  as the length of a largest factor of  $t^\omega$  that is a  $i$ -terms product divided by  $|t|$ . We actually consider conjugacy classes, since if  $t'$  is a conjugate of  $t$ , then  $\text{pexp}_{i,t'} = \text{pexp}_{i,t}$ . Let  $t$  be such a factor. If, for some even  $j$ , we have  $\text{pexp}_{j+2,t} = \text{pexp}_{j,t} + 3$ , then it means that by appending a 2-terms product to a  $j$ -terms product that corresponds to a maximum factor of  $t^\omega$ , that can only add a cube of period  $|t|$ . This implies that for every  $k$ ,  $\text{pexp}_{j+2k,t} = \text{pexp}_{j,t} + 3k$ .

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  $\text{pexp}_{j+2,t} = \text{pexp}_{j,t} + 3$ . Thus we have  $\text{pexp}_i \leq \frac{3i}{2} + \frac{1}{4}$  in all cases.  $\square$

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

*Proof.* Let us show that the image of every  $\frac{7}{5}^+$ -free word over  $\Sigma_4$  by the following 514-uniform morphism satisfies  $\text{pexp}_i \leq \frac{3i}{2} + \frac{1}{6}$  for every odd  $i \geq 3$ . Consider the morphism given by

$0 \mapsto$  01020120210120102120210201210120102012021020121021201020121012  
 02102012102120210120102012102120102012021020121012010212021020  
 12102120102012021012010212021020121021202101201020121021201020  
 12101202102012102120210120102120210201210120102012021020121012  
 010212021020121021201020121012021020121021201020120210210210212  
 02102012101201020120210201210212021012010201210120210201210212  
 01020120210201210120102120210201210212010201210120210201210212  
 02101201021202102012101201020120210201210120102120210201210212  
 021012010201210212

## 6. Repetition Avoidance in Products of Factors

- 1  $\mapsto$  01020120210120102120210201210120102012021020121021201020121012  
02102012102120102012021020121012010212021020121021201020120210  
12010212021020121021202101201020121021201020121012021020121021  
20210120102120210201210120102012021020121021201020121012021020  
12101201021202102012102120210120102012102120102012021012010212  
02102012101201020120210201210120102120210201210212010201210120  
21020121021202101201021202102012101201020120210201210212021012  
01020121012021020121021201020120210201210120102120210201210212  
021012010201210212
- 2  $\mapsto$  01020120210120102120210201210120102012021020121021201020121012  
02102012101201021202102012102120102012021012010212021020121021  
20210120102012102120102012101202102012102120210120102102120210201  
21012010201202102012101201021202102012102120102012101202102012  
10212010201202101201021202102012102120210120102012102120102012  
02102012101201021202102012102120102012021012010212021020121012  
01020120210201210212021012010201210212010201210120210201210212  
02101201021202102012101201020120210201210120102120210201210212  
021012010201210212
- 3  $\mapsto$  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  $\Sigma_4$  of length smaller than  $\frac{2 \cdot \frac{3}{2}}{\frac{3}{2} - \frac{5}{7}} = 30$ . 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  $\text{pexp}_{j+2,t} = \text{pexp}_{j,t} + 3$ . Thus we have  $\text{pexp}_i \leq \frac{3i}{2} + \frac{1}{6}$



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  $i \geq 1$  is given by*

$$\triangleright RT_i(3) = \frac{3i}{2} + \frac{1}{4} \text{ if } i = 1 \text{ or } i \text{ is even.}$$

$$\triangleright RT_i(3) = \frac{3i}{2} + \frac{1}{6} \text{ if } i \text{ is odd and } i \geq 3.$$

## 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,  $RT_{i+1}(k) \geq RT_i(k) + 1$  for every  $i \geq 1$  and  $k \geq 2$ . Mousavi and Shalilit [MS13] verified that  $RT_2(4) \geq \frac{5}{2}$ , so that  $RT_i(4) \geq i + \frac{1}{2}$  for every  $i \geq 2$ . We conjecture that this is best possible, i.e., that  $RT_i(4) = i + \frac{1}{2}$  for every  $i \geq 2$ . However, a proof of an upper bound of the form  $RT_i(4) \leq i + c$  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  $RT_i(k) = RT(k) + i - 1$  is a Dejean word with severe restrictions on the types of repetitions that are allowed to appear.



# Unary patterns under permutations

Here is the list of all unavoidable sets of  $\delta_i$ s:

---

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

---

**Table A.1.** All unavoidable sets of  $\delta_i$ s

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

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

$$0 \rightarrow 0010020010020010, \quad 1 \rightarrow 0202242232322122, \quad 2 \rightarrow 0202212202212202$$

**Lemma 2.** Consider the infinite word:

$$\mathbf{h}_\delta = \delta(\mathbf{h}) = 001002001002001002022422323221220202212202212202001\dots$$

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

## A. Unary patterns under permutations

$$\begin{aligned} 0 &\rightarrow 000112220001122200011222, & 1 &\rightarrow 000113330001133300011, \\ 2 &\rightarrow 444000114440001122200111 \end{aligned}$$

**Lemma 3.** Consider the infinite word:

$$\mathbf{h}_\delta = \delta(\mathbf{h}) = 0001122200011222000112220001133300011333000114440001\dots$$

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

$$\begin{aligned} 0 &\rightarrow 001220112003110221002113, & 1 &\rightarrow 001220112003110221002113, \\ 2 &\rightarrow 001220112003110221002114 \end{aligned}$$

**Lemma 4.** Consider the infinite word:

$$\mathbf{h}_\delta = \delta(\mathbf{h}) = 001220112003110221002113001220112003110221002113001\dots$$

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

$$\begin{aligned} 0 &\rightarrow 0102202112133132232002033034434004022024424114122120020110133 \\ &13003022023323443422420020440411410, \\ 1 &\rightarrow 0102202112133132232002033034434004022024424114122120020110133 \\ &13003022023323443422420020440411410, \\ 2 &\rightarrow 0102202112133132232002033034434004022024424114122120020110133 \\ &130030220233234434224200204404 \end{aligned}$$

**Lemma 5.** Consider the infinite word:

$$\mathbf{h}_\delta = \delta(\mathbf{h}) = 010220211213313223200203303443400402202442411412212\dots$$

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

$$0 \rightarrow 0100131130330200232234334244202204004144101102002122131132332$$

02203003433404402002422414412112022,

1  $\rightarrow$  0100131130330200232234334244202204004144101102002122131132332  
02203003433404402002422414412112022,

2  $\rightarrow$  0100131130330200232234334244202204004144101102002122131132332  
022030034334044020024224

**Lemma 6.** Consider the infinite word:

$\mathbf{h}_\delta = \delta(\mathbf{h}) = 010013113033020023223433424420220400414410110200212 \dots$

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

0  $\rightarrow$  01022110030112100203311002011210020113122001022110030112100203  
311002011210020113122001022110030112100203311002011210020113122001  
0221100301121002033110020112100201131440,

1  $\rightarrow$  01022110030112100203311002011210020113122001022110030112100203  
311002011210020113122001022110030112100203311002011210020113122001  
0221100301121002033110020112100201131440,

2  $\rightarrow$  01022110030112100203311002011210020113122001022110030112100203  
311002011210020113122001022110030112100203311002011210020113122001  
022110030112100203311002011210020113300201121002011220030112100201  
1220030112100201122003011210020112202110020112100201131220

**Lemma 7.** Consider the infinite word:

$\mathbf{h}_\delta = \delta(\mathbf{h}) = 010221100301121002033110020112100201131220010221100 \dots$

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

0  $\rightarrow$  00012221000122201110222011102220111022201112000122210001222011

## A. Unary patterns under permutations

102220111022201110222011120001222100012220111022201110222011102220  
 111200021110222011102220111022201110222100012220111022201110222011  
 1022201112,

1  $\rightarrow$  00012221000122201110222011102220111022201112000122210001222011  
 102220111022201110222011120001222100012220111022201110222011102220  
 1113,

2  $\rightarrow$  00012221000122201110222011102220111022201112000122210001222011  
 102220111022201110222011120001222100012220111022201110222011102220  
 111200012221000122201110222011102220111022201113

**Lemma 8.** Consider the infinite word:

$$\mathbf{h}_\delta = \delta(\mathbf{h}) = 0001222100012220111022201110222011120001222\dots$$

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

0  $\rightarrow$  011202210112022313300112022101120223133001120221011202231330  
 01120221011202233001211002122001130331011202210112200313300114044  
 10112022101122003133001140441011202210112200313300114044101120221  
 0112200313300114244001140441,

1  $\rightarrow$  01120221011220031330011404410112022101122003133001140441011202  
 210112200313300114044101120221011220032330011303310112022101122003  
 133001140441011202210112200313300114044101120221011220031330011404  
 41011202210112200313300114244001140441,

2  $\rightarrow$  01120221011220031330011404410112022101122003133001140441011202  
 21011220031330011404410112022101122003233001130331011202210112200  
 31330011404410112022101122003133001140441011202210112200313300114  
 0441011202210112200313300114244001140441

**Lemma 9.** Consider the infinite word:

$$\mathbf{h}_\delta = \delta(\mathbf{h}) = 011202210112022313300112022101120223133001120221011\dots$$

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

$$\begin{aligned} 0 &\rightarrow 001001002001001002001001003, & 1 &\rightarrow 001001002001001002001001003, \\ 2 &\rightarrow 001001002001001002001001004 \end{aligned}$$

**Lemma 10.** Consider the infinite word:

$$\mathbf{h}_\delta = \delta(\mathbf{h}) = 001001002001001002001001003001001002001001002001001\dots$$

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

$$\begin{aligned} 0 &\rightarrow 001001010010010110110100100101001001001001011011010010010100 \\ &100101001001011011010010010100100101101101001001010010010100100101 \\ &10110100100101001001001001011011010010010010010100100101101101 \\ &011011010110110100100101, \end{aligned}$$

$$\begin{aligned} 1 &\rightarrow 001001010010010110110100100101001001001011011010010010100 \\ &1001010010010110110100100101001001011011010010010100100100101 \\ &101101001001010010010100100101101101001001010010010100100101101101 \\ &010110100100101001100101001001011011010010010100100101001001011011 \\ &010010010100100101001001011011010010010100100101101101001001010010 \\ &01010010010110110100100101001001001001011011010010010010010100 \\ &100101101101011011010010010100100101001001011011010010010100100101 \\ &001001011011010010010100100101001001011011010010010100100101101101 \\ &00100101001001010010010110110100100101001001001001011011010, \end{aligned}$$

$$\begin{aligned} 2 &\rightarrow 0100101001001010010010110110101101101001001010010010100100101 \\ &1011010010010100100100100101101101001001010010010100100101101101 \end{aligned}$$

**Lemma 11.** Consider the infinite word:

$$\mathbf{t}_\delta = \delta(\mathbf{t}) = 0010010100100101101101001001010010010100100101101101\dots$$

If  $\mathbf{h}_\delta$  contains an instance of the pattern  $x\pi^i(x)\pi^j(x)\pi^k(x)$  then this instance

## A. Unary patterns under permutations

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

0  $\rightarrow$  0221220010020033133001002001121100200100221220010020011211002  
001002232200100200112110020010022122001002001121100200100223220010  
020011211002001002212200100200112110020010022322001002001121100200  
100221220010020011211002001003313300200100221220010020011211002001  
002212200100200113110020010022122001002001121100200100221220010020  
011311002001002212200100200112110020010022122001002001131100200100  
2212200100200112110020010,

1  $\rightarrow$  0221220010020033233001002001121100200100221220010020011211002  
001002232200100200112110020010022122001002001121100200100223220010  
020011211002001002212200100200112110020010022322001002001121100200  
100221220010020011211002001003313300200100221220010020011211002001  
002212200100200113110020010022122001002001121100200100221220010020  
011311002001002212200100200112110020010022122001002001131100200100  
2212200100200112110020010,

2  $\rightarrow$  02212200100200331330010020011211002001002212200100200112110020  
0123220010020011211002001002212200100200112110020010022322001002001  
1211002001002212200100200112110020010022322001002001121100200100221  
2200100200112110020010033133002001002212200100200112110020010022122  
0010020011311002001002212200100200112110020010022122001002001131100  
2001002212200100200112110020010022122001002001131100200100221220010  
0200112110020010

**Lemma 12.** Consider the infinite word:

$$\mathbf{h}_\delta = \delta(\mathbf{h}) = 022122001002003313300100200112110020010022122001002\dots$$

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

0  $\rightarrow$  001001101101100100110110110010010011011001001001101100100100110  
11011001001101101100100100110110010010011011001001001101101100100110  
11011001001101101100100100110110010010011011011001001101101100100110



1101100100100110110010010011011001,  
 1  $\rightarrow$  0010011011011001001101101100100100110110010010011011001001001101100100100110  
 110110010011011011001001001101100100100110110010010011011011001001101101100100110  
 110110010011011011001001001101100100100110110110010011011011001001101101100100110  
 1101100100100110110010010011011001,  
 2  $\rightarrow$  001001101101100100110110110010010011011001001001101100100100110110010010011  
 011011001001101101100100100110110010010011011001001001101100100100110110110010011  
 011011001001101101100100100110110010010011011011001001101101100100110110110010011  
 01101100100100110110010010011011011

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

0  $\rightarrow$  022211222000110001100011000220002200011000110001100022000220001  
 100011000110002200022000110001100011000220022200011000110001100022  
 0002200011000110001100022000220001100011000110002200022000110001100  
 0110002220022200011000110001100022000220001100011000110002200022000  
 1100011000110002200022000110001100011000222002220001100011000110002  
 2000220001100011000110002200022000110001100011000220002200011000110  
 001100,  
 1  $\rightarrow$  022233222000110001100011000220002200011000110001100022000220001  
 100011000110002200022000110001100011000220022200011000110001100022  
 0002200011000110001100022000220001100011000110002200022000110001100  
 0110002220022200011000110001100022000220001100011000110002200022000  
 1100011000110002200022000110001100011000222002220001100011000110002  
 2000220001100011000110002200022000110001100011000220002200011000110  
 001100,  
 2  $\rightarrow$  022211222000110001100011000220002200011000110001100022000220001  
 100011000110002200022000110001100011000220022200011000110001100022  
 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  $\{\delta_1, \delta_2, \delta_3, \delta_4, \delta_5, \delta_7, \delta_8, \delta_{11}, \delta_{13}, \delta_{14}\}$ .

## A. Unary patterns under permutations

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

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

**Lemma 15.** Consider the infinite word:

$\mathbf{h}_\delta = \delta(\mathbf{h}) = 0001000100010002000100010003000100010004000\dots$

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

$0 \rightarrow 0104010301020103010401020104010301020103010401030102010401020103,$   
 $1 \rightarrow 0104010301020103010401020104010301040102010301020104010201030104,$   
 $2 \rightarrow 010301020103010401020104010301020103010401030102010401020103010$

**Lemma 16.** Consider the infinite word:

$\mathbf{h}_\delta = \delta(\mathbf{h}) = 010201030102010401020103010201050102010301020104010\dots$

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

$0 \rightarrow 20023321120023022011021001221021120023022011021001221021120023$   
 $022011021001221021120023022011021001331221,$   
 $1 \rightarrow 00130110220120021120122100130110220120021120122100130110220120$   
 $0211201221001301102201,$   
 $2 \rightarrow 20023321120023022011021001221021120023022011021001221021120023$   
 $022011021001221021123320021022011021001221021120021022011021001221$   
 $021120021022011021001221021120021022011021001331221$

**Lemma 17.** Consider the infinite word:

$$\mathbf{h}_\delta = \delta(\mathbf{h}) = 2002332112002302201102100122102112002302201102100122 \dots$$

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

$$0 \rightarrow 22002112002201100122100110220021123322300311331001102201100122 \\ 100110220021120022011001221001102200211233223003113310011033011001 \\ 22100110,$$

$$1 \rightarrow 22002112002201100122100110220021123322300311331001102201100122 \\ 100110220021120022011001221001102200211233223003113310011022011001 \\ 22100110,$$

$$2 \rightarrow 22002112002201100122100110220021123322300311331001102201100122 \\ 100110220021120022011001221001102200211233223003113310011044011001 \\ 22100110$$

**Lemma 18.** Consider the infinite word:

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

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

$$0 \rightarrow 210211210220210010210010211210211210211210220210010210010210010 \\ 2112102112102112102202100102100102100102112102112102112102202100102 \\ 1001021001021121021121021121022021001021001021001021121021121 \\ 0211210220210010210010210010211210211210220210010210010210010 \\ 2112102112102112102202100102100102100102112102112102202100102100102 \\ 1001021121021121021121022021001021001021001021121021121021121022021 \\ 0010210010210010211210211,$$

$$1 \rightarrow 210211210220210010210010211210211210211210220210010210010210010 \\ 21121021121021121022021001021001021001021121021121021121022021001021 \\ 00102100102112102112102112102202100102100102100102112102112102 \\ 11210220210010210010210010211210211210211210220210010210010210010211 \\ 21021121021121022021001021001021001021121021121022021001021001021001 \\ 21121021121021121022021001021001021001021121021121021121022021001021$$

## A. Unary patterns under permutations

0010210010211210211,

2  $\rightarrow$  2102112102202100102100102112102112102112102202100102100102100102  
 11210211210211210220210010210010210010211210211210211210220210010210  
 01021001021121021121022021001021001021001021121021121021121022021001  
 0210010210010211210211210211210220210010210010210010211210211.

**Lemma 19.** Consider the infinite word:

$\mathbf{h}_\delta = \delta(\mathbf{h}) = 2102112102202100102100102112102112102112102202100102 \dots$

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

0  $\rightarrow$  001001221331331221001001221,      1  $\rightarrow$  001001221331331221441441221,  
 2  $\rightarrow$  441441221331331221001001221

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

0  $\rightarrow$  1200012012012000,      1  $\rightarrow$  1200012012012000,      2  $\rightarrow$  1200013013331000

**Lemma 21.** Consider the infinite word:

$\mathbf{h}_\delta = \delta(\mathbf{h}) = 120001201201200012000120120120120001200013013331000120 \dots$

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

0  $\rightarrow$  0001221000122100012210002111022011022201102201110220110222011022  
 20110220111022011022201102201112,  
 1  $\rightarrow$  0001221000122100012210001222011022011102201102220110220111022011  
 10220110222011022011102201102221,  
 2  $\rightarrow$  000122100012210001221000211102201102220110220111022011022011022  
 20110220111022011022201102201113

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

**Lemma 23.** Consider the infinite word:

$\mathbf{h}_\delta = \delta(\mathbf{h}) = 0012221112221000111000122200011120002220001222111222 \dots$

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

**Lemma 24.** Consider the infinite word:

$\mathbf{h}_\delta = \delta(\mathbf{h}) = 012001021141400101200102114140020210020322424002021 \dots$

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

$0 \rightarrow 200100300200100300201220210020010020122021002001002012202100200$   
 $1002012202100200300100200300100200300100200301330310020010020122021$   
 $0020010020122021002001002012202100200100201223203313303100200100201$   
 $2202100200100201220210020010020122021002001002012232033133031002001$   
 $0020122021002001002012202100200100201220210020010020122320331330310$   
 $0200100201220210020010020122021002001002012202100200100201223220210$   
 $0200100201220210020010020122021002001002012202100200100203220210020$   
 $0100201220210020010020122021002001002012202100200100203220210020010$   
 $0201220210020010020122021002001002012202100200100203220210020010020$   
 $1220210020010020122021002001002012202100,$

$1 \rightarrow 200100300200100300201220210020010020122021002001002012202100200$   
 $1002012202100200300100200300100200300100200301330310020010020122021$   
 $0020010020122021002001002012202100200100201223203313303100200100201$   
 $2202100200100201220210020010020122021002001002012232033133031002001$   
 $0020122021002001002012202100200100201220210020010020122320331330310$

## A. Unary patterns under permutations

0200100201220210020010020122021002001002012202100200100201223220210  
0200100201220210020010020122021002001002012202100200100203220210020  
0100201220210020010020122021002001002012202100200100203220210020010  
0201220210020010020122021002001002012202100200100203220210020010020  
1220210020010020122021002001002012202100,

2  $\rightarrow$  200100300200100300201220210020010020122021002001002012202100200  
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  $\delta : \Sigma_3^* \rightarrow \Sigma_4^*$  be the morphism that is defined by  
 $0 \rightarrow 1021110122212000, \quad 1 \rightarrow 1021110122212000, \quad 2 \rightarrow 1021112133323000$

**Lemma 26.** Consider the infinite word:

$\mathbf{h}_\delta = \delta(\mathbf{h}) = 1021110122212000102111213332300010211121333230001021 \dots$

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  $\delta : \Sigma_3^* \rightarrow \Sigma_5^*$  be the morphism that is defined by  
 $0 \rightarrow 2121200010103334, \quad 1 \rightarrow 3434300010101022, \quad 2 \rightarrow 2121200010101022$

**Lemma 27.** Consider the infinite word:

$\mathbf{h}_\delta = \delta(\mathbf{h}) = 212120001010333434343000101033343434300010101022212 \dots$

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  $\delta : \Sigma_3^* \rightarrow \Sigma_4^*$  be the morphism that is defined by  
 $0 \rightarrow 1022200011121000, \quad 1 \rightarrow 1022200011121000, \quad 2 \rightarrow 1022200033323000$

**Lemma 28.** Consider the infinite word:

$\mathbf{h}_\delta = \delta(\mathbf{h}) = 1022200011121000102220001112100010222000333230001022 \dots$

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

**Lemma 29.** Consider the infinite word:

$\mathbf{h}_\delta = \delta(\mathbf{h}) = 032322402001202004242210200120200323224020012020032 \dots$

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

$0 \rightarrow 0010021011201002001002101120100200100210112010020010031011201002$   
 $00100210112010020010021011201002001002101120100200100310112010020010$   
 $02101120100200100210112010020010021011201002001003101120100200100210$   
 $11201002001002101120100200100210112010020012022102001002001202210200$   
 $10020012022102001002001202212201211011201002,$

$1 \rightarrow 0010021011201002001002101120100200100210112010020010031011201002$   
 $00100210112010020010021011201002001002101120100200100310112010020010$   
 $02101120100200100210112010020010021011201002001003101120100200100210$   
 $1120100200100210112010020010021011301002,$

$2 \rightarrow 0010021011201002001002101120100200100210112010020010021011301002$   
 $00100210112010020010021011201002001002101120100200100210113010020010$   
 $02101120100200100210112010020010021011201002001002101130100200100310$   
 $11201002$

**Lemma 30.** Consider the infinite word:

$\mathbf{h}_\delta = \delta(\mathbf{h}) = 0010021011201002001002101120100200100210112010020010031011 \dots$

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

$0 \rightarrow 1202221011102000,$       $1 \rightarrow 1202221011102000,$       $2 \rightarrow 1202223133301000$

**Lemma 31.** Consider the infinite word:

$\mathbf{h}_\delta = \delta(\mathbf{h}) = 1202221011102000120222101110200012022231333010001202 \dots$

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  $\{\delta_1, \delta_3, \delta_5, \delta_7, \delta_{10}, \delta_{11}, \delta_{12}, \delta_{13}\}$ .

## A. Unary patterns under permutations

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

**Lemma 32.** Consider the infinite word:

$\mathbf{h}_\delta = \delta(\mathbf{h}) = 1211102022201000121110202220100012111323222010001211\dots$

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

**Lemma 33.** Consider the infinite word:

$\mathbf{h}_\delta = \delta(\mathbf{h}) = 11100012111020001110001211102000111000232220100011100\dots$

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  $\delta : \Sigma_3^* \rightarrow \Sigma_4^*$  be the morphism that is defined by  
 $0 \rightarrow 0001211102000122201000121110200012220100012111020001222010001211$   
 $10200012220300012111022201,$

$1 \rightarrow 0001211102000122201000121110200012220100012111020001222010001211$   
 $10222010001211102000122201000121110200012220100012111020001222010001$   
 $21110222010001211102000122201000121110200012220100012111020001222010$   
 $0012111022201,$

$2 \rightarrow 0001211102000122201000121110200012220100012111020001222030001211$   
 $1022201$

**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  $\delta : \Sigma_3^* \rightarrow \Sigma_4^*$  be the morphism that is defined by  
 $0 \rightarrow 010102210020201120010102210020201120010102210020201120010102210$   
 $02110101220112001010221002020112001010221002020112001010221002020112$   
 $00101022100211010122011200101022100202011200101022100202011200101022$   
 $10020201120010102210021101012201120010102210020201120010102210020201$   
 $12001010221002020112001010221002330301120,$





## A. Unary patterns under permutations

110122201110122201110122210001022201110122201110222021110122201110122  
 201110122201110122210001222021110122201110122201110122201110122210001  
 022201110122201110222021110122201110122201110122201110122210001022201  
 110122201110222021110122201110122201110122201110122210001022201110122  
 20111022202111012220111012220111012220111012221,

1  $\rightarrow$  00010222011101222011120001022201110122201110122201110122201110222  
 021110122201110122201110122201110122210001022201110122201110222021110  
 122201110122201110122201110122210001022201110122201110222021110122201  
 110122201110122201110122210001022201110122201110222021110122201110122  
 201110122201110122210001222021110122201110122201110122201110122210001  
 022201110122201110222021110122201110122201110122201110122210001022201  
 110122201110222021110122201110122201110122201110122210001022201110122  
 20111022202111012220111012220111012220111012221,

2  $\rightarrow$  00010222011101222011130001022201110122201110122201110122201110222  
 021110122201110122201110122201110122210001022201110122201110222021110  
 122201110122201110122201110122210001022201110122201110222021110122201  
 110122201110122201110122210001022201110122201110222021110122201110122  
 201110122201110122210001222021110122201110122201110122201110122210001  
 022201110122201110222021110122201110122201110122201110122210001022201  
 110122201110222021110122201110122201110122201110122210001022201110122  
 20111022202111012220111012220111012220111012221

**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  $\{\delta_1, \delta_4, \delta_8, \delta_{10}, \delta_{11}, \delta_{12}, \delta_{13}, \delta_{14}\}$ .

# 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 to check these sets can be found at <http://media.informatik.uni-kiel.de/zs/patterns.zip>. They are implemented in Ruby.

## Codes for Chapter 4

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

---

```
1 require "set"
2 def permutation2(x1,x2,x3,x4)
3   t = x1.length
4   arr = Array.new(t,0)
5   total = 0
6   cycles = t
7   while cycles > 0 do
8     j = 0
9     while (arr[j] != 0 && j<t) do
10      j += 1
11    end
12
13    if j<t
14      arr[j]= t + 1 - cycles
15      total += 1
16    end
17
18    s_1 = Set.new([x1[j],x2[j ],x3[j ],x4[j ]])
19    temp = true
```

## B. Code listings

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

```

65         s_2 = Set.new([x1[i],x2[i ],x3[i ],x4[i ]])
66         s_3 = Set.new
67         s_3 = s_1 & s_2
68         if (s_3.empty? == false)
69             temp = true
70             s_1 = s_1 | s_2
71             arr[i] = t + 1 - cycles
72         end
73     end
74 }
75 end
76 cycles = cycles-1
77 end
78
79 if (total>1)
80 end
81
82 map_cycles = Hash.new
83 0.upto(t-1) { |i|
84     if (!(map_cycles.key?(x1[i])))
85         map_cycles[x1[i]] = arr[i]
86     end
87 }
88
89 0.upto(t-1) { |i|
90     if (!(map_cycles.key?(x2[i])))
91         map_cycles[x2[i]] = arr[i]
92     end
93 }
94 0.upto(t-1) { |i|
95     if (!(map_cycles.key?(x3[i])))
96         map_cycles[x3[i]] = arr[i]
97     end
98 }
99 0.upto(t-1) { |i|
100     if (!(map_cycles.key?(x4[i])))
101         map_cycles[x4[i]] = arr[i]
102     end
103 }
104
105 #defining the mappings
106 maps = []
107
108 mapping12 = Hash.new
109 0.upto(t-1) { |pos|

```

## B. Code listings

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

```

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

```

## B. Code listings

```
200     #case1: the letter at from[pos] isn't mapped already
201     if (!(mapping42.key?(x4[pos])))
202         mapping42[x4[pos]] = x2[pos]
203     end
204 }
205 maps.push(mapping42)
206
207 mapping43 = Hash.new
208 0.upto(t-1) { |pos|
209     #case1: the letter at from[pos] isn't mapped already
210     if (!(mapping43.key?(x4[pos])))
211         mapping43[x4[pos]] = x3[pos]
212     end
213 }
214 maps.push(mapping43)
215
216 strings = []
217 strings.push(x1)
218 strings.push(x2)
219 strings.push(x3)
220 strings.push(x4)
221
222 alph = Set.new
223 0.upto(t-1){ |i|
224     0.upto(3){ |j|
225         alph.add(strings[j][i])
226     }
227 }
228
229 temp=true
230 0.upto(11){ |i|
231     0.upto(11) { |j|
232         if i!=j
233             alph.each do |x|
234                 if maps[i].key?(x) && maps[j].key?(x)
235                     if maps[i][x]==maps[j][x]
236                         alph.each do |y|
237                             if maps[i].key?(y) && maps[j].key?(y) && map_cycles[y]==
238                                 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
246                 end
247             end
248         end
249     end
250 }
251 end
```



```

244         end
245     end
246 end
247 end
248 }
249 }
250 return temp
251 end
252
253 #counting different symbols in a word
254 def count_letters(str)
255     count = Hash.new(0)
256     str.delete(" ").each_char { | letter | count[ letter ]+=1}
257     Hash[count.sort_by { | k,v | v}.reverse]
258 end
259
260 #factorizes w into 4 parts of equal length for w of length 4k
261 def factorize(w)
262     fourth = w.length/4
263     return w[0, fourth], w[fourth, fourth], w[(-2)*fourth, fourth], w[-fourth, fourth]
264 end
265
266 #checks whether a morphism exists that maps from to
267 def morphism_exists?(from, to)
268     #lengths have to be equal
269     return false if (from.length != to.length)
270     mapping = Hash.new
271     0.upto(from.length-1) { | pos |
272         #case1: the letter at from[pos] isn't mapped already
273         if (!(mapping.key?(from[pos])))
274             #return false if something else is mapped to the target letter already
275             #(it's not injective then)
276             return false if (mapping.has_value?(to[pos]))
277             #otherwise map from[pos] -> to[pos]
278             mapping[from[pos]] = to[pos]
279         #case2: the letter is mapped already. check if it's mapped consistently
280         else
281             return false if (mapping[from[pos]] != to[pos])
282         end
283     }
284     #if no inconsistency is found
285     return true
286 end
287
288 #short function that checks if  $x_1=x_2$  and  $x_1$  can be mapped to  $f$  by a bijective morphism

```

## B. Code listings

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

```

331 end
332
333 #we assume that  $|w| = 4k$  for some integer  $k$ 
334 def checkxxxx(w)
335   x1, x2, x3, x4 = factorize(w)
336   (!equal?(x1, x2, x3, x4) && !morphisms?(x1, x2, x3, x4))
337 end
338
339 def checkxxfx(w)
340   x1, x2, f1, x3 = factorize(w)
341   (pattern?(x1, x2, f1) && pattern?(x1, x3, f1) && pattern?(x2, x3, f1) && !equal2?(x1,
342     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,
347     f1))
348 end
349 def checkxf1xf2(w)
350   x1, f1, x2, f2 = factorize(w)
351   (pattern?(x1, x2, f1) && pattern?(x1, x2, f2) && !equal2?(x1, f1) && !equal2?(f1, f2)
352     && !equal2?(x1, f2) && !morphisms?(x1, f1, x2, f2))
353 end
354 def checkfxxx(w)
355   f, x1, x2, x3 = factorize(w)
356   (pattern?(x1, x2, f) && pattern?(x1, x3, f) && pattern?(x2, x3, f) && !equal2?(x1, f)
357     && !morphisms?(f, x1, x2, x3))
358 end
359 def checkxf1f2f3(w)
360   x, f1, f2, f3 = factorize(w)
361   (!equal2?(x, f1) && !equal2?(x, f2) && !equal2?(x, f3) && !equal2?(f1, f2) && !equal2
362     ?(f1, f3) && !equal2?(f2, f3) && !morphisms?(x, f1, f2, f3))
363 end
364 def checkxxff(w)
365   x1, x2, f1, f2 = factorize(w)
366   (pattern?(x1, x2, f1) && pattern?(x1, x2, f2) && equal2?(f1, f2) && !equal2?(x1, f1))
367 end
368 def checkxxf1f2(w)
369   x1, x2, f1, f2 = factorize(w)
370   (pattern?(x1, x2, f1) && pattern?(x1, x2, f2) && !equal2?(x1, f1) && !equal2?(x1, f2)
371     && !equal2?(f1, f2) && !morphisms?(x1, x2, f1, f2))
372 end

```

## B. Code listings

```
370 def checkx1fx(w)
371   x1, f1, f2, x2 = factorize(w)
372   (pattern?(x1, x2, f1) && pattern?(x1, x2, f2) && pattern?(f1, f2, x1) && !equal2?(x1,
    f1))
373 end
374
375 def checkx1f1f2f1(w)
376   x1, f1, f2, f3 = factorize(w)
377   (pattern?(f1, f3, x1) && pattern?(f1, f3, f2) && !equal2?(x1, f1) && !equal2?(x1,f2)
    && !equal2?(f1, f2) && !morphisms?(x1, f1, f2, f3))
378 end
379
380 def checkx1f1f2(w)
381   x1, x2, f1, f2 = factorize(w)
382   (pattern?(x2, f1, x1) && morphism_exists?(x1, f1) && morphism_exists?(x1, f2) && !
    equal2?(x1,f1) && !equal2?(x1, f2) && !equal2?(f1, f2))
383 end
384
385 def checkx1x2ff(w)
386   x1, x2, f1, f2 = factorize(w)
387   (pattern?(f1, f2, x1) && pattern?(f1, f2, x2) && !equal2?(x1, f1) && !equal2?(x2, f1)
    && !equal2?(x1,x2) && !morphisms?(x1, x2, f1, f2))
388 end
389
390 def checkx1f1f2f2(w)
391   x1, f1, f2, f3 = factorize(w)
392   (pattern?(f2, f3, x1) && pattern?(f2, f3, f1) && !equal2?(x1, f1) && !equal2?(x1, f2)
    && !equal2?(f1, f2) && !morphisms?(x1, f1, f2, f3))
393 end
394
395 def checkxxxf(w)
396   x1, x2, x3, f = factorize(w)
397   (pattern?(x1, x2, f) && pattern?(x1, x3, f) && pattern?(x2, x3, f) && !equal2?(x1,f)
    && !morphisms?(x1, x2, x3, f))
398 end
399
400 #checks if the word w contains a factor of the form x1f1f2f3, xx1f2, x1xf2, xxxf, xxfx
    simultaneously at any position.
401 def checkfxxx_xxfx_x1f1f2f1_x1x2ff(w)
402   #factorlength
403   (1..(w.length/4).ceil).each { |d|
404     #starting position
405     (0..w.length - 4*d).each { |i|
406       u = w[i..i + d - 1]
407       fu = w[i+d..i+2*d-1]
```

```

408     gu = w[i+2*d..i+3*d-1]
409     hu = w[i+3*d..i+4*d-1]
410     if checkxxx(u+fu+gu+hu)
411         #if permutation(u,gu,hu)
412             puts "The word #{w} contains a factor of the form xxx here: #{u} |#{
fu} |#{gu} |#{hu}"
413             #end
414             return true
415         end
416         if checkxf1f2f3(u+fu+gu+hu)
417             if permutation(u,gu,hu)
418                 puts "The word #{w} contains a factor of the form xf1f2f3 here: #{u} |#{fu
} |#{gu} |#{hu}"
419                 permutation2(u,gu,hu)
420             end
421         return true
422         end
423     end
424     if checkxxf1f2(u+fu+gu+hu)
425         if permutation(u,gu,hu)
426             puts "The word #{w} contains a factor of the form xxf1f2 here: #{u} |#{fu
} |#{gu} |#{hu}"
427             permutation2(u,gu,hu)
428         end
429     return true
430     end
431     end
432     if checkxf1xf2(u+fu+gu+hu)
433         if permutation(u,gu,hu)
434             puts "The word #{w} contains a factor of the form xf1xf2 here: #{u} |#{fu
} |#{gu} |#{hu}"
435             permutation2(u,gu,hu)
436         end
437     return true
438     end
439     end
440     if checkxxxf(u+fu+gu+hu)
441         if permutation(u,gu,hu)
442             puts "The word #{w} contains a factor of the form xxxf here: #{u} |#{fu} |#{
gu} |#{hu}"
443             permutation2(u,gu,hu)
444         end
445     return true
446     end
447     end

```

## B. Code listings

```
448         if checkxxfx(u+fu+gu+hu)
449             if permutation(u,fu,gu,hu)
450                 puts "The word #{w} contains a factor of the form xxfx here: #{u}|#{fu}|#{
gu}|#{hu}"
451                 permutation2(u,fu,gu,hu)
452
453 return true
454     end
455     end
456
457 }
458 }
459 return false
460 end
461
462 #try to produce an infinite word that avoids patterns above by backtracking
463 puts "Trying to generate a word that avoids patterns of size 4:"
464 def backtrack1(w)
465     if (!checkfxx_xxfx_xf1f2f1_x1x2ff(w))
466         $let = count_letters(w).keys.count
467         0.upto($let){ |temp|
468             backtrack1(w + temp.to_s)
469         }
470     end
471 end
472
473 #without loss of generality the word starts with 0
474 backtrack1("0")
```

---

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**

---

```
1 require "set"
2 def permutation(x1,x2,x3,x4)
3     t = x1.length
4     arr = Array.new(t,0)
5     total = 0
6     cycles = t
7     while cycles > 0 do
8         j = 0
9         while (arr[j] != 0 && j<t) do
10             j += 1
11         end
12
```

```

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

```

## B. Code listings

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



```

103
104 mapping23 = Hash.new
105 0.upto(t-1) { |pos|
106   #case1: the letter at from[pos] isn't mapped already
107   if (!(mapping23.key?(x2[pos])))
108     mapping23[x2[pos]] = x3[pos]
109   end
110 }
111 maps.push(mapping23)
112
113 mapping24 = Hash.new
114 0.upto(t-1) { |pos|
115   #case1: the letter at from[pos] isn't mapped already
116   if (!(mapping24.key?(x2[pos])))
117     mapping24[x2[pos]] = x4[pos]
118   end
119 }
120 maps.push(mapping24)
121
122 mapping31 = Hash.new
123 0.upto(t-1) { |pos|
124   #case1: the letter at from[pos] isn't mapped already
125   if (!(mapping31.key?(x3[pos])))
126     mapping31[x3[pos]] = x1[pos]
127   end
128 }
129 maps.push(mapping31)
130
131 mapping32 = Hash.new
132 0.upto(t-1) { |pos|
133   #case1: the letter at from[pos] isn't mapped already
134   if (!(mapping32.key?(x3[pos])))
135     mapping32[x3[pos]] = x2[pos]
136   end
137 }
138 maps.push(mapping32)
139
140 mapping34 = Hash.new
141 0.upto(t-1) { |pos|
142   #case1: the letter at from[pos] isn't mapped already
143   if (!(mapping34.key?(x3[pos])))
144     mapping34[x3[pos]] = x4[pos]
145   end
146 }
147 maps.push(mapping34)

```

## B. Code listings

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

```

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

```

## B. Code listings

```
237   }
238   #if no inconsistency is found
239   return true
240 end
241
242 #short function that checks if  $x_1=x_2$  and  $x_1$  can be mapped to  $f$  by a bijective morphism
243 def pattern?(x1, x2, f)
244   if (x1 != x2 || !morphism_exists?(x1, f))
245     return false
246   else
247     return true
248   end
249 end
250
251 #check whether variables are equal to each others or not (requiring not equal variables)
252 def equal?(x1, x2, x3, x4)
253   if ((x1 == x2) && (x1 == x3) && (x1 == x4) && (x2 == x3) && (x2 == x4) && (x3 == x4
254     ))
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 #check whether variables are equal to each others or not (requiring not equal variables)
280 def morphisms?(x1, x2, x3, x4)
```

```

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

```

## B. Code listings

```

    !morphisms?(x1, x2, f1, f2))
319 end
320 def checkxf1f2(w)
321   x1, x2, f1, f2 = factorize (w)
322   (pattern?(x1, x2, f1) && pattern?(x1, x2, f2) && !equal2?(x1, f1) && !equal2?(x1, f2)
    && !equal2?(f1,f2) && !morphisms?(x1, x2, f1, f2))
323 end
324
325 def checkxffx(w)
326   x1, f1, f2, x2 = factorize (w)
327   (pattern?(x1, x2, f1) && pattern?(x1, x2, f2) && pattern?(f1, f2, x1) && !equal2?(x1,
    f1) && !morphisms?(x1, f1, f2, x2))
328 end
329
330 def checkxf1f2f1(w)
331   x1, f1, f2, f3 = factorize (w)
332   (pattern?(f1, f3, x1) && pattern?(f1, f3, f2) && !equal2?(x1, f1) && !equal2?(x1,f2)
    && !equal2?(f1, f2) && !morphisms?(x1, f1, f2, f3))
333 end
334
335 def checkxf1f1f2(w)
336   x, f1, f2, f3 = factorize (w)
337   (pattern?(f1, f2, x) && pattern?(f1, f2, f3) && !equal2?(x, f1) && !equal2?(x, f3) &&
    !equal2?(f1, f3) && !morphisms?(x, f1, f2, f3))
338 end
339
340 def checkx1x2ff(w)
341   x1, x2, f1, f2 = factorize (w)
342   (pattern?(f1, f2, x1) && pattern?(f1, f2, x2) && !equal2?(x1, f1) && !equal2?(x2, f1)
    && !equal2?(x1,x2) && !morphisms?(x1, x2, f1, f2))
343 end
344
345 def checkxf1f2f2(w)
346   x1, f1, f2, f3 = factorize (w)
347   (pattern?(f2, f3, x1) && pattern?(f2, f3, f1) && !equal2?(x1, f1) && !equal2?(x1, f2)
    && !equal2?(f1, f2) && !morphisms?(x1, f1, f2, f3))
348 end
349
350 def checkxxx(f)(w)
351   x1, x2, x3, f = factorize (w)
352   (pattern?(x1, x2, f) && pattern?(x1, x3, f) && pattern?(x2, x3, f) && !equal2?(x1,f)
    && !morphisms?(x1, x2, x3, f))
353 end
354
355 def checkxfx(w)
```

```

356   x1, f, x2, x3 = factorize(w)
357   (pattern?(x1, x2, f) && pattern?(x1, x3, f) && pattern?(x2, x3, f) && !equal2?(x1,f)
    && !morphisms?(x1, f, x2, x3))
358 end
359
360 def checkxf1f2x(w)
361   x1, f1, f2, x2 = factorize(w)
362   (pattern?(x1, x2, f1) && pattern?(x1, x2, f2) && !equal2?(x1,f1) && !equal2?(x1,f2)
    && !equal2?(f1,f2) && !morphisms?(x1, f1, f2, x2))
363 end
364
365 #checks if the word w contains a factor of the form xxxx or xxf1f2 or x1xf2 or xf1f2f1 or
    xf1f2f2 or xxxf or xxfx or fxxx or xxff or xxfx or xfff or xf1f1f2 or xf1f2x.
366 def check_Functions(w)
367   #factorlength
368   (1..(w.length/4).ceil).each { |d|
369     #starting position
370     (0..w.length - 4*d).each { |i|
371       u = w[i..i + d - 1]
372       fu = w[i+d..i+2*d-1]
373       gu = w[i+2*d..i+3*d-1]
374       hu = w[i+3*d..i+4*d-1]
375       if checkxxxx(u+fu+gu+hu)
376         puts "The word #{w} contains a factor of the form xxxx here: #{u}|#{fu}|#{
gu}|#{hu}"
377
378         return true
379       end
380       if checkxxf1f2(u+fu+gu+hu)
381         if permutation(u,fu,gu,hu)
382
383           puts "The word #{w} contains a factor of the form xxf1f2 here: #{u}|#{fu
}|#{gu}|#{hu}"
384         return true
385       end
386       end
387       if checkxf1xf2(u+fu+gu+hu)
388         if permutation(u,fu,gu,hu)
389
390           puts "The word #{w} contains a factor of the form x1xf2 here: #{u}|#{fu
}|#{gu}|#{hu}"
391         return true
392       end
393       end
394       if checkxf1f2f1(u+fu+gu+hu)

```

## B. Code listings

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



```

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

```

## B. Code listings

```
475  
476 puts "checking if the word contains any patterns"  
477 check_Functions("012304120341023401324031243021340123402134201324012304120341")
```

---

### Codes for Chapter 5

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

```
1 sigma = 5;  
2 wordLength = 5;  
3 nrPatterns = 1;  
4 maxPatternLength = 4;  
5  
6 patterns = array2d(1..nrPatterns, 1..maxPatternLength, [  
7   % x1x2x2x1r  
8   1, 2, 2, -1,  
9   ]);  
10  
11 nrVarsInPattern = array1d(1..nrPatterns, [  
12   2,  
13   ]);  
14  
15 morphicWordLength = 10;  
16 morphicWord = array1d(1..morphicWordLength, [  
17   0, 1, 1, 0, 1, 0, 0, 1, 1, 0,  
18   ]);  
19  
20 numberOfMorphicWordImages = 2;  
21 morphicWordImagesLengths = array1d(1..numberOfMorphicWordImages, [  
22   3, 2,  
23   ]);
```

---

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**

```

1 % input parameters
2 int: sigma;
3 int: wordLength;
4 int: nrPatterns;
5 int: maxPatternLength;
6 array [1..nrPatterns, 1..maxPatternLength] of int: patterns;
7 array [1..nrPatterns] of int: nrVarsInPattern;
8 int: morphicWordLength;
9 array [1..morphicWordLength] of int: morphicWord;
10 int: numberOfMorphicWordImages;
11 array [1..numberOfMorphicWordImages] of int: morphicWordImagesLengths;
12
13 % output decision variables
14 array [1..wordLength] of var 1..sigma: word;
15 array [1..(sum(i in 1..morphicWordLength) (morphicWordImagesLengths[morphicWord[i]
    + 1]))] of var 1..sigma: finalWord;
16
17 % returns length of variable indicated by 'variableIndex' based on 'labelIndex'
18 function int: length(int: variableIndex, int: wordLength, int: nrVars, int: label) =
19   if (ceil (label / pow(wordLength, (nrVars - variableIndex + 1) - 1)) mod wordLength
    == 0) then
20     wordLength
21   else
22     ceil (label / pow(wordLength, (nrVars - variableIndex + 1) - 1)) mod wordLength
23   endif;
24
25 predicate avoidPatterns(array[int] of var int: word, int: length) =
26   % for each pattern
27   forall (p in 1..nrPatterns)
28   (
29     % for each position in word
30     forall (start in 1..length)
31     (
32       let {
33         % maximum number of labels based on number of variables
34         int: maxNumberOfLabels = pow(length, nrVarsInPattern[p]);
35       } in
36       % move through all labels
37       forall (i in 1..maxNumberOfLabels)
38       (

```

## B. Code listings

```
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 + 1 - 1,
58     var int: posSecond =
59     (if (patterns[p, varOcc] == patterns[p, nextOcc]) then
60     (nextOccInWord + 1 - 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
75     constraint avoidPatterns(word, wordLength);
76     constraint avoidPatterns(finalWord, (sum(i in 1..morphicWordLength) (
morphicWordImageLengths[morphicWord[i] + 1]));)
77     constraint
78     forall (i in 1..morphicWordLength)
```

```

79 (
80   let {
81     var int : importantIndexInWord = (sum (k in 1..(morphicWord[i])) (
82       morphicWordImagesLengths[k])) + 1,
83     var int : importantIndexInFinalWord = (sum (k in 1..(i - 1)) (
84       morphicWordImagesLengths[morphicWord[k] + 1])) + 1,
85   } in
86   forall (j in 1..morphicWordImagesLengths[morphicWord[i] + 1])
87     (
88       word[importantIndexInWord + j - 1] = finalWord[importantIndexInFinalWord + j -
89         1]
90     )
91 );
92 solve::int_search(finalWord, input_order, indomain_min, complete) satisfy;
93 output[show(finalWord)];

```

---

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**

```

1 sigma = 4;
2 wordLength = 10;
3 nrPatterns = 1;
4 maxPatternLength = 2;
5
6 patterns = array2d(1..nrPatterns, 1..maxPatternLength, [
7   % xx
8   1, 1,
9 ]);
10
11 nrVarsInPattern = array1d(1..nrPatterns, [
12   1,
13 ]);
14
15 morphicWordLength = 2;
16 morphicWord = array1d(1..morphicWordLength, [ 0, 1, ]);
17
18 numberOfMorphicWordImages = 2;
19 morphicWordImagesLengths = array1d(1..numberOfMorphicWordImages, [5, 5, ]);

```

---

## B. Code listings

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

```
1 % input parameters
2 int: wordLength;
3 int: sigma;
4 int: nrPatterns;
5 int: maxPatternLength;
6 int: morphicWordLength;
7 int: numberOfMorphicWordImages;
8 array [1..morphicWordLength] of int: morphicWord;
9 array [1..numberOfMorphicWordImages] of int: morphicWordImagesLengths;
10 array [1..nrPatterns, 1..maxPatternLength] of int: patterns;
11 array [1..nrPatterns] of int: nrVarsInPattern;
12
13 % output decision variables
14 array [1..wordLength] of var 1..sigma: word;
15 array [1..(sum(i in 1..morphicWordLength) (morphicWordImagesLengths[morphicWord[i]
    + 1]))] of var 1..sigma: finalWord;
16
17 % returns length of variable indicated by 'variableIndex' based on 'labelIndex'
18 function int: length(int: variableIndex, int: wordLength, int: nrVars, int: label) =
19   if (ceil (label / pow(wordLength, (nrVars - variableIndex + 1) - 1)) mod wordLength
    == 0) then
20     wordLength
21   else
22     ceil (label / pow(wordLength, (nrVars - variableIndex + 1) - 1)) mod wordLength
23   endif;
24
25 predicate avoidPatterns(array[int] of var int: word, int: length) =
26   % for each pattern
27   forall (p in 1..nrPatterns)
28   (
29     % for each position in word
30     forall (start in 1..length)
31     (
32       let {
33         % maximum number of labels based on number of variables
34         int: maxNumberOfLabels = pow(length, nrVarsInPattern[p]);
35       } in
36       % move through all labels
37       forall (i in 1..maxNumberOfLabels)
38       (
```

```

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 + 1 - 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) (
morphicWordImageLengths[morphicWord[i] + 1]]));
74 constraint
75 forall (i in 1..morphicWordLength)

```

## B. Code listings

```
76 (
77   let {
78     var int: firstPos = (sum (k in 1..( morphicWord[i])) (morphicWordImagesLengths[
79       k])) + 1,
80     var int: importantIndexInFinalWord = (sum (k in 1..(i - 1)) (
81       morphicWordImagesLengths[morphicWord[k] + 1])) + 1,
82   } in
83   forall (j in 1..morphicWordImagesLengths[morphicWord[i] + 1])
84     (
85       word[firstPos + j - 1] = finalWord[importantIndexInFinalWord + j - 1]
86     )
87 );
88 solve::int_search(finalWord, input_order, indomain_min, complete) satisfy;
89 output[show(finalWord)];
```

---

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**

```
1 sigma = 3;
2 wordLength = 10;
3 nrPatterns = 1;
4 nrPermutations = 6;
5 maxPatternLength = 3;
6
7 repetitions = array3d(1..nrPatterns, 1..maxPatternLength, 1..4, [
8   % x1p1(x1)x1
9   1,0,1,0,    1,1,1,0,    1,0,1,0,
10  ]);
11
12 permutations = array4d(1..nrPatterns, 1..nrPermutations, 1..maxPatternLength, 1..sigma, [
13   % x1p1(x1)x1
14   1,2,3,    1,2,3,    1,2,3,
15   1,2,3,    1,3,2,    1,2,3,
16   1,2,3,    2,1,3,    1,2,3,
17   1,2,3,    2,3,1,    1,2,3,
18   1,2,3,    3,1,2,    1,2,3,
19   1,2,3,    3,2,1,    1,2,3,
20  ]);
21
22 nrVarsInPattern = array1d(1..nrPatterns, [
```



```

23 1,
24 ]);
25
26 nrPermsInPattern = array1d(1..nrPatterns, [
27 1,
28 ]);
29
30 morphicWordLength = 2;
31 morphicWord = array1d(1..morphicWordLength, [
32 0, 1,
33 ]);
34
35 numberOfMorphicWordImages = 2;
36 morphicWordImagesLengths = array1d(1..numberOfMorphicWordImages, [
37 5, 5,
38 ]);

```

---

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**

---

```

1 % input parameters
2 int : wordLength;
3 int : sigma;
4 int : nrPatterns;
5 int : maxPatternLength;
6 int : nrPermutations;
7 int : morphicWordLength;
8 int : numberOfMorphicWordImages;
9 array [1..morphicWordLength] of int: morphicWord;
10 array [1..numberOfMorphicWordImages] of int: morphicWordImagesLengths;
11 array [1..nrPatterns, 1..maxPatternLength, 1..4] of int: repetitions;
12 array [1..nrPatterns, 1..nrPermutations, 1..maxPatternLength, 1..sigma] of int:
    permutations;
13 array [1..nrPatterns] of int: nrVarsInPattern;
14 array [1..nrPatterns] of int: nrPermsInPattern;
15
16 % output decision variables
17 array [1..wordLength] of var 1..sigma: word;
18 array [1..(sum(i in 1..morphicWordLength) (morphicWordImagesLengths[morphicWord[i]
    + 1]))] of var 1..sigma: finalWord;
19
20 % returns length of variable indicated by 'variableIndex' based on 'labelIndex'
21 function int: length(int: variableIndex, int: wordLength, int: nrVars, int: label) =

```

## B. Code listings

```
22  if ( ceil (label / pow(wordLength, (nrVars - variableIndex + 1) - 1)) mod wordLength
23      == 0) then
24      wordLength
25  else
26      ceil (label / pow(wordLength, (nrVars - variableIndex + 1) - 1)) mod wordLength
27  endif;
28  predicate avoidPatterns(array[int] of var int: word, int: length) =
29      % for each pattern
30      forall (p in 1..nrPatterns)
31      (
32          % for each position in word
33          forall (start in 1..length)
34          (
35              let {
36                  % maximum number of labels based on number of variables
37                  int: maxNumberOfLabels = pow(length, nrVarsInPattern[p]);
38              } in
39              % move through all labels
40              forall (i in 1..maxNumberOfLabels)
41              (
42                  % if lengths assigned to variables make sense
43                  if (start + sum(k in 1..maxPatternLength where repetitions[p, k, 3] != 0) (length(
44                      repetitions[p, k, 3], length, nrVarsInPattern[p], i)) - 1 <= length) then
45                      exists (m in 1..nrPermsInPattern[p])
46                      (
47                          forall (z in 1..nrPermutations)
48                          (
49                              exists (varOcc in 1..maxPatternLength where
50                                  repetitions[p, varOcc, 3] != 0 /\ repetitions[p, varOcc, 1] == m)
51                              (
52                                  exists (nextOcc in 1..maxPatternLength where
53                                      (nextOcc != varOcc /\
54  (repetitions[p, varOcc, 3] == repetitions[p, nextOcc, 3]) /\
55  (repetitions[p, nextOcc, 2] == 0 \/\
56  repetitions[p, varOcc, 1] == repetitions[p, nextOcc, 1])))
57                              (
58                                  let {
59                                      var int: varLength = length(abs(repetitions[p, varOcc, 3]), length,
60  nrVarsInPattern[p], i),
61                                  } in
62                                  exists (l in 1..varLength)
63                                  (
64                                      let {
```

```

64         var int: occInWord = start + (if varOcc > 1 then
65             (sum(k in 1..(varOcc - 1))(length(repetitions[p, k, 3], length,
nrVarsInPattern[p], i))) else 0 endif),
66         var int: nextOccInWord = start + (if nextOcc > 1 then
67             (sum(k in 1..(nextOcc - 1))(length(repetitions[p, k, 3], length,
nrVarsInPattern[p], i))) else 0 endif),
68         var int: posFirst = occInWord + 1 - 1,
69         var int: posSecond =
70             (if (repetitions[p, varOcc, 4] == repetitions[p, nextOcc, 4])
then
71             (nextOccInWord + 1 - 1)
72             else (nextOccInWord + varLength - (1 - 1) - 1) endif)
73         } in
74         permutations[p, z, varOcc, word[posFirst]] !=
75         permutations[p, z, nextOcc, word[posSecond]]
76     )
77 )
78 )
79 )
80 )
81 )
82     else 1=1
83     endif
84 )
85 )
86 );
87
88 constraint avoidPatterns(word, wordLength);
89 constraint avoidPatterns(finalWord, (sum(i in 1..morphicWordLength) (
morphicWordImagesLengths[morphicWord[i] + 1]]));
90 constraint
91 forall (i in 1..morphicWordLength)
92 (
93     let {
94         var int: importantIndexInWord = (sum (k in 1..(morphicWord[i])) (
morphicWordImagesLengths[k])) + 1,
95         var int: importantIndexInFinalWord = (sum (k in 1..(i - 1)) (
morphicWordImagesLengths[morphicWord[k] + 1])) + 1,
96     } in
97     forall (j in 1..morphicWordImagesLengths[morphicWord[i] + 1])
98     (
99         word[importantIndexInWord + j - 1] = finalWord[importantIndexInFinalWord + j -
1]
100     )
101 );

```

## B. Code listings

```
102 solve::int_search(finalWord, input_order, indomain_min, complete) satisfy;  
103 output[show(finalWord)];
```

---

# 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>.

## Bibliography

- [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 Mercas, 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] Paweł 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](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](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>.

## Bibliography

- [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](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 Deiningler 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\\_34](https://doi.org/10.1007/978-3-642-38771-5_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.