Gray codes and efficient exhaustive generation for several classes of restricted words

Ahmad Sabri

SPIM

Thèse de Doctorat

école doctorale **sciences pour l'ingénieur et microtechniques**

UNIVERSITÉ DE BOURGOGNE

Gray codes and efficient exhaustive generation for several classes of restricted words

AHMAD SABRI

SPIM

Thèse de Doctorat

école doctorale **sciences pour l'ingénieur et microtechniques**

UNIVERSITÉ DE BOURGOGNE

THÈSE présentée par

AHMAD SABRI

pour obtenir le

Grade de Docteur de
l'Université de Bourgogne

Spécialité : **Informatique**

Gray codes and efficient exhaustive generation for several classes of restricted words

Unité de Recherche :
Laboratoire Electronique, Informatique et Image (LE2I)

Soutenue publiquement le 10 Avril 2015 devant le Jury composé de :

JEAN-MARC FÉDOU	Rapporteur	Professeur à l'Université de Nice Sophia - Antipolis
VLADY RAVELOMANANA	Rapporteur	Professeur à l'Université Paris Diderot - Paris 7
JEAN-LUC BARIL	Examinateur	Professeur à l'Université de Bourgogne - Dijon
ENRICA DUCHI	Examinateur	Maitre de Conference à l'Université Paris Diderot - Paris 7
OLIVIER TOGNI	Examinateur	Professeur à l'Université de Bourgogne - Dijon
VINCENT VAJNOVSZKI	Directeur de thèse	Professeur à l'Université de Bourgogne - Dijon

ACKNOWLEDGEMENTS

Praise to Allah for His blessing and for giving me this beautiful achievement. It is impossible for me to do the research and to pass my doctoral study without the support, involvement and help from other people.

First of all, I would like to express my sincere gratitude to my supervisor, Professor Vincent Vajnovszki, who gave me a lot of guidance, knowledge and insight during my doctoral study and research.

Thank you to all the jury members for reviewing and giving constructive suggestions on my thesis, as well as grading my thesis defense.

Thank you to the Directorate General of Higher Education (DIKTI), Ministry of Education Republic of Indonesia, for financing my study.

Thank you to Prof. Dr. E.S. Margianti, Prof. Suryadi Harmanto, Dr. Eri Prasetyo, Dr. Ernastuti, and Dr. Asep Juarna from Gunadarma University, Indonesia, where I work for.

I would like also to thank the administration people in Ecole Doctoral SPIM and in the laboratory LE2I, who facilitate me with the required things so that my research could go well.

Thank you to all my friends in the laboratory, with whom I interacted, had discussions, or just had a fun talk with cups of coffee in our hands.

I dedicate my doctoral degree to my beloved father, Fauzi Nawawi, who passed away in the last year of my study. May this achievement be my best gift to you father, that makes you smile upthere in heaven. And last but not least, I wish to thank my mother Ratna Fauzi, my lovely wife Novini Nilakusumah, my children Difa Abdussalam and Fatma Raihana, who have supported me with so much love and care.

Finally, thank you to all the people that involved in my study and research, whose names are impossible for me to mention here one by one. It is wonderful to have met you all.

Dijon, April 2015

Contents

I

Problems and Context

1

INTRODUCTION

Our research consists of three connecting parts, which are: combinatorial classes, Gray codes constructions, and generating algorithms. Therefore, it lies in two scientific domains: combinatorics and computer science. We consider three major classes of restricted words, in which the restrictions are applied subject to their growth, to avoidance of a factor or a set of patterns. Our Gray code constructions are induced by order relations, and they are derived from the original Binary Reflected Gray Code introduced by Frank Gray in his 1953 patent [Gra53]. We call them *Reflected Gray Code based order relations*. All the considered generating algorithms we present here are efficient, that is, they have time complexity proportional to the *number* of the generated objects, independent of the *size* of the object.

1.1/ MOTIVATIONS AND OBJECTIVES

1.1.1/ IN RELATION TO COMBINATORICS

There are several *ad-hoc* techniques to define Gray codes for combinatorial classes. However, those techniques lack of generality, in the sense they work only for the very specific combinatorial classes for which such Gray codes are defined. More general designs that work for a wide range of combinatorial classes tend to grow in recent years.

Among them are, for example, *ECO-based* Gray codes (ECO was defined formally in the context of enumeration [BLPP99]), Gray codes for *Gray structures* [BGPP07], for *strictly prefix* (or *suffix*) *partitioned list of words* [Wal08], for *reflectable languages* [LS09], and for *bubble languages* [RW09, RSW12].

Our research is within this spirit. Our Gray code definitions are based on order relations. More precisely, we propose the original Reflected Gray Code (RGC) order relation as "the base" of our Gray code constructions. Indeed, some variations of the original RGC order relation induce Gray codes for some combinatorial classes. In some cases, our techniques can be applied when other techniques, like reflectable languages, do not work

(i.e., for the sets which are not reflectable).

Some important questions might arise, such as: under what conditions a class, when listed in an RGC-based order, is a Gray code? If a restriction is applied to such a Gray code list, does the obtained sublist still remain a Gray code? What kind of restrictions that yield Gray codes? How to apply these restrictions into efficient generating algorithms? It seems there is no general answer for the first question, since the existence of Gray code for a particular class, with respect to an RGC-based order, depend on how the class and the order relation are defined. This relates to the answer "it depends" to the remaining questions. In addition, there is no general Gray code construction that works for every class. Nevertheless, it remains a scientific challenge for combinatorists to design more general Gray code constructions.

At first, the RGC-based order relations technique was used implicitly, for example in compositions [Kli81, Wal00]. Later on, it was developed systematically as a general method in order to define Gray codes (and corresponding generating algorithms) for various classes such as Fibonacci words [Vaj01], Lucas words [BV05], Lyndon words and relatives [Vaj07, Vaj08a], and restricted compositions and permutations [VV11]. In relation to classes of restricted words, some literatures propose Gray codes for restricted growth sequences, such as those given in [MNV11, MV13]; for factor avoiding words [Squ96]; and for pattern avoiding permutations [DFMV08, Bar09].

An order relation induces a unique list for a set of combinatorial objects. In particular, once a Gray code is defined for a class by means of RGC order, we can use all 'knowledge' of RGC order such as: ranking and unranking objects, random generation, and listing for a particular subclass. By applying restrictions to the list, the obtained restricted list (that is, the objects that do not satisfy the restrictions are excluded from the list) is a (possibly scattered) sublist of the original one; it is uniquely defined, and it still maintains the same order relation.

A classical example is the list of length-n binary words with respect to RGC order (the Binary Reflected Gray Code); and below is this list for $n = 4$ (see Table 2.5 for complete list):

$$\mathcal{G} = (0000, 0001, 0011, 0010, 0110, 0110, \ldots, 1011, 1001, 1000).$$

If we restrict this list to contain only binary words having exactly two 1s, then we obtain a 2-Gray code, which is a scattered sublist of $\mathcal{G}$, and for $n = 4$ is:

$$\mathcal{G}' = (0011, 0110, 0101, 1100, 1010, 1001).$$

More generally, restricting BRGC to binary words having exactly k 1s will also produce a Gray code (Nijenhuis and Wilf called this Gray code *the revolving door code* [NW78]).

Extending this idea, our research considers some other restrictions which preserve the Graycodeness, with respect to some RGC-based orders. In particular, we consider three

classes of restricted words, that are: restricted growth sequences, factor avoiding words, and pattern avoiding permutations. The variations of RGC order relation are considered in the first and the second mentioned classes. It might happen that, for a combinatorial class, a variation of RGC order induces more restrictive Gray code than that of the original one (as in the case of restricted growth sequences); or the original RGC order does not induce Gray code, but a variation of it induces one (as in the case of factor avoiding words). In the case of pattern avoiding permutations, we use the classical Steinhaus-Johnson-Trotter order (that is, the order relation induced by Steinhaus-Johnson-Trotter Gray code for permutations), and consider some patterns that preserve the Graycodeness. In fact, SJT order is implicitly in connection with the original RGC order; since the list of permutations in SJT order is order isomorphic to the list of their corresponding inversion tables in RGC order.

1.1.2/ In relation to computer science

The field of combinatorial algorithm includes some recent notable books of references. Donald Knuth's *The Art of Computer Programming, Vol. 4A - Combinatorial Algorithm* [Knu11], is entirely devoted to the generation of some classical combinatorial classes, such as: n-tuples, permutations, combinations, partitions, set partitions, and trees. *Combinatorial Generation*, a monograph by Frank Ruskey [Rus03], provides generating algorithms for many combinatorial classes, and analyze the their time and space complexity.

"Efficient", that is the keyword in developing algorithms. The amount of calculations per generated object should be constant (either in the worst case or in amortized sense), and independent of the object size. Gray codes can be considered as an efficient arrangement for combinatorial objects since two successive objects only differ in a (bounded) small amount of changes, independent of the object size. Thus, generating objects in Gray code manner can be a way to obtain efficient generating algorithms.

We provide efficient generating algorithms for each combinatorial class considered in this thesis. The task for restricting the generated words is done by 'restriction procedures', that are called from the main generating algorithm. By designing restriction procedures that define a particular class, the algorithm generates the objects of the desired class. This makes the algorithms more flexible and applicable for a wider range of combinatorial classes.

1.2/ Outline of the thesis

After Introduction in the current chapter, the rest of the thesis is presented in the following outline.

Chapter 2 gives some introductions to combinatorics, in context of our research. First, we explain what the combinatorial objects are, and also their representations. We explain some aspects of enumerative combinatorics involved in our research, that are, counting, exhaustive generation, and bijection between combinatorial objects. In addition, we give explanations about two other aspects: random generation and ranking/unranking combinatorial objects, even though we will not be using them later.

Chapter 3 explains each of the involved aspect in more details. We introduce formal definitions of the three considered combinatorial classes of words, where two of them belong to product sets, and the other belongs to permutations. Those considered classes are, respectively: restricted growth sequences, factor avoiding words, and pattern avoiding permutations. The notions of Gray code and the particular well-known Gray code constructions (Reflected Gray Code and Steinhaus-Johnson-Trotter Gray Code) are introduced and explained.

The algorithm section explains the concept of "efficient generating algorithms". Among such algorithms are: Constant Amortized Time (CAT) algorithm and loopless algorithm. Whenever possible, obtaining an efficient algorithm is a primary goal in designing algorithm for exhaustive generation of combinatorial objects.

Chapter 4 presents contemporary results in context of our research. The contemporary results are divided into four parts: that of Gray codes, of restricted growth sequences, of factor avoiding words, and of pattern avoiding permutations.

Chapter 5 presents our results on Gray code for restricted growth sequences. We consider four classes that satisfy our prescribed properties; they are subexcedant sequences, ascent sequences, restricted growth functions, and staircase words. In the first part, we give the results about their Graycodeness and generating algorithms, with respect to the original RGC order. The second part of this chapter gives similar results, with respect to Co-RGC order (a variation of the original RGC order). As by product of our results, we investigate the Graycodeness of the restricted ascent sequences.

Chapter 6 presents our results on Gray code for factor avoiding words. We consider words over alphabet $A_q = \{0, 1, \ldots, q-1\}$, and introduce the notion "zero periodicity property" as sufficient (but not necessary) condition for a set of words to be Gray code, when listed in RGC order. To deal with that property, we apply the original RGC order for even q, and its variation called Dual RGC order for odd q. The Graycodeness of factor avoiding words are categorized according to whether q is even or odd, and whether the forbidden factor induces zero periodicity or not. We also investigate the Graycodeness for particular cases, that are, for certain forbidden factors that do not induce zero periodicity. In the algorithm section, we provide an efficient algorithm for generating factor avoiding words. It applies an efficient factor matching technique adapted from that of Knuth-Morris-Pratt. At the end of the chapter, we give the implementation of our results to define the Gray code and, as an application, provide a CAT generating algorithm for cross-bifix-free words.

Chapter 7 presents our results on Gray codes for some classes of pattern avoiding permutations. We begin by presenting Steinhaus-Johnson-Trotter algorithm, that produces the list of all permutations of a given length (an "SJT-list"), and each generated permutation is obtained by an adjacent transposition to the previous one; so the list is actually a 2-adjacent Gray code. To investigate the Graycodeness, we use the classical bijection from inversion tables to permutations and introduce the notion of "restriction functions". Our restrictions to the permutations are subjected to avoid one of the four sets of regular patterns, that are $\{312, 321\}$, $\{321, 3412, 4123\}$, $\{312, 4321, 3421\}$, and $\{p12\ldots(p-1), 321, 231\}$, where the last is the set of p-generalized Fibonacci patterns. The reverse of the first, of the second, and the inverse of the fourth mentioned patterns, are also considered. By using bijective approach, we show that the SJT-list of permutations avoiding one of those sets of patterns remains Gray code (possibly less restricted). The generating algorithm section provides an efficient generating algorithm for permutations avoiding one of those sets of patterns. The algorithm generates and lists the permutations according to their relative order in SJT-list, so it can be considered as SJT-list with 'restriction'. In addition, we give the graph theoretic consequence of our results.

Finally, in the Appendix we give the SAGE implementations of our generating algorithms.

2

PRELIMINARIES

2.1/ COMBINATORICS

Combinatorics can be considered simply as mathematics of counting. More specifically, it is the mathematics of the enumeration, existence, construction, and optimization questions concerning finite sets satisfying a predefined configuration [Maz10].

2.1.1/ A BRIEF HISTORY

Some classical combinatorics problems such as Tower of Hanoi, tiling problems, magic or Latin squares, or works that related to combinatorics such as Murasaki diagrams of the Tale of the Genji, the 64 Hexagrams of the Fu Hsi ordering, show that human has already dealt with problems related to combinatorics since a long time ago.

Biggs [Big79] pointed out that the basic rules for counting have been taken for granted at least since Egyptian civilizations, as shown in the Rhind papyrus (ca. 1650 BC). It is named after Alexander Henry Rhind, a Scottish antiquarian, who purchased the discovered papyrus in 1858 in Luxor, Egypt. Problem 79 therein appears to deal with the summation of a series of powers of 7.

Perhaps the first mention of a combinatorics problem occured in an ancient Indian literature called Bhagabati Sutra, written around 300 BC. It posed a problem about how many ways one could take six tastes one, two, or three tastes at a time. Several ideas of combinatorics were developed by ancient Indian mathematicians in this period. It is worth mentioning that some of these developments are related to finding binomial coefficients, Fibonacci numbers, permutations, and combinations. Later on, the Middle East mathematicians also learned about binomial coefficients from Indian works, and found the connection to polynomial expansion. Abu Bakr ibn Muhammad ibn al Husayn Al-Karaji (ca. 953–1029) wrote on the binomial theorem and the geometrical arrangement of binomial coefficient, which is now well known as Pascal triangle. In a now lost work known only from subsequent quotation by Al-Samaw'al, Al-Karaji introduced the idea of

argument by mathematical induction.

Ancient Chinese were also interested in combinatorial calculations. The famous *Book of Change*, or *I Ching*, seems to be a compilation of material, some of which dates from the 7th century BC. The only relevance of the I Ching for combinatorics concerns the question about ordering the hexagrams. However, their standard ordering seems has no significance impact with the development of combinatorics. Apart from that, perhaps the most well-known Chinese contribution to this field is the subject of magic squares [Big79].

The Italian mathematician Leonardo Fibonacci (ca. 1170–1250), with his book *Liber Abaci*, influenced by Arabian and Indian ideas, perhaps might be considered as one who introduced combinatorics in Europe in 13th century. Classes of Fibonacci numbers are strongly related to many aspects in combinatorics.

In 18th century, Leonhard Euler gave the solution of the Königsberg Bridge problem. His published results marked the birth of graph theory and modern combinatorics, and perhaps since then combinatorics emerged as a formal branch of mathematics.

Today, combinatorics has expanded into branches such as enumerative combinatorics, optimization, analysis, and algebraic combinatorics. It plays important roles in pure mathematics, such as algebra, probability theory, topology, and geometry. Its applications spread into many science domains such as bioinformatics, cryptography, electrical engineering, and many more.

2.1.2/ Combinatorial class

'Combinatorial class' is a primitive notion, and roughly speaking, it is a set of objects endowed with some combinatorial properties. The term 'combinatorial object' can refer either to a single object belonging the class, or to the class itself [Wil09b]. A particular set of combinatorial objects is a subclass, with certain properties are fixed. For example, the class of permutations consists of sets P_n, the sets of permutations of length $n \geq 0$. By fixing $n = 5$, we have P_5, the set of permutations of length 5.

Ruskey proposed an intuitive definition that a class of combinatorial object is *elementary* if it satisfies a simple recurrence relation [Rus03]. Permutations, combinations, set or number partitions, binary trees and labeled graphs are some well-known elementary combinatorial objects.

2.1.3/ Representations of combinatorial object

A combinatorial object can be represented in various ways. For example, some ways to represent a permutation are by one line notation, by cycle notation, or by inversion table (see Table 2.1); a combination can be represented by set notation or by binary

word (see Table 2.2); a number partition by Young diagram; a set partition by set notation or restricted growth function (see Table 2.3); a labeled graph by diagram or adjacency matrix, and many more.

One-line notation	Cyclic notation	Inversion table
1 2 3	(1)(2)(3)	0 0 0
1 3 2	(1)(2 3)	0 0 1
2 1 3	(1 2)(3)	0 1 0
2 3 1	(1 2 3)	0 1 1
3 1 2	(1 3 2)	0 0 2
3 2 1	(1 3)(2)	0 1 2

Table 2.1: Some representations for permutations of the set $\{1, 2, 3\}$.

Set	Binary word
$\{1, 2, 3\}$	1 1 1 0 0
$\{1, 2, 4\}$	1 1 0 1 0
$\{1, 3, 4\}$	1 0 1 1 0
$\{2, 3, 4\}$	0 1 1 1 0
$\{1, 2, 5\}$	1 1 0 0 1
$\{1, 3, 5\}$	1 0 1 0 1
$\{2, 3, 5\}$	0 1 1 0 1
$\{1, 4, 5\}$	1 0 0 1 1
$\{2, 4, 5\}$	0 1 0 1 1
$\{3, 4, 5\}$	0 0 1 1 1

Table 2.2: Some representation for combinations of 3 objects taken from the set $\{1, 2, 3, 4, 5\}$.

Obviously, there is one-to-one correspondence between each representation within the same class. Which one is used depends on what aspects to be investigated in that class.

2.2/ ENUMERATIVE COMBINATORICS

Enumerative combinatorics is a branch of combinatorics specialized in counting and generating combinatorial objects under specified constraints. In addition, the *generation* of combinatorial object is a method to produce a particular object in the class. To obtain *all* combinatorial objects under particular parameters, the task is called the *exhaustive generation* of combinatorial objects. Since in this thesis we only consider exhaustive generation, for the brevity we use the term “generation” to refer “exhaustive generation”.

Set partition	Restricted growth function
$\{1,2,3,4\}$	0 0 0 0
$\{1,2,3\},\{4\}$	0 0 0 1
$\{1,2,4\},\{3\}$	0 0 1 0
$\{1,2\},\{3,4\}$	0 0 1 1
$\{1,2\},\{3\},\{4\}$	0 0 1 2
$\{1,3,4\},\{2\}$	0 1 0 0
$\{1,3\},\{2,4\}$	0 1 0 1
$\{1,3\},\{2\},\{4\}$	0 1 0 2
$\{1,4\},\{2,3\}$	0 1 1 0
$\{1\},\{2,3,4\}$	0 1 1 1
$\{1\},\{2,3\},\{4\}$	0 1 1 2
$\{1,4\},\{2\},\{3\}$	0 1 2 0
$\{1\},\{2,4\},\{3\}$	0 1 2 1
$\{1\},\{2\},\{3,4\}$	0 1 2 2
$\{1\},\{2\},\{3\},\{4\}$	0 1 2 3

Table 2.3: Some representation for partitions of the set $\{1,2,3,4\}$.

2.2.1/ COUNTING COMBINATORIAL OBJECTS

Stanley [Sta97] stated that the basic problem of enumerative combinatorics is *how* to find the cardinality of a given finite set (i.e., counting the number of objects of a finite set), by using the appropriate counting formula. For example, $n!$ counts the number of n-permutations. Counting formula can be given in several standard algebraic methods, those are: explicit formula, recurrence relation, asymptotic formula, and generating function (see Table 2.4 for examples). A relatively more recent method called ECO [BLPP99] is suitable to enumerate most of combinatorial classes.

Combinatorial class	Counting formula	Type of formula
n-permutations	$n!$	Explicit formula
n-permutations (large n)	$\sqrt{2\pi n}\left(\frac{n}{e}\right)^n$	Asymptotic formula
Dyck words of length $2n$	$\frac{1-\sqrt{1-4x}}{2x}$	Generating function
Fibonacci words of length n	$f(0)=0,\ f(1)=1,$ $f(n)=f(n-1)+f(n-2)$	Recurrence relations

Table 2.4: Some combinatorial classes with their counting formulas.

If the counting formula for a set is not known, or it is too complicated to deal with, then a solution is to design a 'counting algorithm'. In this case, the algorithm efficiency is crucial. Alternatively, one can use bijection between combinatorial classes.

2.2.2/ EXHAUSTIVE GENERATION

Exhaustive generation of combinatorial objects is a way to generate with no omissions nor repetitions all the combinatorial objects in a given class of a fixed object size. It can be used to test hypotheses about a class of objects, to solve some NP-complete problems, to analyze or prove programs, and often to exhibit new combinatorial properties of the class being considered. It can also be used to count the object in the class.

In *combinatorial optimization*, where every possible object is associated with a value, it might be necessary and feasible to evaluate all the possibilities under a particular condition, to find which one gives the optimality.

The exhaustive generation can be obtained by a generating algorithm, and the objects are generated according to a particular order, usually the lexicographic order or its generalizations, or bounded change orders. For that purpose, it is more convenient to have objects represented by tuples.

LEXICOGRAPHIC ORDER AND ITS GENERALIZATIONS

Lexicographic order is a formalization of the idea of ordering the words alphabetically, as in the dictionary. In lexicographic order, a word $a_1a_2\ldots a_n$ is *less than* (i.e., is ordered before) a word $b_1b_2\ldots b_n$ if, for some k, a_k is less than b_k, and $a_i = b_i$ for $i = 1, 2, \ldots, k-1$.

There are variations of lexicographic order derived from the original one, such as reverse lexicographic (relex) order, co-lexicographic (colex) order, or combinations of both relex and colex order. Note that in lexicographic order, two successive words can differ in arbitrary number of positions, and there are possibilities that they differ in all positions.

Chase introduced the generalization of lexicographic order, called the *Graylex* order [Cha89]. Graylex order, like the lexicographic order and its variations, produces a list of words which is either *prefix partitioned* or *suffix partitioned* (all the words having the same prefix/suffix are contiguous). A Graylex order simultaneously generalizes lexicographical and Reflected Gray Code order.

BOUNDED CHANGE ORDER

In *bounded change order*, the succeeding object is obtained by applying a number of changes to the previous one, and this number is bounded independently in the size of objects. However, such order does not necessarily exist for a given combinatorial class. In addition, if the bound is proven to be minimal, then we say such an order as a *minimal change order*.

A Graylex order might also be a bounded change order for a particular class. For example,

Lexicographic order	Number of changes
0 0 0 0	0
0 0 0 1	1
0 0 1 0	2
0 0 1 1	1
0 1 0 0	3
0 1 0 1	1
0 1 1 0	2
0 1 1 1	1
1 0 0 0	4
1 0 0 1	1
1 0 1 0	2
1 0 1 1	1
1 1 0 0	3
1 1 0 1	1
1 1 1 0	2
1 1 1 1	1

Bounded change order	Number of changes
0 0 0 0	0
0 0 0 1	1
0 0 1 1	1
0 0 1 0	1
0 1 1 0	1
0 1 1 1	1
0 1 0 1	1
0 1 0 0	1
1 1 0 0	1
1 1 0 1	1
1 1 1 1	1
1 1 1 0	1
1 0 1 0	1
1 0 1 1	1
1 0 0 1	1
1 0 0 0	1

Table 2.5: The list of binary words with respect to lexicographic order and to a bounded change order. The column "Number of changes" shows the amount of changes applied to the previous words to obtain the next one. Generally, for a set of length-n binary words, its list in lexicographic order yields n changes between successive words in the worst case; while in this particular bounded change order, the number of change is 1.

Reflected Gray Code order is a bounded (and also minimal) change order for the set of binary words. Ordering rules proposed in [BGPP07, Wal08, LS09] give minimal change order for Gray structures, strictly prefix/suffix partitioned list, or reflectable languages, respectively (explained later in Chapter 4).

Table 2.5 shows the lists of binary words of length 4, with respect to lexicographic order and a bounded change order (in this case it is given by Reflected Gray Code order). In general, lexicographic order gives the amount of changes equals to the word length (in the worst case), and in this example is 4; and in the bounded change order, the amount of change is 1, which is minimal (independent of the word length). The similar example is also shown in Table 2.6

2.2.3/ BIJECTION BETWEEN COMBINATORIAL OBJECTS

A function $f : A \to B$ is called a *bijection* if f satisfies both of the following properties:

- f is *injective* (*one-to-one*), that is, for any $a_1, a_2 \in A$, if $f(a_1) = f(a_2)$, then $a_1 = a_2$, and
- f is *surjective* (*onto*), that is, for any $b \in B$, there exists some $a \in A$ such that $f(a) = b$.

Lexicographic order	Number of changes
0 0 0 0	0
0 0 0 1	1
0 0 1 0	2
0 0 1 1	1
0 1 0 0	3
0 1 0 1	1
0 1 1 0	2
1 0 0 0	4
1 0 0 1	1
1 0 1 0	2
1 1 0 0	2

Bounded change order	Number of changes
0 0 0 0	0
0 0 0 1	1
0 0 1 1	1
0 0 1 0	1
0 1 1 0	1
0 1 0 1	2
0 1 0 0	1
1 1 0 0	1
1 0 1 0	2
1 0 0 1	2
1 0 0 0	1

Table 2.6: The list of length-4 binary words having at most two 1s, with respect to lexicographic order and to a bounded change order. Note that these lists are sublists of those in Table 2.5. Generally, for a set of length-n binary words, its list in lexicographic order yields n changes between successive words in the worst case; while in this particular bounded change order, the number of changes are bounded by 2.

Clearly, two finite sets A and B have the same cardinality if and only if there exists a bijection from one set to the other. In this case, we say that A and B are *isomorphic*. Moreover, if A and B are the sets of combinatorial objects and are isomorphic, it follows that generating objects in A is equivalent to generating objects in B (for example, when A is the set of inversion tables and B is the set of permutations, since the two sets are isomorphic). By using bijective proof, the properties of a class (such as the Graycodeness of the class) usually can be investigated by using the properties of its isomorphic class.

It might happen that the counting formula for a combinatorial class is not known yet. However, if there exists a bijection to such a class from a class with known counting formula, then the problem is solved; since the isomorphic classes have the same cardinality, so they share the same counting formula.

2.2.4/ RANDOM GENERATION

Random generation consists of designing an algorithm that chooses randomly and with uniform probability an object in a certain class. Apart from the theoretical interest, random generation is used principally for producing objects of large size, when the exhaustive generation is inefficient. It has applications in: the estimation of the average complexity of algorithms which deal with combinatorial objects; verification of combinatorial conjectures and assistance to one intuition (random generation is therefore associated with Monte-Carlo method); modeling (in Computer Science, Physics, Statistics, etc.) In these application areas, one needs to generate objects of very large size, which justifies the desire to achieve very low computational complexities, ideally linear in the size of the object to be constructed. One among methods for random generation is by using *object*

grammar given by Dutour and Fédou in [DF98].

2.2.5/ RANKING AND UNRANKING

Often it may be desirable to predetermine the position of a given object in a generated list without generating the whole list. This requires a ranking algorithm, and the inverse problem, determine the object with a given rank, requires an unranking algorithm. Formally, the *ranking function* for the class A_n of the objects of size n of a given type is a function which assigns a unique integer in the range $[0, card(A_n) - 1]$ to each of the $card(A_n)$ objects of size n. The corresponding *unranking function* is the inverse: given an integer between 0 and $card(A_n) - 1$, the value of the function is the object having this rank, and generally, the unranking algorithm is more complicated than the ranking one The traditional approach to this problem is to first define an ordering of the set of combinatorial objects and then find ranking and unranking functions relative to that ordering. For example, in lexicographic order, the rank of an object is simply the number of objects that precede it in lexicographic order. Naive implementations of ranking and unranking functions in lexicographic order are not linear and require arithmetic over large integers, but by not insisting on lexicographic order one may obtain efficient ranking and uranking algorithms. Such algorithms can be used for random generation or data compression.

3

Notations and Definitions

We consider the following classes of restricted words: restricted growth sequences, factor avoiding words, and pattern avoiding permutations. Each object in those classes is represented by *tuple* (i.e., an ordered list of elements). In particular, for $n > 0$, a length-n sequence, word, or permutation is represented as an n-tuple, and we adopt the convention that a lower case bold letter represents an n-tuple, for example: $\boldsymbol{a} = a_1 a_2 \dots a_n$. The empty (the length-$0$) sequence, word, or permutation is denoted by ϵ.

In this chapter, we give notions and definitions in relation with our considered classes of words, Gray codes, and efficient generating algorithm.

3.1/ Our considered classes of words

We define the three classes of restricted words below with respect to their parental classes, that are product sets and permutations. In general, all these classes belong to the classes of words.

3.1.1/ Product sets

Given n finite sets S_i, $1 \leq i \leq n$, the *product set* $S_1 \times S_2 \times \dots \times S_n$ is the set of n-tuples $\boldsymbol{s} = s_1 s_2 \dots s_n$, where $s_i \in S_i$ for all i, $1 \leq i \leq n$. In particular, if $S_i = \{0, 1, \dots, (i-1)\}$ for all $i = 1, 2, \dots, n$, then the product set is called the set of *subexcedant sequences* of length n.

Restricted growth sequences defined by means of statistics

A *statistic* on a set of sequences is an association of an integer to each sequence in the set. For a sequence $s_1 s_2 \dots s_n$, its length minus one, numbers of ascents/levels/descents, maximal value, and last value are classical examples of statistics. Table 3.1 shows the definitions for such statistics.

Statistics	Symbols and definitions	Examples
len	$\mathtt{len}(s_1 s_2 \dots s_n) = n - 1$	$\mathtt{len}(00213) = 4$
ascent	$\mathtt{asc}(s_1 s_2 \dots s_n) = \mathrm{card}\{i \mid 1 \le i < n \text{ and } s_i < s_{i+1}\}$	$\mathtt{asc}(45012) = 3$
maximum	$\mathtt{m}(s_1 s_2 \dots s_n) = \max\{s_1, s_2, \dots, s_n\}$	$\mathtt{m}(21573) = 7$
last value	$\mathtt{lv}(s_1 s_2 \dots s_n) = s_n$	$\mathtt{lv}(31254) = 4$
level	$\mathtt{lev}(s_1 s_2 \dots s_n) = \mathrm{card}\{i \mid 1 \le i < n \text{ and } s_i = s_{i+1}\}$	$\mathtt{lev}(10003) = 2$
descent	$\mathtt{des}(s_1 s_2 \dots s_n) = \mathrm{card}\{i \mid 1 \le i < n \text{ and } s_i > s_{i+1}\}$	$\mathtt{des}(62101) = 3$

Table 3.1: The definition of several statistics

Definition 1: `st`-restricted growth sequence

For a given statistic `st`, an `st`-*restricted growth sequence* $s_1 s_2 \dots s_n$ is a sequence with $s_1 = 0$ and

$$0 \le s_{k+1} \le \mathtt{st}(s_1 s_2 \dots s_k) + 1 \text{ for } 1 \le k < n, \tag{3.1}$$

and the set of `st`-restricted growth sequences is the set of all sequences $s_1 s_2 \dots s_n$ satisfying relation (3.1).

Briefly, an `st`-restricted growth sequence $s_1 s_2 \dots s_n$ is a sequence where, for $1 \le k < n$, the value range for s_{k+1} are restricted by $\mathtt{st}(s_1 s_2 \dots s_k)$. From this definition, it follows that any prefix of an `st`-restricted growth sequence is also (a shorter) `st`-restricted growth sequence.

Factor avoiding words

Words over a finite alphabet are special cases of product sets, where $S_1 = S_2 = \dots S_n = A$; in this case the product sets are denoted by A^n [Com74, p. 3], and A is called an *alphabet*, which is generally a set of distinct symbols. Alternatively, a *word of length* n over the alphabet A is an n-tuple $\boldsymbol{a} = a_1 a_2 \dots a_n$, where $a_i \in A$ for all i, $1 \le i \le n$, and $\boldsymbol{a}$ can be seen as function $\{1, 2, \dots, n\} \to A$. A q-*ary word* is simply a word over the particular alphabet $A_q = \{0, 1, \dots, q-1\}$.

A^n denotes the set of words of length n over A, $A^* = \cup_{n \ge 0} A^n$, and $A^+ = \cup_{n \ge 1} A^n$. The word $\boldsymbol{f} \in A^*$ is a *factor* of $\boldsymbol{a} \in A^* \cup A^\infty$ if there are $\boldsymbol{b} \in A^*$ and $\boldsymbol{c} \in A^* \cup A^\infty$ such that $\boldsymbol{a} = \boldsymbol{b}\boldsymbol{f}\boldsymbol{c}$; when $\boldsymbol{b} = \epsilon$ (resp. $\boldsymbol{c} = \epsilon$), then $\boldsymbol{f}$ is a *prefix* (resp. *suffix*) of $\boldsymbol{a}$; and in this case, the prefix or the suffix is *proper* if $\boldsymbol{f} \ne \boldsymbol{a}$ and $\boldsymbol{f} \ne \epsilon$.

For $\boldsymbol{a} \in A^* \cup A^\infty$ and $X \subset A^* \cup A^\infty$ we denote by $\boldsymbol{a} \mid X$ the set of words in X having the prefix $\boldsymbol{a}$, and by $X(\boldsymbol{a})$ the set of words in X that avoiding $\boldsymbol{a}$. Clearly $A^*(\boldsymbol{a}) = \cup_{n \ge 0} A^n(\boldsymbol{a})$.

By the previously defined notions, we have the following definition:

Definition 2: Factor avoiding q-ary word

For a given word $\boldsymbol{f}$, the word $\boldsymbol{a} = a_1 a_2 \dots a_n$ is said to *avoid* $\boldsymbol{f}$ if $\boldsymbol{a}$ does not contain $\boldsymbol{f}$ as its factor. We denote by $A_q^n(\boldsymbol{f})$ the set of q-ary words of length n that avoids factor $\boldsymbol{f}$.

A *bifix* (or *border*) of a word is a factor which is both a prefix and a suffix. A word is said to be *bifix-free* if it does not contain any bifix [Nie73].

Definition 3: Cross-bifix-free word

A set of bifix-free words is *cross-bifix-free* if, given any two words, any prefix of the first one is not a suffix of the second one.

3.1.2/ PERMUTATIONS

A *permutation* of a finite set S is an arrangement of all elements in S. In a permutation, the order of arrangement is considered; for example, if $S = \{1, 2, 3\}$, a permutation 132 is different from 231. We denote by P_n the set of all permutations of $S_n = \{1, 2, \dots, n\}$. Alternatively, a permutation of a set S is a bijection from S onto itself.

Restrictions to a set of permutation yield a set of restricted permutations. In literature, there are various ways to restrict permutations. Among them, the well-known class of restricted permutations is the class of *pattern avoiding permutations*.

PATTERN AVOIDING PERMUTATIONS

Let P_n be the set of length-n permutations. We say that $\boldsymbol{\pi} \in P_n$ *contains* a pattern $\boldsymbol{\tau} \in P_k$ if there is a subsequence $1 \le i_1 < i_2 < \cdots < i_k \le n$ such that $\pi_{i_1} \dots \pi_{i_k}$ is order isomorphic to $\boldsymbol{\tau}$.

Definition 4: Pattern avoiding permutation

For a given permutation $\boldsymbol{\tau}$, a permutation $\boldsymbol{\pi}$ is said to *avoid* $\boldsymbol{\tau}$ if $\boldsymbol{\pi}$ does not contain subsequence that is order isomorphic to $\boldsymbol{\tau}$. We denote by $P_n(\boldsymbol{\tau})$ the set of permutations that avoids $\boldsymbol{\tau}$. Similarly, for a set T of permutations, $P_n(T)$ is the set of permutations in P_n avoiding each permutation in T

3.2/ GRAY CODE

In 1947, Frank Gray, an engineer at Bell Laboratories, invented an analog-to-digital signal conversion method so that spurious output from electromechanical switches can be prevented. His method was based on a binary numerical system, which at that moment

had as yet no recognized name, and just referred as “reflected binary code”. Not until 1953 when finally this code was named as *Gray Code*, or to be more specific, *Binary Reflected Gray Code* (BRGC). However, the code itself had been studied and applied long before. For example, a French engineer named Emile Baudot used it in telegraphy in 1878. Briefly, for an integer n, BRGC is a list of all length n binary words (or sequences), ordered so that the successive words differ only in one position, and by $+1$ or -1 in this position. The term “reflected” refers to the reflection technique used in the construction. The amount of position(s) where two distinct words differ is called the *Hamming distance*. In BRGC, the Hamming distance between any two successive words is 1.

In combinatorics context, Gray code has come through a less restrictive definition (which sometimes called as *combinatorial Gray code*). Here we adopt the definition from Walsh [Wal08].

Definition 5: Gray code

A *Gray code* is an infinite set of word-lists with unbounded word-length such that the Hamming distance between any two successive words in any list is bounded independently of the word-length.

Through this thesis, we will often use the notions defined below to describe Gray code more specifically:

Definition 6: d-(adjacent) Gray code

A d-Gray code is a Gray code where the Hamming distance is bounded by a constant d. In addition, if the positions where two successive words differ are always adjacent, then such lists are called a d-*adjacent* Gray code.

3.2.1/ Reflected Gray Code

In the following we provide the construction rules for BRGC, followed by that for Reflected Gray Code (RGC), that is, the generalization of BRGC to q-ary words. For $n > 1$, construct the n-bit Gray code by first appending 0 to each element of the $(n-1)$-bit Gray code, then list the $(n-1)$-bit Gray code in reverse, appending 1 to each element. The following recursive relations represent this technique:

$$\mathcal{B}_n = \begin{cases} \epsilon & \text{if} \quad n = 0, \\ 0\mathcal{B}_{n-1},\ 1\overline{\mathcal{B}_{n-1}}, & \text{if} \quad n > 0, \end{cases} \tag{3.2}$$

where ϵ is the empty word, $\overline{\mathcal{B}_{n-1}}$ is the reverse of $\mathcal{B}_{n-1}$. See Table 3.2 for the list of BRGC of length 5, generated by relation (3.2).

Now, let G_q^n be the set of length-n q-ary words $s_1 s_2 \dots s_n$ with $s_i \in \{0, 1, \dots, q-1\}$; or

0 0 0 0 $\underline{0}$	1 1 0 0 $\underline{0}$
0 0 0 $\underline{0}$ 1	1 1 0 $\underline{0}$ 1
0 0 0 1 $\underline{1}$	1 1 0 1 $\underline{1}$
0 0 $\underline{0}$ 1 0	1 1 $\underline{0}$ 1 0
0 0 1 1 $\underline{0}$	1 1 1 1 $\underline{0}$
0 0 1 $\underline{1}$ 1	1 1 1 $\underline{1}$ 1
0 0 1 0 $\underline{1}$	1 1 1 0 $\underline{1}$
0 $\underline{0}$ 1 0 0	1 $\underline{1}$ 1 0 0
0 1 1 0 $\underline{0}$	1 0 1 0 $\underline{0}$
0 1 1 $\underline{0}$ 1	1 0 1 $\underline{0}$ 1
0 $\underline{1}$ 1 1 1	1 0 1 1 $\underline{1}$
0 $\underline{0}$ 1 1 0	1 0 $\underline{1}$ 1 0
0 1 0 1$\underline{0}$	1 0 0 1 $\underline{0}$
0 1 0 $\underline{1}$ 1	1 0 0 $\underline{1}$ 1
0 1 0 0 $\underline{1}$	1 0 0 0 $\underline{1}$
0 1 $\underline{0}$ 0 0	1 0 0 0 0

Table 3.2: The columnwise list $\mathcal{B}_5$, the BRGC of length 5. The symbol that differs with that of the next sequence is underlined.

equivalently, G_q^n is the product set $\{0, 1, \ldots, q-1\}^n$. The *Reflected Gray Code* for the set G_q^n, denoted by $\mathcal{G}_q^n$, is the natural extension of the Binary Reflected Gray Code to this set. The list $\mathcal{G}_q^n$ is defined recursively by the following relation [Er84]:

$$\mathcal{G}_q^n = \begin{cases} \epsilon & \text{if} \quad n = 0, \\ 0\mathcal{G}_q^{n-1},\ 1\overline{\mathcal{G}_q^{n-1}},\ 2\mathcal{G}_q^{n-1},\ \ldots,\ (q-1)\widehat{\mathcal{G}_q^{n-1}} & \text{if} \quad n > 0, \end{cases} \tag{3.3}$$

where ϵ is the empty word, $\overline{\mathcal{G}_q^{n-1}}$ is the reverse of $\mathcal{G}_q^{n-1}$, and $\widehat{\mathcal{G}_q^{n-1}}$ is $\mathcal{G}_q^{n-1}$ or $\overline{\mathcal{G}_q^{n-1}}$ according to q is odd or even.

In $\mathcal{G}_q^n$, two successive words differ in a single position and by $+1$ or -1 in this position.

Order relation inducing Reflected Gray Code

The Reflected Gray Code can also be constructed by using order relation. The order relation defined in the following induces the Reflected Gray Code on G_q^n .

Definition 7: Reflected Gray Code order

The *Reflected Gray Code order* on G_q^n is defined as: $\boldsymbol{s} = s_1 s_2 \ldots s_n$ is less than $\boldsymbol{t} = t_1 t_2 \ldots t_n$, denoted by $\boldsymbol{s} \prec \boldsymbol{t}$, if either

- $\sum_{i=1}^{k-1} s_i$ is even and $s_k < t_k$, or
- $\sum_{i=1}^{k-1} s_i$ is odd and $s_k > t_k$,

where k is the leftmost position where $\boldsymbol{s}$ and $\boldsymbol{t}$ differ.

We denote the RGC order by $\prec$ order. Table 3.3 shows the list of sequences in G_3^3 ordered by $\prec$ order. It is easy to see this list is equal to $\mathcal{G}_3^3$.

0 0 $\underline{0}$	1 2 $\underline{2}$	2 0 $\underline{0}$
0 0 $\underline{1}$	1 2 $\underline{1}$	2 0 $\underline{1}$
0 $\underline{0}$ 2	1 $\underline{2}$ 0	2 $\underline{0}$ 2
0 1 $\underline{2}$	1 1 $\underline{0}$	2 1 $\underline{2}$
0 1 $\underline{1}$	1 1 $\underline{1}$	2 1 $\underline{1}$
0 $\underline{1}$ 0	1 $\underline{1}$ 2	2 $\underline{1}$ 0
0 2 $\underline{0}$	1 0 $\underline{2}$	2 2 $\underline{0}$
0 2 $\underline{1}$	1 0 $\underline{1}$	2 2 $\underline{1}$
$\underline{0}$ 2 2	$\underline{1}$ 0 0	2 2 2

Table 3.3: The list of 3-ary sequences of length 3, G_3^3, ordered by $\prec$ order. The changed symbols to obtain the next sequence are underlined. The list is equal to $\mathcal{G}_3^3$.

THE REFLECTED-GRAY-CODE-BASED ORDER RELATIONS

The RGC-based order relations are variations of RGC order relation. They are obtained by some slight modifications to the original relations in Definition 7. As we mentioned earlier, it might happen that a variation of RGC order induces more restrictive Gray code than that of the original one; or the RGC order does not induce Gray code, but a variation of it induces one; or else, a variation might give the properties that we want to investigate. The RGC-based order relations we consider through this thesis are *Co-Reflected Gray Code order* and *Dual Reflected Gray Code order* (we will define them later in the subsequent chapters).

3.2.2/ STEINHAUS-JOHNSON-TROTTER GRAY CODE

A classical exhaustive generation technique for unrestricted permutations was given independently by Johnson [Joh63] and Trotter [Tro62]; while previously in 1958 Steinhaus had posed a puzzle related to generating permutations and solved it in the similar way [Ste64]. Their method, known as *Steinhaus-Johnson-Trotter* (SJT) algorithm, lists all permutations of length n so that any successive permutations differ by an adjacent transposition. The obtained list is a 2-adjacent Gray code, so it is also called the Steinhaus-Johnson-Trotter Gray code. See Table 3.4 for permutations of length 4 listed by this method.

3.3/ EFFICIENT GENERATING ALGORITHM

An *exhaustive generating algorithm* is an algorithm to generate exhaustively all objects belong to a class of combinatorial objects. For the brevity, we will use the shorter term

1 2 <u>3 4</u>	<u>4 3</u> 2 1
1 <u>2 4</u> 3	3 <u>4 2</u> 1
<u>1 4</u> 2 3	3 2 <u>4 1</u>
4 1 <u>2 3</u>	<u>3 2</u> 1 4
<u>4 1</u> 3 2	2 3 <u>1 4</u>
1 <u>4 3</u> 2	2 <u>3 4</u> 1
1 3 <u>4 2</u>	<u>2 4</u> 3 1
<u>1 3</u> 2 4	4 2 <u>3 1</u>
3 1 <u>2 4</u>	<u>4 2</u> 1 3
3 <u>1 4</u> 2	2 <u>4 1</u> 3
<u>3 4</u> 1 2	2 1 <u>4 3</u>
4 3 <u>1 2</u>	2 1 3 4

Table 3.4: The list of permutations of length 4 generated by SJT algorithm. The transposed symbols to obtain the next permutation is underlined. The list is 2-adjacent Gray code.

"generating algorithm". Usually the output of such algorithm is presented in a list form, arranged according to a certain order, such as lexicographic order or Gray code order. Every generating algorithm is also an enumeration algorithm, since each object can be counted when it is generated. From the computer science point of view, it is necessary to have an efficient generating algorithm to do this task.

The *time complexity* indicates the worst-case running time of an algorithm, regardless which computer is used to execute it. We denote the time complexity by using *Big O* notation. Let $f : \mathbb{N} \to \mathbb{R}$ and $g : \mathbb{N} \to \mathbb{R}$; then, we denote by $f = O(g)$ if there is a constant c and an integer n_0 such that $f(n) \leq cg(n)$ for $n > n_0$. If f and g both describe the number of computations required for two algorithms given input size n, then $f = O(g)$ means that f is 'not worse' than g when the problem size is large (see [KT12]). Generally, we say f *has time complexity* $O(g)$, or alternatively, f *runs in* $O(g)$ *worst-case time*.

For a generating algorithm, the time complexity is very crucial since the *number* of generated objects (the cardinality of the set) is, generally, an exponential function of the *size* of the generated objects. In terms of performance, the primary goal in developing a generating algorithm is to obtain an algorithm whose running time is proportional to the number of generated objects [RS03]. For that purpose, we need the algorithm that runs in constant time per generated object. It can be constant in average, or constant in the worst case, regardless the length of the object. In both cases, we consider such algorithm as *efficient*.

3.3.1/ Constant Amortized Time algorithm

A generating algorithm runs in Constant Amortized Time (CAT) if there is a constant c such that for any size of the objects we have:

$$\frac{\text{the total amount of computations}}{\text{number of generated objects}} \leq c. \tag{3.4}$$

In other words, such algorithm generates each object in $O(1)$ time in amortized sense, and the time complexity is independent to the size of the object [Rus03]. With this consideration, a generating algorithm that runs in CAT (a CAT algorithm) is considered an efficient algorithm.

A recursive algorithm runs in CAT if it satisfies the following properties:

1. Every call results in the output of at least one object.
2. Excluding the computation done by recursive calls, the amount of computation of any call is proportional to the degree of a call (the number of immediate recursive calls initiated by the current call).
3. The number of calls of degree one, if they exist, is $O(N)$ (where N is the number of generated objects).

The three properties above are called the *CAT principle* [Rus03]. Alternatively, based on the second property above, if a recursive generating algorithm satisfies

$$\frac{\text{the total amount of calls}}{\text{number of generated objects}} \leq c, \tag{3.5}$$

then it runs in CAT, and we call such algorithm a *CAT algorithm*.

3.3.2/ LOOPLESS ALGORITHM

A generating algorithm is *loopless* (or *loop-free*) if, after the initial object is generated, each succeeding object may be obtained by $O(1)$ worst-case time (see [Ehr73]). A loopless generating algorithm is considered an efficient algorithm.

A loopless generating algorithm produces a Gray code list. The converse is not true: Gray codes can be generated by algorithms that are not loopless. Essentially, there are two techniques to design loopless generating algorithms: Ruskey's method by *finished and unfinished lists* and Ehrlich's *'e'* array [Ehr73] which was generalized by Walsh [Wal08].

4

State of the Art

4.1/ Gray codes

The classical Binary Reflected Gray Code (BRGC), after its formal introduction in 1953 [Gra53], came through implementations on many combinatorial classes, including permutations and combinations. Later on, the term *combinatorial Gray code* was introduced in 1980 to refer to any method for listing combinatorial objects such that successive objects differ in some pre-specified small way [JJW80]. This generalized the notions of the original Gray code, in which successive binary words differ in one position, and by 1 or -1 in this position.

The study about combinatorial Gray codes was popularized by Wilf in late 1980s [Wil88, Wil89], and has been developing rapidly since then. In the beginning, many literatures propose *ad hoc* Gray codes, which means they only work for very specific combinatorial class. The comprehensive survey on such Gray codes were given in [Sav97]. More general designs that fit to many combinatorial classes tend to grow in recent years. Such designs utilize the common properties among combinatorial classes. Here we give briefly some notable contemporary results among them.

4.1.1/ ECO-based Gray codes

Introduced by Barcucci et al. [BLPP99], ECO (Enumerating Combinatorial Objects) originally was a technique to enumerate combinatorial objects, and it is based on *succession rules* method. This technique uses generating trees (or ECO operators) and usually objects are encoded by their corresponding path in the generating tree; and often it is possible to translate the obtained codes into codes for objects. In the context of ECO-based Gray codes, a (k) or $(\bar{k})$ labeled node in the generating tree corresponds to a word with k successors; and the successors of a (k) labeled node are the same as for $(\bar{k})$, but in reverse order. In particular, this technique can induce Reflected Gray Code. Vajnovszki [Vaj08b] used this technique to construct Gray codes for combinations, Dyck and grand

Dyck words. The results were extended to Gray codes for Motzkin and Schröeder words in [Vaj12a]. The implementation on several classes of pattern avoiding permutations was given in [DFMV08]. The generation for p-generalized Fibonacci and Lucas permutations were given by Baril and Do [Bar06], and they posed an open problem for Gray codes considerations.

4.1.2/ Gray codes for Gray structures

This technique was introduced by Bernini et al. [BGPP07]. It is also based on succession rules. Moreover, the Gray code obtained by this technique can be generalized to all succession rules with the *stability property*, that is, each labeled node (k) has in its successors two labeled nodes c_1 and c_2, always in the same position, regardless of k. The sets of combinatorial objects whose construction can be encoded by a succession rule with the stability property are called *Gray structures*, and in [BGPP07] was shown that there always exist Gray codes for them.

4.1.3/ Gray codes for strictly prefix (or suffix) partitioned word-list

The *defining sequence* concept was introduced by Walsh [Wal08]. If a list of length-n words is prefix-partitioned, then for any positive integer $i < n$ the interval of words (that is, sublist containing consecutive words) with the same prefix of length $i-1$ is partitioned into sub-intervals with the same prefix of length i, so that in this interval the i-th letter follows a sequence of values, depending on the prefix, such that all the copies of the same value are contiguous in the sequence. The subsequence of distinct values assumed by the i-th letter is called the defining sequence of the prefix

Bitner, Ehrlich, and Reingold [BER76] presented a loopless algorithm to generate the BRGC. Walsh [Wal08] applied the concept of *defining sequences* to their algorithm, which he call BER algorithm, and showed that it works on any prefix (or suffix) partitioned word-list satisfying the condition that every defining sequence has at least two values. In this case, the list is said to be *strictly prefix* (or *suffix*) *partitioned*.

4.1.4/ Gray codes for reflectable languages

Reflectable language was introduced by Li and Sawada [LS09]. A *language* (i.e., class of words) L over the alphabet A is said to be *reflectable* if for every word $w_1w_2\dots w_n$ and $i > 1$ there exists two distinct symbols x_i and y_i in A such that if $w_1w_2\dots w_{i-1}$ is a prefix of a word in L then both $w_1w_2\dots w_{i-1}x_i$ and $w_1w_2\dots w_{i-1}y_i$ are also prefixes of words in L.

For any combinatorial class satisfying such conditions, there exist construction yielding

Gray code. Combinatorial classes such as binary and k-ary words, restricted growth functions and tails, binary and k-ary trees, are proven to be reflectable; thus, they can be listed in a Gray code manner.

Roughly, the previous three techniques are equivalent, but expressed and motivated in different way.

4.1.5/ COOL-LEX ORDER AND BUBBLE LANGUAGES

In 2005, Ruskey and Williams presented a new method for generating combinations [RW05]. The combinations are represented as binary words with s 0s and t 1s, and are generated by a technique called *prefix shifts*, with a remarkably simple rule: identify the shortest prefix ending in 010 or 011 (or the entire word if no such prefix exists) and then rotate (shift) it by one position to the right. This technique implies that successive combination can be generated by transposing only one or two pairs of bits (which induces Gray code); and another important result that such a technique induces a new order relation, which they call *cool-lex* order. Cool-lex order also induces Gray code for multiset permutations [Wil09a].

A *bubble language* (introduced in 2012 by Ruskey, Sawada, and Williams) is a set of binary words with a simple closure property: The first 01 of any word can be replaced by 10 to obtain another word in the set. Bubble language can represent a wide range of combinatorial classes, such as: combinations, necklaces, k-ary Dyck words, Lyndon words, and interval graphs. The words in any bubble language are proven to be a Gray code when listed in cool-lex order [RSW12].

4.1.6/ GRAY CODES INDUCED BY RGC ORDER RELATION OR ITS VARIATIONS

RGC order relation was used implicitly by Knuth, when he defined Gray codes for integer compositions in his unpublished answer to a question of Nijenhuis and Wilf, around late 1970s. Knuth's Gray code was then implemented to an efficient algorithm by Klingsberg [Kli81]. Later on, Walsh gave an efficient algorithm for generating a Gray code for bounded compositions of an integer, based on Knuth's Gray code [Wal00].

Further, the RGC order relation technique has been developed systematically as a general method to define Gray codes for various classes. In this direction, RGC-based Gray codes and some of its variations (called RGC-based order relations) are worth to be mentioned, such as: Gray codes for Fibonacci and Lucas words, which are given in, respectively, [Vaj01] and [BV05]; for Lyndon words and relatives in [Vaj07, Vaj08a]; and for restricted compositions and permutations in [VV11].

Our Gray codes techniques are within this category.

4.2/ RESTRICTED GROWTH SEQUENCES

Literatures about restricted growth sequences mostly focus on four particular mainstream classes: subexcedant sequences, ascent sequences, restricted growth functions, and staircase words.

Ascent sequences were introduced in joint work of Bousquet-Mélou, Claesson, Dukes, and Kitaev [BMCDK10]. By constructing bijections, they showed that ascent sequences are equinumerous with three other combinatorial classes: unlabeled (2 + 2)-free posets, Stoimenows involutions, and permutations avoiding $3\overline{1}52\overline{4}$. Duncan and Steingrímsson introduced the study about pattern avoiding ascent sequences in [DS11]. Among the involved patterns, they exhibited a connection between ascent sequences and restricted growth functions. Further study by Chen, Dai, Dokos, Dwyer, and Sagan [CDD+13] established a bijection which led to the equidistribution of the pair of statistics (asc; rmin) over 021-avoiding ascent sequences and over 132-avoiding permutations, where rmin is the right-to-left minima statistic; and these results confirmed one of the conjectures posed in [DS11]. In [MS14], Mansour and Shattuck gave the enumerative results for 1012-avoiding and 0123-avoiding ascent sequences; while Pudwell did that for 0021-avoiding ascent sequences [Pud14]. Those enumerative results confirmed the three conjectures posed in [DS11]. However, so far there are no studies specifically dedicated to Gray codes for restricted ascent sequences.

The restricted growth functions encode the partitions of an n-set. The subject of Gray codes for restricted growth functions was triggered by a question posed by Nijenhuis and Wilf in [NW75]: “is it possible to arrange all partitions of an n-set so that each partition is obtained from its immediate predecessor by changing the class of exactly one object?”. Knuth gave the affirmative answer in his unpublished work, which became the first Gray code in recursive form for such set. Alternatively, the non recursive construction was given by Kaye [Kay76]. Later on, Ruskey and Savage generalized Ehrlich’s 1973 results [Ehr73] (where he introduced loopless generating algorithms for several combinatorial classes) to obtain Gray codes for restricted growth functions [RS94]. This technique was also reconsidered in [Rus93].

Another Gray code and loop-free generating algorithm for restricted growth functions was given by adopting plane tree techniques in [MN08]. In 2011, Mansour, Nassar, and Vajnovszki gave Gray codes for generalized restricted growth functions called e-restricted growth functions [MNV11]. Gray codes for several classes belong to st-restricted growth sequences (i.e., restricted growth sequences defined by statistics) were considered in [MV13].

There is no *strong* Gray code for restricted growth functions, that is, where two successive restricted growth functions differ in a single position and by 1 or -1 in this position.

The subexcedant sequences are mostly used as an alternative way to represent permuta-

tions and also for their exhaustive generation. A technique using subexcedant sequences to encode permutations were introduced in 1888 [Lai88]. About 70 years later, this technique was reintroduced by Lehmer [Leh60] and such sequence is now known as *Lehmer code*. Another well known technique is by using *inversion table* [Sta97], which is the Lehmer code for the inverse permutation. Dijkstra [Dij76] proposed an efficient algorithm to generate $n!$ permutations by using their inversion tables, implicitly ordered in RGC order, which eventually generates Steinhaus-Johnson-Trotter list for permutations.

In 2000, Babson and Steingrímsson [BS00] introduced the notion of what is now known as *vincular patterns* in permutations. Vajnovszki [Vaj12b] gave several bijections between subexcedant sequences which, when applied to Lehmer code, yield new permutation codes which count occurrences of some vincular patterns. In recent years, Foata and Han introduced two new permutation codings, one of which is based on Lehmer code [FH09].

Vajnovszki and Vernay [VV11] proved that the list for the set of even (or odd) permutations, as well as the set of permutations with an even (or odd) number of cycles, are Gray codes when their corresponding inversion tables are listed with respect to RGC order.

Stanley introduced the class of sequences defined in [Sta97, exercise (u), p. 222], which referred as staircase words in [MV13]. The cardinality for this class follows Catalan numbers.

In relation to Gray codes, the constructions provided by Bernini et.al. [BGPP07], Walsh [Wal08], Li and Sawada [LS09], Mansour and Vajnovszki [MV13] work on those four classes of restricted growth sequences mentioned above. However, no literature proposes Gray codes for those classes using RGC-based order technique.

4.3/ FACTOR AVOIDING WORDS

Perhaps the most challenging task in generating factor avoiding words is to find efficiently the occurrence of a given factor in the word. Knuth, Morris, and Pratt introduced, in their joint paper, the algorithm to find the occurrence of a word factor f within a given word w [KMP77]. They employed the notions *period* (a shift that causes the word to match over itself) so that if a mismatch occurs, the beginning of the next possible occurrence of f in w can be determined. In [GO81a] Guibas and Odlyzko introduced the notion *correlation* of two words (the representation of how the second word can overlap into the first), and used the generating function to enumerate the words with a given correlation. In their subsequent work [GO81b], they provided a generating function that enumerates the q-ary words of length n avoiding a given factor (for various types of words avoiding a pattern, see [HM09]). The notion 'correlation' was then modified to *border* in [Lot05, p. 8]. However, in the worst case there is no algorithm that can check for the occurrence

of a given factor without examining essentially all symbols of the word. A systematic construction and enumeration results for particular factor avoidance in binary case are considered in [BPP12a] for 1^j0^i-avoiders, and in [BMPP12] for $1^{j+1}0^j$-avoiders.

In Gray code context, Squire [Squ96] investigated whether there exists strong Gray code for q-ary words avoiding a given factor, so that successive words differ in only one position, and by 1 or -1 in this position. He proved such Gray codes exist if there is no parity problem induced by the employed forbidden factor. The parity of the word $\boldsymbol{w} = w_1w_2\dots w_n$ is even (resp. odd) if $\sum w_i$ is even (resp. odd). In particular, a Gray code for such set exists only if the numbers of words having even and odd parity differ by at most one. He also provided Gray codes construction for those words with q even. However, no generating algorithm was provided.

Vajnovszki [Vaj01] gave Gray codes and generating algorithm for the set of length-n binary words avoiding p consecutive 1s (which equal to the set of length n Fibonacci words of order p). The Gray codes are based on the RGC order relation, and the sets are enumerated by Fibonacci numbers of order p. The results in [Vaj01] was recently generalized in [BBPV13] where two Gray codes (one prefix partitioned and the other trace partitioned) for q-ary words avoiding a factor constituted by p consecutive equal symbols were given.

One of the variations of factor avoiding words is cross-bifix-free words (recall Definition 3). The term cross-bifix-free appeared for the first time in [Baj07]. Cross-bifix-free sets are involved in the study of distributed sequences for frame synchronization [dLvWW00]. The problem of determining the set of cross-bifix-free words is also related to several other scientific applications, for instance in pattern matching [CHL07] and automata theory [BPR09]. Moreover, in some applications, listing a cross-bifix-free set in Gray code manner can be of a particular interest.

Several methods for constructing cross-bifix-free sets have been recently proposed as in [Baj14, BPP12b, CKPW13]. For a fixed cardinality of the alphabet and of the length of the words, the construction giving the cross-bifix-free set having the largest cardinality is the one proposed in [CKPW13].

4.4/ PATTERN AVOIDING PERMUTATIONS

The study of pattern avoiding permutations began in 1968 when Knuth [Knu68] worked on the stack-sorting problem. He showed that a permutation is *stack sortable* (i.e., it can be sorted by a stack) if it avoids pattern 231; and the enumeration of such permutations for a fixed length n is given by n-th Catalan number. Early combinatorics developments to this field were given by Simion and Schmidt [SS85] and a group of mathematicians called Lothaire [Lot83]. In [SS85] was given the enumeration of permutations avoiding R, the set of patterns of length 3, for each $|R| \in \{2,\dots,5\}$. In particular, permutations avoiding either

$\{123, 132\}$, $\{123, 213\}$, $\{231, 321\}$, or $\{312, 321\}$, are enumerated by 2^{n-1} (this covers Rotem's result previously in [Rot81]); and those avoiding $\{123, 132, 213\}$ or $\{231, 312, 321\}$ are enumerated by $(n + 1)$-th Fibonacci numbers. Egge and Mansour [EM05] generalized the results in [SS85] to pattern avoiding permutations counted by k-generalized Fibonacci numbers. They showed that permutations avoiding $\{12 \dots k, 132, 213\}$, for a $k \geq 3$, are counted by $F^{(k-1)}_{(n-1)}$ (that is, $(n-1)$-th Fibonacci number of order $(p-1)$). Barcucci, Bernini, and Poneti [BBP06] showed the connections of several well-known classes of permutations having cardinality between the Fibonacci and Catalan numbers.

There are many algorithms for generating permutations. An earlier survey in 1977 by Sedgewick [Sed77] revealed about thirty of them had been published then. A natural way to permute an array of symbols efficiently is to exchange two of its symbols. The fastest permutation algorithms operate in this way. Some classical and well-known among them are that of Knuth [Knu73], Wells [Wel61], Heap [Hea63], Johnson [Joh63], and Trotter [Tro62]. The last two authors, independently gave the equivalent methods, which then widely recognized as *Steinhaus-Johnson-Trotter algorithm*; and they were the first to introduce permutation generating algorithm by adjacent transposition between symbols. The output of Steinhaus-Johnson-Trotter (SJT) algorithm is a 2-adjacent Gray code, which is the minimal Gray code for permutations. As mentioned in Section 4.2, an SJT-list is the image (under a bijection) of the list of subexcedant sequences (i.e., the inversion tables) with respect to RGC order, as shown in [Dij76].

Many studies have been made on Gray codes and generating algorithms for restricted permutations. A CAT generating algorithm for up-down permutations was given by Ruskey and van Baronaigien [vBR92], while later Korsh [Kor01] gave the loopless one. Walsh [Wal01] designed Gray codes for all length-n involutions and fixed-point free involutions. Gray code and CAT generating algorithm for derangements (permutations with no fixed points) were given by Baril and Vajnovszki [BV04]; and those for permutations with a fixed number of cycles (resp., with a fixed number of left-to-right minima) were given by Baril [Bar07] (resp., [Bar13]). Vajnovszki [Vaj03] gave Gray code and loopless generating algorithm for multiset permutations. Ko and Ruskey [KR92] gave CAT generating algorithm for *bag permutation* (a class derived from multiset permutation). Gray codes for wide classes of pattern avoiding permutations were given in [DFMV08], and the results were refined in [Bar09].

II

Our Contributions

5

GRAY CODES FOR RESTRICTED GROWTH SEQUENCES

In this chapter, we consider Gray codes for sets belonging to the class of st-restricted growth sequences (defined in Definition 1), and give efficient exhaustive generating algorithms for them. In particular, we consider four statistic st that satisfies relations (5.1) and (5.2) below.

5.1/ ADDITIONAL NOTIONS

If st is one of the statistics len, asc, m, and lv, then st satisfy the following:

$$\mathtt{st}(s_1 s_2 \dots s_n) \leq n - 1, \tag{5.1}$$

and

$$\text{if } s_n = \mathtt{st}(s_1 s_2 \dots s_{n-1}) + 1, \text{then } s_n = \mathtt{st}(s_1 s_2 \dots s_{n-1} s_n). \tag{5.2}$$

Accordingly, through this thesis we will consider only the four statistics above. However, as we will point out, some of the results presented here are also true for arbitrary statistics satisfying relations (5.1) and (5.2).

Remark 1:

If st is a statistic satisfying relations (5.1) and (5.2), then:

1. $\max\{\mathtt{st}(s_1 s_2 \dots s_n) \mid s_1 s_2 \dots s_n \text{ is an st-restricted growth sequence}\} = n - 1$;
2. if $s_1 s_2 \dots s_n$ is an st-restricted growth sequence, then for any k, $1 \leq k < n$, $s_{k+1} = \mathtt{st}(s_1 s_2 \dots s_k) + 1$ implies $s_{k+1} = \mathtt{st}(s_1 s_2 \dots s_k s_{k+1})$.

It is routine to check that the sets of st-restricted growth sequences, where st is one of the statistics len, asc, m, and lv, satisfy additional properties above. In the following we define those sets.

SUBEXCEDANT SEQUENCES

The set SE_n of *subexcedant sequences* of length n is defined as:

$$SE_n = \{s_1 s_2 \dots s_n \mid s_1 = 0 \text{ and } 0 \leq s_{k+1} \leq \texttt{len}(s_1 s_2 \dots s_k) + 1 \text{ for } 1 \leq k < n\}.$$

SE_n is trivially in bijection with the set of length n permutations; and so, it is counted by $n!$. Notice that alternatively, $SE_n = \{0\} \times \{0,1\} \times \dots \times \{0,1,\dots,n-1\}$.

ASCENT SEQUENCES

The set A_n of *ascent sequences* of length n is defined as:

$$A_n = \{s_1 s_2 \dots s_n \mid s_1 = 0 \text{ and } 0 \leq s_{k+1} \leq \texttt{asc}(s_1 s_2 \dots s_k) + 1 \text{ for } 1 \leq k < n\}.$$

A_n is in one-to-one correspondence with Stoimenow's diagrams, certain upper triangular matrices and some pattern-avoiding permutations (see Section 3.2 in [Kit11]). The generating function for the sequence counting A_n is shown in [BMCDK10] to be $\sum_{n\geq 0} \prod_{i=1}^{n} (1-(1-x)^i)$.

RESTRICTED GROWTH FUNCTIONS

The set R_n of *restricted growth functions* of length n is defined as:

$$R_n = \{s_1 s_2 \dots s_n \mid s_1 = 0 \text{ and } 0 \leq s_{k+1} \leq \texttt{m}(s_1 s_2 \dots s_k) + 1 \text{ for } 1 \leq k < n\}.$$

R_n is in bijection with the partitions of the set $\{1,2,\dots n\}$ (see for instance [RS94]) and is counted by the Bell numbers b_n, with the exponential generating function e^{e^x-1}.

STAIRCASE WORDS

The set S_n of *staircase words* of length n is defined as:

$$S_n = \{s_1 s_2 \dots s_n \mid s_1 = 0 \text{ and } 0 \leq s_{k+1} \leq \texttt{lv}(s_1 s_2 \dots s_k) + 1 \text{ for } 1 \leq k < n\}.$$

S_n is counted by Catalan numbers (see [Sta99, exercise u, p. 222]) with the generating function $\frac{1-\sqrt{1-4x}}{2x}$.

Remark 2:

$S_n \subset R_n \subset A_n \subset SE_n \subset G_n^n$, where G_n^n is the set of n-ary words of length n.

Below we give examples to illustrate Remark 2.

Example 1:

- If $s = 010145$, then $s \in SE_6$, but $s \notin A_6$, $s \notin R_6$ and $s \notin S_6$.
- If $s = 010103$, then $s \in SE_6$, $s \in A_6$, but $s \notin R_6$ and $s \notin S_6$.
- If $s = 010102$, then $s \in SE_6$, $s \in A_6$, and $s \in R_6$, but $s \notin S_6$.
- If $s = 010101$, then $s \in SE_6$, $s \in A_6$, $s \in R_6$ and $s \in S_6$.

Table 5.1 shows the sets S_5, R_5, and A_5 listed in $\prec$ order.

Sequence	S_5	R_5	A_5
0 0 0 0 0	✓	✓	✓
0 0 0 0 1	✓	✓	✓
0 0 0 1 2	✓	✓	✓
0 0 0 1 1	✓	✓	✓
0 0 0 1 0	✓	✓	✓
0 0 1 2 3	✓	✓	✓
0 0 1 2 2	✓	✓	✓
0 0 1 2 1	✓	✓	✓
0 0 1 2 0	✓	✓	✓
0 0 1 1 0	✓	✓	✓
0 0 1 1 1	✓	✓	✓
0 0 1 1 2	✓	✓	✓
0 0 1 0 2		✓	✓
0 0 1 0 1	✓	✓	✓
0 0 1 0 0	✓	✓	✓
0 1 2 3 0	✓	✓	✓
0 1 2 3 1	✓	✓	✓
0 1 2 3 2	✓	✓	✓

Sequence	S_5	R_5	A_5
0 1 2 3 3	✓	✓	✓
0 1 2 3 4	✓	✓	✓
0 1 2 2 3	✓	✓	✓
0 1 2 2 2	✓	✓	✓
0 1 2 2 1	✓	✓	✓
0 1 2 2 0	✓	✓	✓
0 1 2 1 0	✓	✓	✓
0 1 2 1 1	✓	✓	✓
0 1 2 1 2	✓	✓	✓
0 1 2 1 3		✓	✓
0 1 2 0 3		✓	✓
0 1 2 0 2		✓	✓
0 1 2 0 1	✓	✓	✓
0 1 2 0 0	✓	✓	✓
0 1 1 0 0	✓	✓	✓
0 1 1 0 1	✓	✓	✓
0 1 1 0 2		✓	✓
0 1 1 1 2	✓	✓	✓

Sequence	S_5	R_5	A_5
0 1 1 1 1	✓	✓	✓
0 1 1 1 0	✓	✓	✓
0 1 1 2 0	✓	✓	✓
0 1 1 2 1	✓	✓	✓
0 1 1 2 2	✓	✓	✓
0 1 1 2 3	✓	✓	✓
0 1 0 2 3		✓	✓
0 1 0 2 2		✓	✓
0 1 0 2 1		✓	✓
0 1 0 2 0		✓	✓
0 1 0 1 0	✓	✓	✓
0 1 0 1 1	✓	✓	✓
0 1 0 1 2	✓	✓	✓
0 1 0 1 3			✓
0 1 0 0 2		✓	✓
0 1 0 0 1	✓	✓	✓
0 1 0 0 0	✓	✓	✓

Table 5.1: The sets S_5, R_5, and A_5 listed in $\prec$ order.

We denote by

- $\mathcal{X}_n$ the list for the set X_n in $\prec$ order;
- $\mathtt{succ}_X(s)$, $s \in X_n$, the successor of s in the set X_n listed in $\prec$ order (recall Definition 7 for $\prec$ order); that is, the smallest sequence in X_n larger than s with respect to $\prec$ order;
- $\mathtt{first}(\mathcal{L})$ the first sequence in the list $\mathcal{L}$;
- $\mathtt{last}(\mathcal{L})$ the last sequence in the list $\mathcal{L}$.

Without another specification, X_n generically denotes a set of $\mathtt{st}$-restricted growth sequences, with $\mathtt{st}$ being one of the statistic $\mathtt{len}$, $\mathtt{asc}$, $\mathtt{m}$, and $\mathtt{lv}$. In other words, X_n is either the set SE_n, A_n, R_n, or S_n.

5.2/ THE REFLECTED GRAY CODE ORDER FOR THE SET X_n

For general purpose, we denote by $\mathcal{X}_n$ the lists $\mathcal{SE}_n$, $\mathcal{A}_n$, $\mathcal{R}_n$, or $\mathcal{S}_n$.

5.2.1/ THE GRAYCODENESS OF THE LIST $\mathcal{X}_n$

Lemma 1:

If $\boldsymbol{s} = s_1 s_2 \dots s_n$ and $\boldsymbol{t} = t_1 t_2 \dots t_n$ are two sequences in X_n with $\boldsymbol{t} = \mathrm{succ}_X(\boldsymbol{s})$ and k is the leftmost position where they differ, then $s_k = t_k + 1$ or $s_k = t_k - 1$.

Proof. Let $\boldsymbol{t} = \mathrm{succ}_X(\boldsymbol{s})$ and k be the leftmost position where they differ. Let us suppose that $s_k < t_k$ and $s_k \neq t_k - 1$ (the case $s_k > t_k$ and $s_k \neq t_k + 1$ being similar).
It is easy to check that

$$\boldsymbol{u} = s_1 s_2 \dots s_{k-1}(s_k + 1)0 \dots 0$$

belongs to X_n, and considering the definition of $\prec$ order relation, it follows that $\boldsymbol{s} \prec \boldsymbol{u} \prec \boldsymbol{t}$, which is in contradiction with $\boldsymbol{t} = \mathrm{succ}_X(\boldsymbol{s})$, and the statement holds. □

If $\boldsymbol{a} = a_1 a_2 \dots a_k \in X_k$, then for any $n > k$, $\boldsymbol{a}$ is the prefix of at least one sequence in X_n, and we denote by $\boldsymbol{a}\,|\,\mathcal{X}_n$ the sublist of $\mathcal{X}_n$ of all sequences having the prefix $\boldsymbol{a}$. Clearly, a list in $\prec$ order for a set of sequences is a *prefix partitioned* list (all sequences with same prefix are contiguous), and for any $\boldsymbol{a} \in X_k$ and $n > k$, it follows that $\boldsymbol{a}\,|\,\mathcal{X}_n$ is a contiguous sublist of $\mathcal{X}_n$.

For a given $\boldsymbol{a} \in X_k$, the set of all x such that $\boldsymbol{a}x \in X_{k+1}$ is called the *defining set of the prefix* $\boldsymbol{a}$, and obviously $\boldsymbol{a}x$ is also a prefix of some sequences in X_n, for any $n > k$. We denote by

$$\omega_X(\boldsymbol{a}) = \max\{x \,|\, \boldsymbol{a}x \in X_{k+1}\} \tag{5.3}$$

the largest value in the defining set of $\boldsymbol{a}$. And if we denote $\omega_X(\boldsymbol{a})$ by M, then by Remark 1 we have

$$\begin{aligned} M &= \mathtt{st}(\boldsymbol{a}) + 1 \\ &= \mathtt{st}(\boldsymbol{a}M). \end{aligned}$$

And consequently,

$$\begin{aligned} \omega_X(\boldsymbol{a}M) &= \mathtt{st}(\boldsymbol{a}M) + 1 \\ &= M + 1. \end{aligned} \tag{5.4}$$

The next two propositions give the form of $\boldsymbol{s} \in X_n$, if $\boldsymbol{s} = \mathtt{last}(\boldsymbol{a}\,|\,\mathcal{X}_n)$ or $\boldsymbol{s} = \mathtt{first}(\boldsymbol{a}\,|\,\mathcal{X}_n)$.

Proposition 1:

Let $k < n$ and $\boldsymbol{a} = a_1 a_2 \dots a_k \in X_k$. If $\boldsymbol{s} = \texttt{last}(\boldsymbol{a} \,|\, \mathcal{X}_n)$, then the form of $\boldsymbol{s}$ is given by:

- if $\sum_{i=1}^{k} a_i$ is odd, then $\boldsymbol{s} = \boldsymbol{a}0 \dots 0$;
- if $\sum_{i=1}^{k} a_i$ is even and M is odd, then $\boldsymbol{s} = \boldsymbol{a}M0 \dots 0$;
- if $\sum_{i=1}^{k} a_i$ is even and M is even, then $\boldsymbol{s} = \boldsymbol{a}M(M+1)0 \dots 0$;

where M denotes $\omega_X(\boldsymbol{a})$.

Proof. Let $\boldsymbol{s} = a_1 a_2 \dots a_k s_{k+1} \dots s_n = \texttt{last}(\boldsymbol{a} \,|\, \mathcal{X}_n)$. If $\sum_{i=1}^{k} a_i$ is odd, then by considering the definition of $\prec$ order, it follows that s_{k+1} is the smallest value in the defining set of $\boldsymbol{a}$, and so $s_{k+1} = 0$, and finally $\boldsymbol{s} = \boldsymbol{a}0 \dots 0$, and the first point holds.

Now let us suppose that $\sum_{i=1}^{k} a_i$ is even. In this case s_{k+1} equals $\omega_X(\boldsymbol{a}) = M$, the largest value in the defining set of $\boldsymbol{a}$. When in addition M is odd, so is the summation of $\boldsymbol{a}M$, the length $k+1$ prefix of $\boldsymbol{s}$, and thus $\boldsymbol{s} = \boldsymbol{a}M0 \dots 0$, and the second point holds.
Finally, when M is even, then s_{k+2} is the largest value in the defining set of $\boldsymbol{a}M$, which by relation (5.4) is $M+1$. In this case $M+1$ is odd, and thus $\boldsymbol{s} = \boldsymbol{a}M(M+1)0 \dots 0$, and the last point holds.

□

With the similar proof, the following proposition holds:

Proposition 2:

Let $k < n$ and $\boldsymbol{a} = a_1 a_2 \dots a_k \in X_k$. If $\boldsymbol{s} = \texttt{first}(\boldsymbol{a} \,|\, \mathcal{X}_n)$, then the form of $\boldsymbol{s}$ is given by:

- if $\sum_{i=1}^{k} a_i$ is even, then $\boldsymbol{s} = \boldsymbol{a}0 \dots 0$;
- if $\sum_{i=1}^{k} a_i$ is odd and M is odd, then $\boldsymbol{s} = \boldsymbol{a}M0 \dots 0$;
- if $\sum_{i=1}^{k} a_i$ is odd and M is even, then $\boldsymbol{s} = \boldsymbol{a}M(M+1)0 \dots 0$;

where M denotes $\omega_X(\boldsymbol{a})$.

By Proposition 1 and 2 above, we have the following result:

Theorem 1:

The lists $\mathcal{A}_n$, $\mathcal{R}_n$ and $\mathcal{S}_n$ are 3-adjacent Gray codes.

Proof. Let $\boldsymbol{t} = \texttt{succ}_X(\boldsymbol{s})$, and k be the leftmost position where $\boldsymbol{s}$ and $\boldsymbol{t}$ differ. Let us also denote by $\boldsymbol{a}$ the length k prefix of $\boldsymbol{s}$ and $\boldsymbol{a}'$ that of $\boldsymbol{t}$; so, $\boldsymbol{s} = \texttt{last}(\boldsymbol{a} \,|\, \mathcal{X}_n)$ and $\boldsymbol{t} =$

$\texttt{first}(\boldsymbol{a}' \,|\, \mathcal{X}_n)$. If $k+3 \leq n$, then by Proposition 1 and 2, it follows that $s_{k+3} = s_{k+4} = \cdots = s_n = 0$ and $t_{k+3} = t_{k+4} = \cdots = t_n = 0$. So $\boldsymbol{s}$ and $\boldsymbol{t}$ differ in position k, and possibly in position $k+1$ and in position $k+2$.

Now we show the adjacency, that is, if $k+2 \leq n$ and $s_{k+1} = t_{k+1}$ implies $s_{k+2} = t_{k+2}$. If $s_{k+1} = t_{k+1}$, by Lemma 1, it follows that the summation of the length k prefix of $\boldsymbol{s}$ and that of $\boldsymbol{t}$ have different parity, and two cases can occur:

- $s_{k+1} = t_{k+1} = 0$, and by Proposition 1 and 2, it follows that $s_{k+2} = t_{k+2} = 0$; or
- $s_{k+1} = t_{k+1} \neq 0$, and thus $s_{k+1} = t_{k+1} = \omega(\boldsymbol{a}) = \omega(\boldsymbol{a}')$. In this case, $\omega(\boldsymbol{a})$ either is odd and so $s_{k+2} = t_{k+2} = 0$, or is even and so $s_{k+2} = t_{k+2} = \omega(\boldsymbol{a}) + 1$.

In both cases, $s_{k+2} = t_{k+2}$.

□

It is well known that the restriction of $\mathcal{G}_q^n$ defined in relation (3.3) to any product space remains a 1-Gray code, see for example [VV11]. In particular, for $SE_n = \{0\} \times \{0,1\} \times \ldots \times \{0,1,\ldots,n-1\}$ we have the next proposition. Its proof is simply based on Lemma 1, Proposition 1 and 2, and on the additional remark: for any $\boldsymbol{a} \in SE_k$, $k < n$, it follows that $\omega_{SE}(\boldsymbol{a}) = k$.

Proposition 3:

The list $\mathcal{SE}_n$ is 1-Gray code.

It is worth mentioning that, for any statistic st satisfying relations (5.1) and (5.2), the list in $\prec$ order for the set of st-restricted growth sequences of length n is an at most 3-Gray code.

Actually, the lists $\mathcal{SE}_n$, $\mathcal{A}_n$, $\mathcal{R}_n$ and $\mathcal{S}_n$ are *circular* Gray codes, that is, the last and the first sequences in the list differ in the same way. Indeed, by the definition of $\prec$ order, it follows that:

- $\texttt{last}(\mathcal{X}_n) = 010\ldots0$;
- $\texttt{first}(\mathcal{X}_n) = 000\ldots0$.

where $\mathcal{X}_n$ is one of the lists $\mathcal{SE}_n$, $\mathcal{A}_n$, $\mathcal{R}_n$ and $\mathcal{S}_n$.

5.2.2/ GENERATING ALGORITHM FOR THE LIST $\mathcal{X}_n$

Procedure GEN1 in Figure 5.1a is a general procedure generating exhaustively the list of st-restricted growth sequences with respect to $\prec$ order. Notice that st does not necessarily satisfy relations (5.1) and (5.2). According to particular instances of the function OMEGA_X (and so, of the statistic st) called by it, GEN1 produces specific st-restricted

growth sequences, and in particular the lists $\mathcal{SE}_n$, $\mathcal{A}_n$, $\mathcal{R}_n$, and $\mathcal{S}_n$ (see Figure 5.1b for particular instances of OMEGA_X). From the length one sequence 0, GEN1 constructs recursively increasing length st-restricted growth sequences: for a given prefix $s_1 s_2 \dots s_k$ it produces all prefixes $s_1 s_2 \dots s_k i$, with i covering (in increasing or decreasing order) the defining set of $s_1 s_2 \dots s_k$; and eventually all length n st-restricted growth sequences. It has the following parameters:

- k, the position in the sequence s which is updated by the current call;
- x, belongs to the defining set of $s_1 s_2 \dots s_{k-1}$, and is the value to be assigned to s_k;
- dir, the direction (ascending for dir mod $2 = 0$ and descending for dir mod $2 = 1$) to cover the defining set of $s_1 s_2 \dots s_{k-1}$;
- v, the value of the statistic of the prefix $s_1 s_2 \dots s_{k-1}$ from which the value of the statistic of the current prefix $s_1 s_2 \dots s_k$ is computed. Remark that $v = \omega_X(s_1 s_2 \dots s_{k-1}) - 1$.

Function OMEGA_X computes $\omega_X(s_1 s_2 \dots s_k)$ (see relation (5.3)), and the main call is GEN1(1,0,0,0).

One can see that every recursive call to GEN1 will either:

- be a terminal call (and so, a sequence belongs to the set is generated), or
- produce subsequent calls.

Function OMEGA_X returns a positive integer, since $\omega_X(s_1 s_2 \dots s_k) > 0$. This means that every non-terminating calls will produce at least two subsequent calls. As long as OMEGA_X is done in $O(1)$ time, each of calls will have the amount of computations proportional to the degree of the call. In this case, GEN1 satisfies the CAT principle stated in Section 3.3.1, and so it is an efficient generating algorithm.

The computational tree of GEN1 producing the list $\mathcal{A}_4$ is given in Figure 5.2. Each node at level k, $1 \le k \le 4$, represents prefixes $s_1 s_2 \dots s_k$, and leaves sequences in $\mathcal{A}_4$.

```
 1  procedure GEN1(k, x, dir, v)
 2    global: n, s;
 3    s_k := x;
 4    if k = n then PRINT(s);
 5    else u :=OMEGA_X(v, k);
 6         if dir mod 2 = 0
 7         then for i := 0 to u do
 8                   GEN1(k + 1, i, i, u − 1);
 9         else for i := u to 0 do
10                   GEN1(k + 1, i, i + 1, u − 1);
11  end.
```

(a)

```
(i) 1  function OMEGA_SE(w, p)
    2    return p;
    3  end.

(ii) 1  function OMEGA_A(w, p)
     2    if p > 1 and s_p > s_{p−1}
     3    then return w + 2;
     4    else return w + 1;
     5  end.

(iii) 1  function OMEGA_R(w, p)
      2    if p > 1 and s_p > w
      3    then return w + 2;
      4    else return w + 1;
      5  end.

(iv) 1  function OMEGA_S(w, p)
     2    return s_p + 1;
     3  end.
```

(b)

Figure 5.1: (a) Algorithm GEN1, generating the list X_n; (b) Particular function OMEGA_X called by GEN1, and returning the value for $\omega_X(s_1 s_2 \dots s_k)$, if X_n is one of the sets: (i) SE_n, (ii) A_n, (iii) R_n, and (iv) S_n.

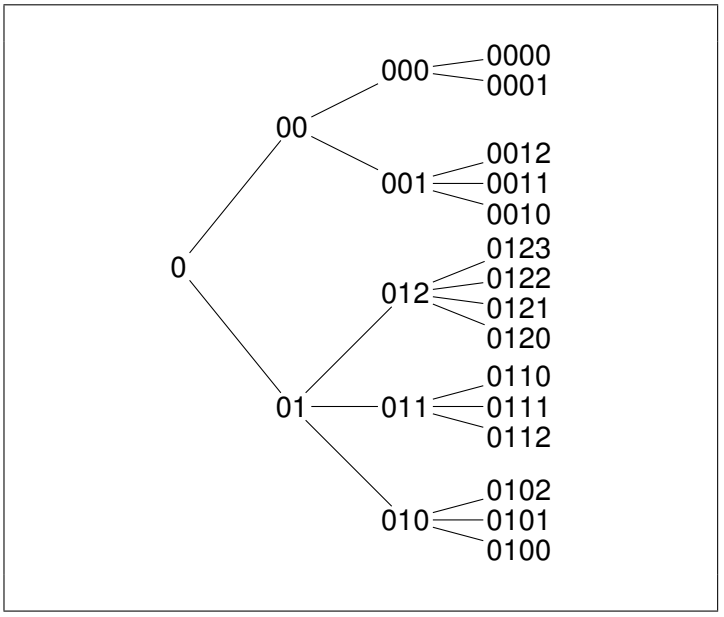

Figure 5.2: The tree induced by the initial call GEN1$(1,0,0,0)$ for $n = 4$ and particular function OMEGA_A, generating the list $\mathcal{A}_4$.

5.3/ THE CO-REFLECTED GRAY CODE ORDER FOR THE SET X_n

5.3.1/ ADDITIONAL NOTIONS

In the previous chapter, we considered RGC order, which induces RGC for the set G_q^n, namely $\mathcal{G}_q^n$. Now we give a variation of $\mathcal{G}_q^n$. Let $\boldsymbol{s} = s_1 s_2 \dots s_n$ be a sequence in G_q^n. The complement of s_i, $1 \le i \le n$, is

$$(q - 1 - s_i),$$

and the reverse of $s_1 s_2 \dots s_n$ is

$$s_n s_{n-1} \dots s_1.$$

Let $\widetilde{\mathcal{G}}_q^n$ be the list obtained by transforming each sequence $\boldsymbol{s}$ in $\mathcal{G}_q^n$ as follows:

- complementing each digit in $\boldsymbol{s}$ if q is even, or complementing only digits in odd positions if q is odd, then
- reversing the obtained sequence.

Clearly, $\widetilde{\mathcal{G}}_q^n$ is also a Gray code for G_q^n, and the sequences therein are listed in Co-Reflected Gray Code order, as defined formally below.

Table 5.2 shows the list for G_3^3 and G_4^4, with respect to RGC and Co-RGC order. The second column in each table is obtained by applying two steps of transformation above.

Rank	$\mathcal{G}_3^3$	$\widetilde{\mathcal{G}}_3^3$
0	0 0 0	2 0 2
1	0 0 1	1 0 2
2	0 0 2	0 0 2
3	0 1 2	0 1 2
4	0 1 1	1 1 2
5	0 1 0	2 1 2
6	0 2 0	2 2 2
7	0 2 1	1 2 2
8	0 2 2	0 2 2
9	1 2 2	0 2 1
10	1 2 1	1 2 1
11	1 2 0	2 2 1
12	1 1 0	2 1 1
13	1 1 1	1 1 1
14	1 1 2	0 1 1
15	1 0 2	0 0 1
16	1 0 1	1 0 1
17	1 0 0	2 0 1
18	2 0 0	2 0 0
19	2 0 1	1 0 0
20	2 0 2	0 0 0
21	2 1 2	0 1 0
22	2 1 1	1 1 0
23	2 1 0	2 1 0
24	2 2 0	2 2 0
25	2 2 1	1 2 0
26	2 2 2	0 2 0

(a)

Rank	$\mathcal{G}_4^4$	$\widetilde{\mathcal{G}}_4^4$
0	0 0 0 0	3 3 3 3
1	0 0 0 1	2 3 3 3
2	0 0 0 2	1 3 3 3
3	0 0 0 3	0 3 3 3
4	0 0 1 3	0 2 3 3
5	0 0 1 2	1 2 3 3
6	0 0 1 1	2 2 3 3
7	0 0 1 0	3 2 3 3
8	0 0 2 0	3 1 3 3
9	0 0 2 1	2 1 3 3
⋮	⋮	⋮
99	1 1 0 3	0 3 2 2
100	1 1 1 3	0 2 2 2
101	1 1 1 2	1 2 2 2
⋮	⋮	⋮
199	3 3 1 0	3 2 0 0
200	3 3 2 0	3 1 0 0
201	3 3 2 1	2 1 0 0
⋮	⋮	⋮
249	3 0 1 1	2 2 3 0
250	3 0 1 2	1 2 3 0
251	3 0 1 3	0 2 3 0
252	3 0 0 3	0 3 3 0
253	3 0 0 2	1 3 3 0
254	3 0 0 1	2 3 3 0
255	3 0 0 0	3 3 3 0

(b)

Table 5.2: The list for (a) G_3^3, and (b) G_4^4 (shown partially), with respect to RGC and Co-RGC order. The last column in each table is obtained by applying two steps of transformation: complementing only digits in odd positions for sequences in $\mathcal{G}_3^3$, or in all positions for sequences in $\mathcal{G}_4^4$, then reversing the obtained sequence.

Definition 8: Co-Reflected Gray Code order

The *Co-Reflected Gray Code order* $\prec_c$ on G_q^n is defined as:
$\boldsymbol{s} = s_1 s_2 \dots s_n$ is less than $\boldsymbol{t} = t_1 t_2 \dots t_n$, denoted by $\boldsymbol{s} \prec_c \boldsymbol{t}$, if either

- $\sum_{i=k+1}^{n} s_i + (n-k)$ is even and $s_k > t_k$, or
- $\sum_{i=k+1}^{n} s_i + (n-k)$ is odd and $s_k < t_k$,

where k is the rightmost position where $\boldsymbol{s}$ and $\boldsymbol{t}$ differ.

Although this definition sounds somewhat arbitrary, as we will see in Section 5.3.3, it

Sequence	S_5	R_5	A_5
0 1 2 3 4	✓	✓	✓
0 1 2 3 3	✓	✓	✓
0 1 0 2 3		✓	✓
0 0 1 2 3	✓	✓	✓
0 1 1 2 3	✓	✓	✓
0 1 2 2 3	✓	✓	✓
0 1 2 1 3		✓	✓
0 1 0 1 3			✓
0 1 2 0 3		✓	✓
0 1 2 0 2		✓	✓
0 1 1 0 2		✓	✓
0 0 1 0 2		✓	✓
0 1 0 0 2		✓	✓
0 1 0 1 2	✓	✓	✓
0 0 0 1 2	✓	✓	✓
0 0 1 1 2	✓	✓	✓
0 1 1 1 2	✓	✓	✓
0 1 2 1 2	✓	✓	✓

Sequence	S_5	R_5	A_5
0 1 2 2 2	✓	✓	✓
0 1 1 2 2	✓	✓	✓
0 0 1 2 2	✓	✓	✓
0 1 0 2 2		✓	✓
0 1 2 3 2	✓	✓	✓
0 1 2 3 1	✓	✓	✓
0 1 0 2 1		✓	✓
0 0 1 2 1	✓	✓	✓
0 1 1 2 1	✓	✓	✓
0 1 2 2 1	✓	✓	✓
0 1 2 1 1	✓	✓	✓
0 1 1 1 1	✓	✓	✓
0 0 1 1 1	✓	✓	✓
0 0 0 1 1	✓	✓	✓
0 1 0 1 1	✓	✓	✓
0 1 0 0 1	✓	✓	✓
0 0 0 0 1	✓	✓	✓
0 0 1 0 1	✓	✓	✓

Sequence	S_5	R_5	A_5
0 1 1 0 1	✓	✓	✓
0 1 2 0 1	✓	✓	✓
0 1 2 0 0	✓	✓	✓
0 1 1 0 0	✓	✓	✓
0 0 1 0 0	✓	✓	✓
0 0 0 0 0	✓	✓	✓
0 1 0 0 0	✓	✓	✓
0 1 0 1 0	✓	✓	✓
0 0 0 1 0	✓	✓	✓
0 0 1 1 0	✓	✓	✓
0 1 1 1 0	✓	✓	✓
0 1 2 1 0	✓	✓	✓
0 1 2 2 0	✓	✓	✓
0 1 1 2 0	✓	✓	✓
0 0 1 2 0	✓	✓	✓
0 1 0 2 0		✓	✓
0 1 2 3 0	✓	✓	✓

Table 5.3: The sets S_5, R_5, and A_5 listed in $\prec_c$ order.

turns out that $\prec_c$ order gives suffix partitioned Gray codes for some sets of restricted growth sequences. Obviously, the restriction of $\mathcal{G}_q^n$ (resp. $\widetilde{\mathcal{G}}_q^n$) to a set of sequences is simply the list of sequences in the set listed in $\prec$ (resp. $\prec_c$) order. In this section we will consider, as in the previous one, the sets SE_n, A_n, R_n and S_n, but listed in $\prec_c$ order. Our main goal is to prove that the obtained lists are Gray codes as well, and to develop generating algorithms for those sets, based on $\prec_c$ order.

Recall that X_n generically denotes one of the sets SE_n, A_n, R_n, or S_n; and let $\widetilde{\mathcal{X}}_n$ denote their corresponding list in $\prec_c$ order, that are, $\widetilde{\mathcal{SE}}_n$, $\widetilde{\mathcal{A}}_n$, $\widetilde{\mathcal{R}}_n$, or $\widetilde{\mathcal{S}}_n$. Clearly, a set of sequences listed in $\prec_c$ order is a *suffix partitioned* list, that is, all sequences with same suffix are contiguous, and such are the lists we consider here.

For a set X_n and a sequence $\boldsymbol{b} = b_{k+1}b_{k+2}\ldots b_n$, we call $\boldsymbol{b}$ an *admissible suffix* in X_n if there exists at least a sequence in X_n having suffix $\boldsymbol{b}$. For example, 124 is an admissible suffix in A_6, because there are sequences in A_6 ending with 124, namely 012124 and 010124. On the other hand, 224 is not an admissible suffix in A_6; indeed, there is no length-6 ascent sequence ending with 224.

We denote by $\widetilde{\mathcal{X}}_n \,|\, \boldsymbol{b}$ the sublist of $\widetilde{\mathcal{X}}_n$ of all sequences having suffix $\boldsymbol{b}$, and clearly, $\widetilde{\mathcal{X}}_n \,|\, \boldsymbol{b}$ is a contiguous sublist of $\widetilde{\mathcal{X}}_n$. The set of all x such that $x\boldsymbol{b}$ is also an admissible suffix in X_n is called the *defining set of the suffix* $\boldsymbol{b}$.

For $\prec$ order discussed in Section 5.2, the characterization of prefixes is straightforward: $a_1a_2\ldots a_k$ is the prefix of some sequences in X_n, $n > k$, if and only if $a_1a_2\ldots a_k$ is in X_k. And the defining set of the prefix $a_1a_2\ldots a_k$ is $\{0, 1, \ldots, \mathtt{st}(a_1a_2\ldots a_k) + 1\}$. In the case of

$\prec_c$ order, it turns out that similar notions are more complicated: for example, 13 is an admissible suffix in A_5, but 13 is not in A_2; and the defining set of the suffix 13 is $\{0, 2\}$, because 013 and 213 are both admissible suffixes in A_5, but 113 is not. See Table 5.3 for the sets A_5, R_5, and S_5 listed in $\prec_c$ order.

5.3.2/ SUFFIX EXPANSION OF SEQUENCES IN THE SET X_n

For a suffix partitioned list, we need to build `st`-restricted growth sequences under consideration from right to left, i.e., by expanding their suffix. For this purpose, we need the notions defined below.

Definition 9: $\alpha_X(\boldsymbol{b})$ and $\mu_X(\boldsymbol{b})$

Let $\boldsymbol{b} = b_{k+1}b_{k+2}\dots b_n$, $1 \le k < n$, an admissible suffix in X_n.

- $\alpha_X(\boldsymbol{b})$ is the set of all elements in the defining set of the suffix $\boldsymbol{b}$. Formally:

$$\alpha_X(\boldsymbol{b}) = \{x \,|\, x\boldsymbol{b} \text{ is an admissible suffix in } X_n\},$$

  and for the empty suffix ϵ, $\alpha_X(\epsilon) = \{0, 1, \dots, n-1\}$.

- $\mu_X(\boldsymbol{b})$ is the minimum required value of the statistic defining the set X_n, and provided by a length $(k+1)$ prefix of a sequence in X_n having suffix $\boldsymbol{b}$. Formally:

$$\mu_X(\boldsymbol{b}) = \min\{\mathtt{st}(s_1 s_2 \dots s_k b_{k+1}) \,|\, s_1 s_2 \dots s_k \boldsymbol{b} \in X_n\}.$$

Notice that $\mu_X(x\boldsymbol{b}) \in \{\mu_X(\boldsymbol{b}) - 1, \mu_X(\boldsymbol{b}), x\}$ for $x \in \alpha_X(\boldsymbol{b})$.

Remark 3:

Let `st` be one of statistics `asc`, `m` or `lv`, and $\boldsymbol{s} = s_1 s_2 \dots s_n$ be an `st`-restricted growth sequence. If there is a $k < n$ such that $s_{k+1} = k$, then $s_i = i - 1$ for all i, $1 \le i \le k$.

Proof. If $s_k < k-1$, then in each case for `st`, $\mathtt{st}(s_1 s_2 \dots s_k) < k-1$, which is in contradiction with $s_{k+1} = k$, and so $s_k = k-1$. Similarly, $s_{k-1} = k-2$, $\dots$, $s_2 = 1$, and $s_1 = 0$. □

Under the conditions in the previous remark, $s_{k+1} = k$ imposes that all values at the left of $k+1$ in $\boldsymbol{s}$ are uniquely determined. As we will see later, in the induced tree of the generating algorithm, all descendants of a node with $s_{k+1} = k$ have degree one, and we will eliminate the obtained degree-one path in order not to alter the algorithm efficiency.

It is routine to check the following propositions. (Actually, Proposition 4 is a consequence of Remark 1.)

Proposition 4:

Let X_n be one of the sets SE_n, A_n, R_n, or S_n. If $\boldsymbol{b} = b_{k+1}b_{k+2}\dots b_n$, $1 \le k < n$, is an admissible suffix in X_n, then $b_{k+1} \le \mu_X(\boldsymbol{b})$.

Proposition 5:

Let Y_n be one of the sets A_n, R_n, or S_n. If $\boldsymbol{b} = b_{k+1}b_{k+2}\dots b_n$, $1 \le k < n$, is an admissible suffix in Y_n, then

1 if $\boldsymbol{b} = b_n$, that is, a length one admissible suffix, then $\mu_Y(\boldsymbol{b}) = b_n$;

2 $\mu_Y(\boldsymbol{b}) = k$ if and only if $b_{k+1} = k$;

3 if $x\boldsymbol{b}$ is also an admissible suffix in Y_n (i.e., $x \in \alpha_Y(\boldsymbol{b})$) and $x \ge b_{k+1}$, then

$$\mu_Y(x\boldsymbol{b}) = \max\{x, \mu_Y(\boldsymbol{b})\}.$$

The following propositions give the values for $\alpha_X(\boldsymbol{b})$ and $\mu_X(x\boldsymbol{b})$, if X_n is one of the sets SE_n, A_n, R_n, or S_n. We do not provide the proofs for Propositions 6, 7, 12, and 13, because they are obviously based on the definition of the corresponding sequences.

Proposition 6:

Let $\boldsymbol{b} = \epsilon$ or $\boldsymbol{b} = b_{k+1}b_{k+2}\dots b_n$ be an admissible suffix in SE_n. Then

$$\alpha_{SE}(\boldsymbol{b}) = \begin{cases} \{0, 1, , \dots, n-1\} & \text{if } \boldsymbol{b} = \epsilon, \\ \{0, 1, \dots, k-1\} & \text{otherwise.} \end{cases}$$

Proposition 7:

Let $\boldsymbol{b} = b_{k+1}b_{k+2}\dots b_n$ be an admissible suffix in SE_n and $x \in \alpha_{SE}(\boldsymbol{b})$. Then

$$\mu_{SE}(x\boldsymbol{b}) = \mu_{SE}(\boldsymbol{b}) - 1.$$

Obviously, for a length one suffix $\boldsymbol{b} = b_n$, it follows that $\mu_{SE}(\boldsymbol{b}) = n - 1$.

Example 2:

If $\boldsymbol{b} = \epsilon$, and $n = 10$, then $\alpha_{SE}(\boldsymbol{b}) = \{0, 1, \dots, 9\}$;
and for $\boldsymbol{b} = b_{10}$, $b_{10} \in \{0, 1, \dots, 9\}$, it follows that $\mu_{SE}(x\boldsymbol{b}) = 9 - 1 = 8$, for all $x \in \alpha_{SE}(\boldsymbol{b})$.

Proposition 8:

Let $\boldsymbol{b} = \epsilon$ or $\boldsymbol{b} = b_{k+1}b_{k+2}\dots b_n$ be an admissible suffix in A_n. Then

$$\alpha_A(\boldsymbol{b}) = \begin{cases} \{0,1,\dots,n-1\} & \text{if} \quad \boldsymbol{b} = \epsilon, \\ \{k-1\} & \text{if} \quad \mu_A(\boldsymbol{b}) = k, \text{ or } \mu_A(\boldsymbol{b}) = k-1 \text{ and } b_{k+1} = 0, \\ \{0,1,\dots,b_{k+1}-1\} \cup \{k-1\} & \text{if} \quad \mu_A(\boldsymbol{b}) = k-1 \text{ and } 0 < b_{k+1} < k, \\ \{0,1,\dots,k-1\} & \text{if} \quad \mu_A(\boldsymbol{b}) < k-1. \end{cases}$$

Proof. If $\boldsymbol{b} = \epsilon$, the result is obvious.
For $\boldsymbol{b} \neq \epsilon$, let $x \in \alpha_A(\boldsymbol{b})$.
If $\mu_A(\boldsymbol{b}) = k$, by Proposition 5 point 2, $b_{k+1} = k$ and by Remark 3 we have $x = k-1$.
If $\mu_A(\boldsymbol{b}) = k-1$ and $b_{k+1} = 0$, then $\mathsf{asc}(xb_{k+1}) = 0$, and so $\mu_A(x\boldsymbol{b}) = \mu_A(\boldsymbol{b}) = k-1$, and again by Proposition 5 point 2 we have $x = k-1$.
If $\mu_A(\boldsymbol{b}) = k-1$ and $0 < b_{k+1} < k$, then there are two possibilities for $\mu_A(x\boldsymbol{b})$:

- $\mu_A(x\boldsymbol{b}) = \mu_A(\boldsymbol{b}) = k-1$, if $\mathsf{asc}(xb_{k+1}) = 0$, and as above $x = k-1$;
- $\mu_A(x\boldsymbol{b}) = \mu_A(\boldsymbol{b}) - 1 = k-2$, if $\mathsf{asc}(xb_{k+1}) = 1$. In this case $x \in \{0,1,\dots,b_{k+1}-1\}$.

If $\mu_A(\boldsymbol{b}) < k-1$ (and consequently $0 \leq b_{k+1} < k$), then there are two possibilities for $\mu_A(x\boldsymbol{b})$:

- $\mu_A(x\boldsymbol{b}) = \mu_A(\boldsymbol{b})$, if $\mathsf{asc}(xb_{k+1}) = 0$, and we have $x \in \{b_{k+1}, b_{k+1}+1, \dots, k-1\}$;
- $\mu_A(x\boldsymbol{b}) = \mu_A(\boldsymbol{b}) - 1$, if $\mathsf{asc}(xb_{k+1}) = 1$, and we have $x \in \{0,1,\dots,b_{k+1}-1\}$.

□

Proposition 9:

Let $\boldsymbol{b} = b_{k+1}b_{k+2}\dots b_n$ be an admissible suffix in A_n and $x \in \alpha_A(\boldsymbol{b})$. Then

$$\mu_A(x\boldsymbol{b}) = \begin{cases} x & \text{if} \quad x \geq \mu_A(\boldsymbol{b}), \\ \mu_A(\boldsymbol{b}) & \text{if} \quad b_{k+1} \leq x < \mu_A(\boldsymbol{b}), \\ \mu_A(\boldsymbol{b}) - 1 & \text{if} \quad x < b_{k+1}. \end{cases}$$

Proof. If $x \geq \mu_A(\boldsymbol{b})$, by Proposition 4 it follows that $x \geq b_{k+1}$, and by Proposition 5 point 3, that $\mu_A(x\boldsymbol{b}) = \max\{x, \mu_A(\boldsymbol{b})\} = x$.
If $b_{k+1} \leq x < \mu_A(\boldsymbol{b})$, then, again by Proposition 5 point 3, it follows that $\mu_A(x\boldsymbol{b}) = \max\{x, \mu_A(\boldsymbol{b})\} = \mu_A(\boldsymbol{b})$.
If $x < b_{k+1}$, then $\mathsf{asc}(xb_{k+1}) = 1$, so $\mu_A(x\boldsymbol{b}) = \mu_A(\boldsymbol{b}) - 1$. □

Example 3:

Let $k = 5$, $n = 9$, and $\boldsymbol{b} = b_6b_7b_8b_9 = 2050$ be an admissible suffix in A_9. Clearly, $\mu_A(\boldsymbol{b})$, the minimum number of ascents in a prefix $s_1s_2\dots s_5b_6$ such that $s_1s_2\dots s_5\boldsymbol{b} \in A_9$, is 4.
In this case, denoting s_5 by x, we have

- the set $\alpha_A(\boldsymbol{b})$ of all possible values for x is $\{0,1,,\dots,b_{k+1}-1\} \cup \{k-1\} = \{0,1\} \cup \{4\}$.
- $\mu_A(x\boldsymbol{b}) = \mu_A(\boldsymbol{b}) - 1 = 4 - 1 = 3$, if $x \in \{0,1\}$; or $\mu_A(x\boldsymbol{b}) = \mu_A(\boldsymbol{b}) = 4$, if $x = 4$.

Proposition 10:

Let $\boldsymbol{b} = \epsilon$ or $\boldsymbol{b} = b_{k+1}b_{k+2}\dots b_n$ be an admissible suffix in R_n. Then

$$\alpha_R(\boldsymbol{b}) = \begin{cases} \{0,1,\dots,n-1\} & \text{if} \quad \boldsymbol{b} = \epsilon, \\ \{k-1\} & \text{if} \quad \mu_R(\boldsymbol{b}) = k, \text{ or } \mu_R(\boldsymbol{b}) = k-1 \text{ and } b_{k+1} < k-1, \\ \{0,1,\dots,k-1\} & \text{if} \quad \mu_R(\boldsymbol{b}) = k-1 \text{ and } b_{k+1} = k-1, \text{ or } \mu_R(\boldsymbol{b}) < k-1. \end{cases}$$

Proof. If $\boldsymbol{b} = \epsilon$, the result is obvious.
For $\boldsymbol{b} \neq \epsilon$, let $x \in \alpha_R(\boldsymbol{b})$.
If $\mu_R(\boldsymbol{b}) = k$, by Proposition 5 point 2, $b_{k+1} = k$ and by Remark 3 we have $x = k-1$.
If $\mu_R(\boldsymbol{b}) = k-1$ and $b_{k+1} < k-1$, then the maximal value of the statistic $\mathtt{m}$ (defining the set R_n) of a length $k+1$ prefix ending with $b_{k+1} < k-1$ is $k-1$, and it is reached when $x = k-1$.
If $\mu_R(\boldsymbol{b}) = k-1$ and $b_{k+1} = k-1$, then there are two possibilities for $\mu_R(x\boldsymbol{b})$:

- $\mu_R(x\boldsymbol{b}) = \mu_R(\boldsymbol{b}) = k-1$, and as above, this implies $x = k-1$;
- $\mu_R(x\boldsymbol{b}) = \mu_R(\boldsymbol{b}) - 1 = k-2$, which implies $x \in \{0,1,\dots,k-2\}$.

Finally, if $\mu_R(\boldsymbol{b}) < k-1$, then x can be any value in $\{0,1,\dots,k-1\}$. □

Proposition 11:

Let $\boldsymbol{b} = b_{k+1}b_{k+2}\dots b_n$ be an admissible suffix in R_n and $x \in \alpha_R(\boldsymbol{b})$. Then

$$\mu_R(x\boldsymbol{b}) = \begin{cases} x & \text{if} \quad x \geq \mu_R(\boldsymbol{b}), \\ \mu_R(\boldsymbol{b}) & \text{if} \quad b_{k+1} \leq x < \mu_R(\boldsymbol{b}) \text{ or } x < b_{k+1} < \mu_R(\boldsymbol{b}), \\ \mu_R(\boldsymbol{b}) - 1 & \text{if} \quad x < b_{k+1} = \mu_R(\boldsymbol{b}). \end{cases}$$

Proof. The case $x \geq \mu_R(\boldsymbol{b})$ is analogous with the similar case in Proposition 9.
The next case is equivalent with $x < \mu_R(\boldsymbol{b})$ and $b_{k+1} < \mu_R(\boldsymbol{b})$, and since R_n corresponds to

the statistic $\mathtt{m}$, the result holds.
Finally, if $x < b_{k+1} = \mu_R(\boldsymbol{b})$, then $\mu_R(\boldsymbol{b}) = \mu_R(x\boldsymbol{b}) + 1$, and so $\mu_R(x\boldsymbol{b}) = \mu_R(\boldsymbol{b}) - 1$. □

Example 4:

Let $k = 4$, $n = 7$, and $\boldsymbol{b} = b_5 b_6 b_7 = 241$ be an admissible suffix in R_7. It follows that $b_{k+1} = 2$, $\mu_R(\boldsymbol{b}) = 3$ and

- $\alpha_R(\boldsymbol{b}) = \{k-1\} = \{3\}$;
- $\mu_R(x\boldsymbol{b}) = x = 3$.

Proposition 12:

Let $\boldsymbol{b} = \epsilon$ or $\boldsymbol{b} = b_{k+1} b_{k+2} \dots b_n$ be an admissible suffix in S_n. Then

$$\alpha_S(\boldsymbol{b}) = \begin{cases} \{0, 1, \dots, n-1\} & \text{if} \quad \boldsymbol{b} = \epsilon, \\ \{k-1\} & \text{if} \quad b_{k+1} = k, \\ \{C, C+1, \dots, k-1\} & \text{if} \quad 0 \le b_{k+1} \le k-1, \end{cases}$$

where $C = \max\{0, b_{k+1} - 1\}$.

Since $\mu_S(\boldsymbol{b}) = b_{k+1}$, the next result follows:

Proposition 13:

Let $\boldsymbol{b} = b_{k+1} b_{k+2} \dots b_n$ be an admissible suffix in S_n and $x \in \alpha_S(\boldsymbol{b})$. Then

$$\mu_S(x\boldsymbol{b}) = x.$$

Example 5:

Let $k = 6$, $n = 9$, and $\boldsymbol{b} = b_7 b_8 b_9 = 457$ be an admissible suffix in S_9. So we have $\mu_S(\boldsymbol{b}) = 4$, $C = \max\{0, b_{k+1} - 1\} = \max\{0, 3\} = 3$, and

- $\alpha_S(\boldsymbol{b}) = \{C, C+1, \dots, k-1\} = \{3, 4, 5\}$;
- $\mu_S(x\boldsymbol{b}) = x$, where $x \in \{3, 4, 5\}$.

5.3.3/ THE GRAYCODENESS OF THE LIST $\widetilde{\mathcal{X}}_n$

Now we show that the Hamming distance between two successive sequences in the mentioned lists is bounded from above by a constant, which implies that the lists are Gray codes. These results are embodied in Theorems 2, 3 and 4.

THE LIST $\widetilde{\mathcal{SE}}_n$

Theorem 2:

The list $\widetilde{\mathcal{SE}}_n$ is 1-Gray code.

Proof. The result follows from the fact that the restriction of the 1-Gray code list $\mathcal{G}_n^n$ to any product space remains a 1-Gray code (see [VV11]), in particular to the set

$$V_n = \vartheta_1 \times \vartheta_2 \times \ldots \times \vartheta_n,$$

where

- $\vartheta_i = \{0, 1, ..., n-i\}$, if n is odd and i is even, or
- $\vartheta_i = \{i-1, i, ..., n-1\}$, if n is even, or n and i are both odd.

Then by applying to each sequence $\boldsymbol{s}$ in the list $\mathcal{V}_n$ the two transforms mentioned before Definition 8, namely:

- complementing each digit in $\boldsymbol{s}$ if n is even, or only digits in odd positions if n is odd, then
- reversing the obtained sequence,

the desired 1-Gray code for the set SE_n in $\prec_c$ order is obtained. □

THE LISTS $\widetilde{\mathcal{A}}_n$ AND $\widetilde{\mathcal{R}}_n$

The next proposition describes the form of $\boldsymbol{s} = \texttt{last}(\widetilde{\mathcal{Y}}_n \mid \boldsymbol{b})$ and $\boldsymbol{s} = \texttt{first}(\widetilde{\mathcal{Y}}_n \mid \boldsymbol{b})$, where $\widetilde{\mathcal{Y}}_n$ is one of the lists $\widetilde{\mathcal{A}}_n$ or $\widetilde{\mathcal{R}}_n$.

Proposition 14:

Let Y_n be one of the sets A_n or R_n, and $\boldsymbol{b} = b_{k+1}b_{k+2}\ldots b_n$ be an admissible suffix in Y_n. If $\boldsymbol{s} = \texttt{last}(\widetilde{\mathcal{Y}}_n \mid \boldsymbol{b})$ or $\boldsymbol{s} = \texttt{first}(\widetilde{\mathcal{Y}}_n \mid \boldsymbol{b})$, then $\boldsymbol{s}$ has one of the following forms:

- $\boldsymbol{s} = 012\ldots(k-2)(k-1)\boldsymbol{b}$, or
- $\boldsymbol{s} = 012\ldots(k-2)0\boldsymbol{b}$.

Proof. Let $\boldsymbol{s} = s_1 s_2 \ldots s_k b_{k+1} b_{k+2} \ldots b_n$. Since $\boldsymbol{s} = \texttt{last}(\widetilde{\mathcal{Y}}_n \mid \boldsymbol{b})$ or $s = \texttt{first}(\widetilde{\mathcal{Y}}_n \mid \boldsymbol{b})$, according to $\alpha_Y(\boldsymbol{b})$ given in Propositions 8 and 10, it follows that $s_k \in \{0, k-1\}$. In other words, s_k

is either the smallest or the largest value in $\alpha_Y(\boldsymbol{b})$.
If $s_k = k-1$, then by Remark 3 we have $\boldsymbol{s} = 012\ldots(k-2)(k-1)\boldsymbol{b}$.

If $s_k = 0$, then considering the definition of $\prec_c$ order we have either

- $\boldsymbol{s} = \texttt{first}(\widetilde{\mathcal{Y}}_n \mid \boldsymbol{b})$ and $\sum_{i=k+1}^{n} b_i + (n-k)$ is odd, or
- $\boldsymbol{s} = \texttt{last}(\widetilde{\mathcal{Y}}_n \mid \boldsymbol{b})$ and $\sum_{i=k+1}^{n} b_i + (n-k)$ is even.

For the first case, again by the definition of $\prec_c$ order, it follows that s_{k-1} must be the largest value in $\alpha_Y(0\boldsymbol{b})$, and so $s_{k-1} = k-2$, and by Remark 3, $\boldsymbol{s} = 012\ldots(k-2)0\boldsymbol{b}$. Similarly, the same result is obtained for the second case. □

A direct consequence of the previous proposition is that $\prec_c$ order gives a more restrictive Gray codes than those given by $\prec$ order for the sets A_n and R_n. This is formalized in the next theorem.

Theorem 3:

The lists $\widetilde{\mathcal{A}}_n$ and $\widetilde{\mathcal{R}}_n$ are 2-adjacent Gray codes.

Proof. Let $\boldsymbol{s}, \boldsymbol{t} \in Y_n$, with $\boldsymbol{t} = \widetilde{\text{succ}}_Y(\boldsymbol{s})$. If $k+1$ is the rightmost position where $\boldsymbol{s}$ and $\boldsymbol{t}$ differ, then there are admissible suffixes $\boldsymbol{b} = b_{k+1}b_{k+2}\ldots b_n$ and $\boldsymbol{b}' = b'_{k+1}b_{k+2}\ldots b_n$ in Y_n such that $\boldsymbol{s} = \texttt{last}(\widetilde{\mathcal{Y}}_n \mid \boldsymbol{b})$ and $\boldsymbol{t} = \texttt{first}(\widetilde{\mathcal{Y}}_n \mid \boldsymbol{b}')$.

By Proposition 14, $\boldsymbol{s}$ has form

$$\begin{aligned} &012\ldots(k-2)(k-1)\boldsymbol{b}, \text{ or} \\ &012\ldots(k-2)0\boldsymbol{b}; \end{aligned}$$

and $\boldsymbol{t}$ has form

$$\begin{aligned} &012\ldots(k-2)(k-1)\boldsymbol{b}', \text{ or} \\ &012\ldots(k-2)0\boldsymbol{b}'. \end{aligned}$$

And in any case, $\boldsymbol{s}$ and $\boldsymbol{t}$ differ in position $k+1$ and possibly in position k. □

THE LIST $\widetilde{\mathcal{S}}_n$

The next proposition gives the form of $\texttt{last}(\widetilde{\mathcal{S}}_n \mid \boldsymbol{b})$ and $\texttt{first}(\widetilde{\mathcal{S}}_n \mid \boldsymbol{b})$ for an admissible suffix $\boldsymbol{b}$ in S_n.

Proposition 15:

Let $\boldsymbol{b} = b_{k+1}b_{k+2}\dots b_n$ be an admissible suffix in S_n. If $\boldsymbol{s} = \texttt{last}(\widetilde{\mathcal{S}}_n \mid \boldsymbol{b})$, then the form of $\boldsymbol{s}$ is given by:

- if $b_{k+1} = k$ or $\sum_{i=k+1}^{n} b_i + (n-k)$ is odd, then
$$\boldsymbol{s} = 012\dots(k-2)(k-1)\boldsymbol{b};$$
- if $b_{k+1} < k$ and $\sum_{i=k+1}^{n} b_i + (n-k)$ is even, and either $b_{k+1} = 0$ or b_{k+1} is odd, then
$$\boldsymbol{s} = 012\dots(k-2)(\max\{0, b_{k+1}-1\})\boldsymbol{b};$$
- if $b_{k+1} < k$ and $\sum_{i=k+1}^{n} b_i + (n-k)$ is even, and $b_{k+1} > 0$ is even, then
$$\boldsymbol{s} = 012\dots(k-3)(b_{k+1}-2)(b_{k+1}-1)\boldsymbol{b}.$$

Similar results hold for $\boldsymbol{s} = \texttt{first}(\widetilde{\mathcal{S}}_n \mid \boldsymbol{b})$ by replacing 'odd' by 'even', and *vice versa*, for the parity of $\sum_{i=k+1}^{n} b_i + (n-k)$.

Proof. Let $\boldsymbol{s} = s_1 s_2 \dots s_k b_{k+1} \dots b_n = \texttt{last}(\widetilde{\mathcal{S}}_n \mid \boldsymbol{b})$.
If $b_{k+1} = k$ or $\sum_{i=k+1}^{n} b_i + (n-k)$ is odd, then s_k is the largest value in $\alpha_S(\boldsymbol{b})$, so $s_k = k-1$, and by Remark 3, $s_i = i-1$ for $1 \le i \le k$. So the first case holds.
If $\sum_{i=k+1}^{n} b_i + (n-k)$ is even and $b_{k+1} < k$, then s_k is the smallest value in $\alpha_S(\boldsymbol{b})$, namely $\max\{0, b_{k+1}-1\}$, which is even if $b_{k+1} = 0$ or b_{k+1} is odd. Thus, by the definition of $\prec_c$ order, s_{k-1} is the largest value in $\alpha_S(s_k\boldsymbol{b})$, which is $k-2$, and by Remark 3, the second case holds.
For the last case, as above, $s_k = \max\{0, b_{k+1}-1\}$, and considering $b_{k+1} > 0$ and even, it follows that $s_k = b_{k+1} - 1$ is odd. Thus s_{k-1} is the minimal value in $\alpha_S(s_k\boldsymbol{b})$, that is $b_{k+1}-2$, which in turns is even, and the last case holds.
The proof for the case $\boldsymbol{s} = \texttt{first}(\widetilde{\mathcal{S}}_n \mid \boldsymbol{b})$ is similar. □

Theorem 4:

The list $\widetilde{\mathcal{S}}_n$ is 3-adjacent Gray codes.

Proof. Let $\boldsymbol{t} = \widetilde{\text{succ}}_S(\boldsymbol{s})$, and $k+1$ be the rightmost position where $\boldsymbol{s}$ and $\boldsymbol{t}$ differ. Let

us denote by $\boldsymbol{b}$ the length $(n-k)$ suffix of $\boldsymbol{s}$ and $\boldsymbol{b}'$ that of $\boldsymbol{t}$; so, $\boldsymbol{s} = \mathtt{last}(\boldsymbol{b}\,|\,\mathcal{S}_n)$ and $\boldsymbol{t} = \mathtt{first}(\boldsymbol{b}'\,|\,\mathcal{S}_n)$. It follows from Proposition 15, that $s_i = t_i = i-1$ for all $i \leq k-2$, and so the other differences possibly occur in position k and in position $k-1$.

Considering all valid combinations for $\boldsymbol{s}$ and $\boldsymbol{t}$ as given in Proposition 15, the proof of the adjacency is routine, and based on the following: $s_k \neq t_k$ if and only if $s_{k-1} \neq t_{k-1}$. It follows that $\boldsymbol{s}$ and $\boldsymbol{t}$ differ in one position, or three positions which are adjacent.

□

In addition, the lists $\widetilde{\mathcal{A}}_n$, $\widetilde{\mathcal{R}}_n$, and $\widetilde{\mathcal{S}}_n$, are circular Gray codes. This is a consequence of the following remarks based on Propositions 14 and 15:

- $\mathtt{first}(\widetilde{\mathcal{Y}}_n) = 012\ldots(n-2)(n-1)$;
- $\mathtt{last}(\widetilde{\mathcal{Y}}_n) = 012\ldots(n-2)0$;

where $\widetilde{\mathcal{Y}}_n$ is one of the lists $\widetilde{\mathcal{A}}_n$, $\widetilde{\mathcal{R}}_n$, or $\widetilde{\mathcal{S}}_n$.

5.3.4/ GENERATING ALGORITHM FOR THE LIST $\widetilde{\mathcal{X}}_n$

Here we explain algorithm GEN2 in Figure 5.3 which generates suffix partitioned Gray codes for restricted growth sequences; according to particular instances of the functions called by it, GEN2 produces the list $\widetilde{\mathcal{SE}}_n$, $\widetilde{\mathcal{A}}_n$, $\widetilde{\mathcal{R}}_n$, or $\widetilde{\mathcal{S}}_n$. Actually, for convenience, GEN2 produces length $(n+1)$ sequences $\boldsymbol{s} = s_1 s_2 \ldots s_{n+1}$ with $s_{n+1} = 0$, and so, neglecting the last value in each sequence $\boldsymbol{s}$ the desired list is obtained. Notice that with this dummy value for s_{n+1} we have $\mu_X(s_k s_{k+1}\ldots s_n) = \mu_X(s_k s_{k+1}\ldots s_n 0)$, for $k \leq n$, and similarly for α_X.

In GEN2, the sequence $\boldsymbol{s}$ is a global variable, and initialized by $01\ldots(n-1)0$, which is the first length n sequence in $\prec_c$ order, followed by a 0; and the main call is GEN2$(n+1,0,0,0)$. Procedure GEN2 has the following parameters (the first three of them are similar with those of procedure GEN1):

- k, the position in the sequence $\boldsymbol{s}$ which is updated by the current call;
- x, the value to be assigned to s_k;
- dir, gives the direction in which s_{k-1} covers $\alpha_X(s_k s_{k+1}\ldots s_n 0)$, the defining set of the current suffix;
- v, the value of $\mu_X(s_{k+1}\ldots s_n 0)$.

The functions called by GEN2 are given in Figures 5.4 and 5.5. They are principally based on the evaluation of α_X and μ_X for the current suffix of $\boldsymbol{s}$, and are as follows:

- MU_$X(k,x,v)$ returns the value of $\mu_X(s_k s_{k+1} \dots s_n 0)$, with $x = s_k$.
- ISDEGREEONE_$X(k)$ stops the recursive calls when $\alpha_X(s_k s_{k+1} \dots s_n 0)$ has only one element, namely $k-2$. In this case, by Remark 3 the sequence is uniquely determined by the current suffix, and this prevents GEN2 to produce degree one calls. In addition, ISDEGREEONE_$X(k)$ sets appropriately $d-1$ values at the left of s_k, where d is the upper bound of the Hamming distance in the list (changes at the left of s_1 are considered with no effect). This can be considered as a *Path Elimination Technique* or PET (see [Rus03]).
- LOWEST_$X(k)$ is called when ISDEGREEONE_X returns false, and gives the lowest value in $\alpha_X(s_k s_{k+1} \dots s_n 0)$.
- SECLARGEST_$X(k,u)$, is called when ISDEGREEONE_X returns false, and gives the second largest value in $\alpha_X(s_k s_{k+1} \dots s_n 0)$ (the largest value being always $k-2$).

By this construction, algorithm GEN2 has no degree one calls and it satisfies the CAT principle. Figure 5.6 shows the tree induced by the algorithm when generates $\widetilde{\mathcal{A}}_4$.

```
 1  procedure GEN2(k, x, dir, v)
 2  global n,s;
 3  s_k := x;
 4  u :=MU_X(k, x, v);
 5  if ISDEGREEONE_X(k, u) then PRINT(s);
 6  else c:=LOWEST_X(k);
 7       d:=SECLARGEST_X(k, u);
 8       if dir mod 2 = 1 then
 9         for i := c to d do
10             GEN2(k − 1, i, i, u);
11         GEN2(k − 1, k − 2, k − 1 − (dir mod 2), u);
12       if dir mod 2 = 0 then
13         for i := d downto c do
14             GEN2(k − 1, i, i + 1, u);
15  end.
```

Figure 5.3: Algorithm GEN2, generating the list $\widetilde{\mathcal{X}}_n$.

```
1  function MU_A(m, i, w)
2    if i ≥ w then return i;
3    else if i ≥ s_{m+1}
4         then return w;
5         else return w − 1;
6  end.

7  function ISDEGREEONE_A(m, v)
8    if v = m − 1 or
9      (v = m − 2 and s_m = 0)
10   then s_{m−1} := m − 2;
11        return true;
12   else return false;
13 end.

14 function LOWEST_A(m)
15   return 0;
16 end.

17 function SECLARGEST_A(m, w)
18   if w = m − 2 and s_m > 0
19     and s_m < m − 1
20   then return s_m − 1;
21   else return m − 3;
22 end.
```

(a)

```
1  function MU_R(m, i, w)
2    if i ≥ w then return i;
3    else if s_{m+1} < w
4         then return w;
5         else return w − 1;
6  end.

7  function ISDEGREEONE_R(m, v)
8    if v = m − 1 or
9      (v = m − 2 and s_m < m − 2)
10   then s_{m−1} := m − 2;
11        return true;
12   else return false;
13 end.

14 function LOWEST_R(m)
15   return 0;
16 end.

17 function SECLARGEST_R(m, w)
18   return m − 3;
19 end.
```

(b)

Figure 5.4: Particular functions called by GEN2, generating the lists: (a) $\widetilde{\mathcal{A}}_n$, and (b) $\widetilde{\mathcal{R}}_n$.

```
1  function MU_S(m, i, w)
2    return i;
3  end.

4  function ISDEGREEONE_S(m, v)
5    if s_m = m − 1
6    then s_{m−1} := m − 2;
7         s_{m−2} := m − 3;
8         return true;
9    else return false;
10 end.

11 function LOWEST_S(m)
12   if s_m > 1 and s_m ≤ m − 2
13   then return s_m − 1;
14   else return 0;
15 end.

16 function SECLARGEST_S(m, w)
17   return m − 3;
18 end.
```

(a)

```
1  function MU_SE(m, i, w)
2    if m = n then return n − 1;
3    else return w − 1;
4  end.

5  function ISDEGREEONE_SE(m, v)
6    if m = 2 then return true;
7    else return false;
8  end.

9  function LOWEST_SE(m)
10   return 0;
11 end.

12 function SECLARGEST_SE(m, w)
13   return m − 3;
14 end.
```

(b)

Figure 5.5: Particular functions called by GEN2, generating the lists: (a) $\widetilde{\mathcal{S}}_n$, and (b) $\widetilde{\mathcal{SE}}_n$.

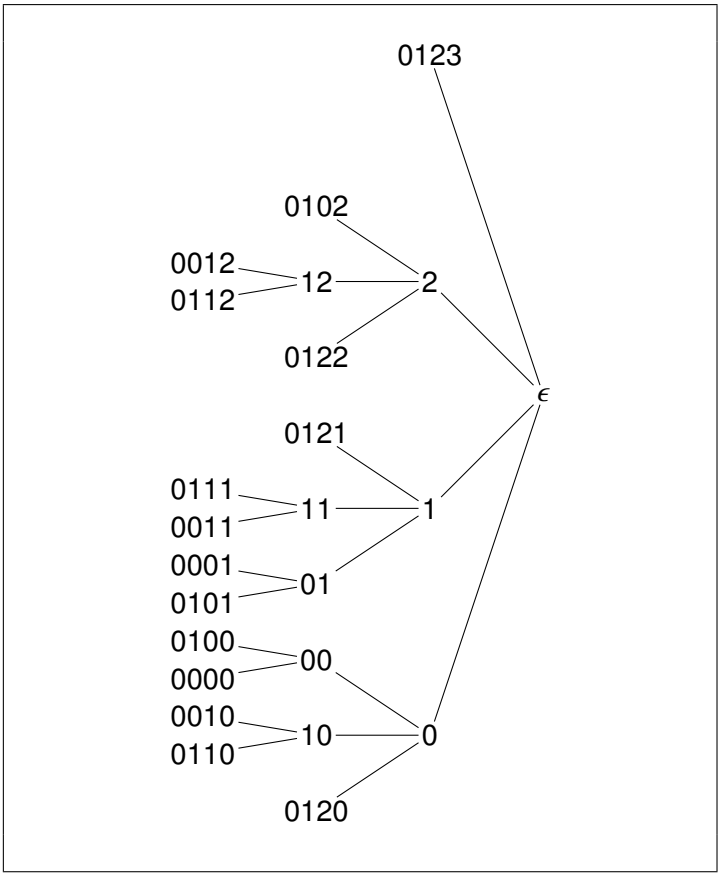

Figure 5.6: The generating tree induced by initial call GEN2$(1, 0, 0, 0)$, in this case to generate $\widetilde{\mathcal{A}}_4$.

5.4/ ADDITIONAL RESULT: GRAY CODES FOR RESTRICTED ASCENT SEQUENCES

The study about restricted ascent sequences was initiated by Duncan and Steingrímsson [DS11]. They investigate the cardinality of the set A_n avoiding pattern T, denoted by $A_n(T)$, where T is one of the patterns 001, 010, 011, 012, 102, 101, 021, 000, 100, 110, 120, 201, 210, 0102, 0112, 0101, 0123, 0021, or 1012. Their conjectures for the cardinality of $A_n(1012)$ and $A_n(0123)$ were then affirmed by Mansour and Shattuck [MS14], and later of $A_n(0012)$ were affirmed by Pudwell [Pud14]. However, none of them provide Gray code for $A_n(T)$.

Here we show some of those restrictions still yield Gray code with respect to $\prec$ order. We point out that these restrictions can cancel the reflectable property (see [LS09]), and this implies Gray code construction provided by [LS09] and [Wal08] might not work for $A_n(T)$.

Notice that, for convenience, the patterns use non-negative integers since ascent sequence is defined over non-negative integers. This causes a slight different representation with that for permutations (discussed in Chapter 6), which uses positive integers. For example, 012 represents the same pattern with 123, as well as 201 with 312.

Theorem 5:

Let $T \in \{011, 101, 021, 201\}$. Then, the list $\mathcal{A}_n(T)$ is a 3-adjacent Gray code.

Proof. Let $\boldsymbol{t} = \mathtt{succ}_A(\boldsymbol{s})$, and k be the leftmost position where $\boldsymbol{s}$ and $\boldsymbol{t}$ differ. Let $\boldsymbol{a}$ be the length k prefix of $\boldsymbol{s}$ and $\boldsymbol{a}'$ that of $\boldsymbol{t}$; so, $\boldsymbol{s} = \mathtt{last}(\boldsymbol{a} \,|\, \mathcal{A}_n(T))$ and $\boldsymbol{t} = \mathtt{first}(\boldsymbol{a}' \,|\, \mathcal{A}_n(T))$.

Let $M = \omega(\boldsymbol{a})$ (recall relation 5.3). Let also $T' \in \{01, 10, 02, 20\}$, that is, a pattern obtained by removing the last symbol of pattern T. If M is odd (resp. even), $\boldsymbol{a}$ contains pattern T' and $\sum_{i=1}^{k} a_i$ is even, then $\boldsymbol{s} = \boldsymbol{a}M0\ldots0$ (resp. $\boldsymbol{s} = \boldsymbol{a}M(M+1)0\ldots0$), since M is larger than any symbol in $\boldsymbol{a}$, so $\boldsymbol{s}$ does not contain pattern T.

If $\boldsymbol{a}'$ contains pattern T', one of the following two cases holds:

- If $\sum_{i=1}^{k-1} a_i + a'_k$ is even, then $\boldsymbol{t} = \boldsymbol{a}'0\ldots0$.
- If $\sum_{i=1}^{k-1} a_i + a'_k$ is odd, then $\boldsymbol{t} = \boldsymbol{a}'M'0\ldots0$ (resp. $\boldsymbol{t} = \boldsymbol{a}'M'(M'+1)0\ldots0$), where $M' = \omega(\boldsymbol{a}')$.

It is easy to check that in both cases, $\boldsymbol{t}$ does not contain pattern T.

Any possible pairings of $\boldsymbol{s}$ and $\boldsymbol{t}$ differ in at most 3 adjacent position.

The proof is similar for $\sum_{i=1}^{k} a_i$ being odd.

□

Theorem 6:

The list $\mathcal{A}_n(012)$ is a 1-Gray code.

Proof. All ascents in $A_n(012)$ are only created by 01. It is easy to see that $\mathcal{A}_n(012)$ is the BRGC of length n that begins by 0. □

5.5/ SUMMARY

We conclude this chapter by comparing for each of the sets SE_n, A_n, R_n and S_n the prefix partitioned Gray codes induced by $\prec$ order and the suffix partitioned one induced by $\prec_c$ order. This can be done by comparing the Hamming distance between all pairs of successive sequences either in the worst case or in average.

Table 5.4 summarizes Theorems 1, 3, and 4, and Proposition 3, and gives the upper bound of the Hamming distance (that is, the worst case Hamming distance) for the two order relations. It shows that for the sets SE_n and S_n these relations have same performances, and for the sets A_n and R_n, $\prec_c$ order induces more restrictive Gray codes.

Set	$\prec$ (RGC) order	$\prec_c$ (Co-RGC) order
SE_n	1	1
A_n	3	2
R_n	3	2
S_n	3	3

Table 5.4: The bound of the Hamming distance between two successive sequences in $\prec$ and $\prec_c$ orders.

For a list $\mathcal{L}$ of sequences, the *average Hamming distance* is defined as

$$\frac{\sum d(\boldsymbol{s}, \boldsymbol{t})}{N-1},$$

where the summation is taken over all $\boldsymbol{s}$ in $\mathcal{L}$, except its last element, $\boldsymbol{t}$ is the successor of $\boldsymbol{s}$ in $\mathcal{L}$, d is the Hamming distance, and N the number of sequences in $\mathcal{L}$.

Surprisingly, despite $\prec_c$ order has same or better performances in terms of worst case Hamming distance, if we consider the average Hamming distance, numerical evidences show that $\prec$ order is 'more optimal' than $\prec_c$ order on A_n, R_n ($n \geq 5$), and S_n ($n \geq 6$). And this phenomenon strengthens for large n; see Table 5.5.

Algorithmically, $\prec_c$ order has the advantage that its corresponding generating algorithm, GEN2, is more appropriate to be parallelized than its $\prec$ order counterpart, GEN1. Indeed, the main call of GEN2 produces n recursive calls (compared to two recursive calls produced by the main call of GEN1), and so we can have more parallelized computations;

n	$\prec$ (RGC) order				$\prec_c$ (Co-RGC) order			
	$\mathcal{SE}_n$	$\mathcal{A}_n$	$\mathcal{R}_n$	$\mathcal{S}_n$	$\widetilde{\mathcal{SE}}_n$	$\widetilde{\mathcal{A}}_n$	$\widetilde{\mathcal{R}}_n$	$\widetilde{\mathcal{S}}_n$
4	1	1.21	1.21	1.31	1	1.14	1.14	1.15
5	1	1.13	1.12	1.29	1	1.19	1.18	1.24
6	1	1.09	1.07	1.27	1	1.23	1.20	1.31
7	1	1.06	1.06	1.26	1	1.25	1.22	1.35
8	1	1.04	1.04	1.25	1	1.26	1.23	1.37
9	1	1.03	1.03	1.24	1	1.28	1.24	1.39
10	1	1.02	1.03	1.23	1	1.28	1.24	1.41

Table 5.5: The average Hamming distance for $\prec$ order and $\prec_c$ order.

and this is more suitable for large n. See Figure 5.2 and 5.6 for examples of computational trees.

It will be of interest to explore order relation based Gray codes for restricted growth sequences defined by statistics other than those considered in this paper. In this vein we suggest the following conjecture, checked by computer for $n \leq 10$, and concerning descent sequences (defined similarly with ascent sequences in Section 2).

Conjecture 1:

The set of length n descent sequences listed in $\prec_c$ order is a 4-adjacent Gray code.

Finally, we extend the result of Duncan and Steingrímsson [DS11] concerning ascent sequences avoiding a patterrn of length 3 or 4. We investigate the Graycodeness of $\mathcal{A}_n(T)$, and show that $\mathcal{A}_n(T)$ is a 1-Gray code if $T = \{012\}$, and is a 3-Gray code if $T \in \{011, 101, 021, 201\}$.

6

GRAY CODES FOR FACTOR AVOIDING q-ARY WORDS

6.1/ ADDITIONAL NOTIONS

Through this chapter, we use the notions defined in Section 3.1.1, and some additional ones defined here. Recall that the set A^n is the set of words of length n over alphabet A. For $\boldsymbol{a} \in A^*$, $|\boldsymbol{a}|$ denotes the length of $\boldsymbol{a}$, or equivalently, the number of symbols in $\boldsymbol{a}$; and when $0 \in A$, $|\boldsymbol{a}|_{\neq 0}$ denotes the number of non-zero symbols in $\boldsymbol{a}$ (the alphabets we will be using contain the symbol 0).

An *infinite word* is a function $\mathbb{N} \to A$, and A^∞ is the set of infinite words. For $\boldsymbol{a} \in A^*$, $\boldsymbol{a}^i$ is the word obtained by i copies of $\boldsymbol{a}$, with $i = 0$ denotes ϵ, and $\boldsymbol{a}^\infty$ is the infinite *periodic* word $\boldsymbol{aaa}\ldots$. The word $\boldsymbol{a} \in A^\infty$ is *ultimately periodic* if there are $\boldsymbol{b}, \boldsymbol{c} \in A^*$ such that $\boldsymbol{a} = \boldsymbol{bc}^\infty$, and we say that $\boldsymbol{a}$ has *ultimate period* $\boldsymbol{c}$. Incidentally, we will make use of *left infinite words*, which are infinite words $\boldsymbol{a}$ of the form $\boldsymbol{a} = \ldots a_{-3}a_{-2}a_{-1}$. Left infinite words are the reverse of infinite words, and formally a left infinite word is a function $\{\ldots, -3, -2, -1\} \to A$, and for $\boldsymbol{a} \in A^*$, $\boldsymbol{a}^{-\infty}$ is the left infinite word $\ldots \boldsymbol{aaa}$.

The set A_q^n is the set of words of length n over alphabet $A = \{0, 1, \ldots q-1\}$. Our constructions of Gray codes for factor avoiding q-words are based on two order relations on A_q^n. One of them is the original RGC order, already introduced in Definition 7; and the other one is a variation of it, as we will define below.

It follows that A_q^n listed in $\prec$ order yields precisely the q-ary Reflected Gray Code (see [Er84, Wil85]), where two consecutive words differ in one position and by 1 or -1 in this position. For a set of same length words X, we refer to $\prec$-*first* (resp., $\prec$-*last*) word in X for the first (resp., last) word in X with respect to $\prec$ order.

In the binary case ($q = 2$), the Reflected Gray Code order can alternatively be defined as follows (see Definition 7 for the original definition): Let $\boldsymbol{s} = s_1 s_2 \ldots s_n$, $\boldsymbol{t} = t_1 t_2 \ldots t_n$ be two words in A_2^n, and k is the leftmost position where $\boldsymbol{s}, \boldsymbol{t}$ differ. Let also v be the number of

non-zero symbols in the length $k-1$ prefix of $\boldsymbol{s}$ and of $\boldsymbol{t}$. Then $\boldsymbol{s} \prec \boldsymbol{t}$ if either

- v is even and $s_k < t_k$, or
- v is odd and $s_k > t_k$.

Now, by 'adding' u in Definition 7 and v defined above, we obtain a new order relation as follows:

Definition 10:

Let $\boldsymbol{s} = s_1 s_2 \dots s_n$ and $\boldsymbol{t} = t_1 t_2 \dots t_n$ be two words in A_q^n, k be the leftmost position where $\boldsymbol{s}, \boldsymbol{t}$ differ, $u = \sum_{i=1}^{k-1} s_i = \sum_{i=1}^{k-1} t_i$, and v be the number of non-zero symbols in the length $k-1$ prefix of $\boldsymbol{s}$. We say that $\boldsymbol{s}$ is *less than* $\boldsymbol{t}$ in *Dual Reflected Gray Code order*, denoted by $\boldsymbol{s} \lhd \boldsymbol{t}$, if either

- $u+v$ is even and $s_k < t_k$, or
- $u+v$ is odd and $s_k > t_k$.

Clearly, listing a set of words in $\prec$ or in $\lhd$ order gives a prefix partitioned list, in the sense that words with the same prefix are contiguous.

For a set of same length words X, we refer to $\lhd$-*first* (resp., $\lhd$-*last*) word in X for the first (resp., last) word in X with respect to $\lhd$ order. In the following, unless the contrary is explicitly stated otherwise, we will consider $\prec$ order on A_q^* when q is even and $\lhd$ order when q is odd.

The next remark states that $\lhd$ order yields a Gray code when q is odd, but generally for q even, A_q^n listed in $\lhd$ order is not a Gray code.

Remark 4:

For any odd $q \geq 3$ and $n \geq 1$, the set A_q^n listed in $\lhd$ order is a Gray code where two consecutive words differ in at most two adjacent positions. In addition, if $\boldsymbol{s}$ and $\boldsymbol{t}$ are two consecutive words in this list and k is the leftmost position where they differ, then

- $s_k = t_k \pm 1$, and
- if $s_{k+1} \neq t_{k+1}$, then $\{s_{k+1}, t_{k+1}\} = \{0, q-1\}$.

Table 6.1 shows the list in $\lhd$ order, for the sets A_3^3 and A_4^4. One can conclude that the list for A_3^3 is a 2-Gray code, but for A_4^4 is not a Gray code.

We define the *parity* of the word $\boldsymbol{a} = a_1 a_2 \dots a_n \in A_q^n$ according to two cases:

- when q is even, the parity of $\boldsymbol{a}$ is the parity of the integer $\sum_{i=1}^{n} a_i$, and

- when q is odd, the parity of $\boldsymbol{a}$ is the parity of the integer $\sum_{i=1}^{n} a_i + |\boldsymbol{a}|_{\neq 0}$.

For example, if $\boldsymbol{a} = 0222$, then

- considering $\boldsymbol{a} \in A_4^*$, the parity of $\boldsymbol{a}$ is even, and is given by $0+2+2+2=6$, and
- considering $\boldsymbol{a} \in A_5^*$, the parity of $\boldsymbol{a}$ is odd, and is given by $0+2+2+2+3=9$.

Now, we introduce a critical concept for our purposes: we say that the forbidden factor $\boldsymbol{f} \in A_q^*$ *induces zero periodicity* on A_q^∞ if for any $\boldsymbol{p} \in A_q^*(\boldsymbol{f})$ the first and the last (as mentioned previously, with respect to $\prec$ order for even q, or $\lhd$ order for odd q) words in $\boldsymbol{p}|A_q^\infty(\boldsymbol{f})$ both have ultimate period 0. Consequently, if $\boldsymbol{f} \in A_q^*$ *does not* induce zero periodicity on A_q^∞, it follows that there exists a $\boldsymbol{p} \in A_q^*(\boldsymbol{f})$ such that the first and/or the last word in $\boldsymbol{p}|A_q^\infty(\boldsymbol{f})$ do not have ultimate period 0.

Whether or not $\boldsymbol{f}$ induces zero periodicity on A_q^∞ depends on q; and q will often be understood from the context. For example, $\boldsymbol{f} = 3130$ induces zero periodicity on A_6^∞ but not on A_4^∞. Indeed, the first word in $\prec$ order in $A_6^\infty(\boldsymbol{f})$ and having prefix 313 is $3135000\ldots$, and the last one is $3131500\ldots$; whereas the last word in $\prec$ order in $A_4^\infty(\boldsymbol{f})$ and with the same prefix is the periodic word $31313\ldots$.

In Section 6.3, it is shown that the *Graycodeness* of the set $A_q^n(\boldsymbol{f})$ listed in the appropriate order is intimately related to the property that $\boldsymbol{f}$ induces zero periodicity.

OUTLINE OF THE CHAPTER

Avoiding a factor of length 1 is equivalent to shrinking the underlying alphabet, but for the sake of generality, we will consider most of the time forbidden factors of any positive length.

In the next section, we will characterize the forbidden factors $\boldsymbol{f}$ inducing zero periodicity on A_q^∞. By means of three sets $U_q, V, W_q \subset A_q^+$, these factors are characterized in Corollary 1. The non-zero ultimate periods produced by forbidden factors that do not induce zero periodicity have the form $1(q-1)0^m$, 10^m or $(q-2)$; see Propositions 16, 18 and 20.

Theorems 10 to 12 and Proposition 27 in Section 6.3 prove that the property of $\boldsymbol{f}$ to induce zero periodicity guarantees the set $A_q^n(\boldsymbol{f})$ listed in the appropriate order to be a Gray code. However, this property of $\boldsymbol{f}$ is not a necessary condition: there are two 'special' forbidden factors, namely $\boldsymbol{f} = 0^\ell$ and $\boldsymbol{f} = (q-1)0^\ell$ belonging to U_q (and thus they do not induce zero periodicity) but $A_q^n(\boldsymbol{f})$ listed in $\prec$ order is still a Gray code. These two cases are discussed in Section 6.3.2. Table 6.2 summarizes the Graycodeness for the set $A_q^n(\boldsymbol{f})$ if $\boldsymbol{f}$ does not induce zero periodicity. Section 6.3 ends by showing that simple transformation of forbidden factors $\boldsymbol{f}$, which do not induce zero periodicity, make it possible to obtain Gray code for the set $A_q^n(\boldsymbol{f})$.

Finally, we present in Section 6.4 an efficient generating algorithm for the obtained Gray codes.

6.2/ PERIODICITY

As stated above, without restriction, the set A_q^n listed in $\prec$ order is a 1-Gray code for any $q \geq 2$, and listed in $\lhd$ order is a 2-Gray code for odd $q \geq 3$. Roughly, it is due to the fact that for any $\boldsymbol{p} \in A_q^*$ the first and the last words—with respect to $\prec$ order for any $q \geq 2$, or $\lhd$ order for odd $q \geq 3$— in the set $\boldsymbol{p}|A_q^\infty$ are among the three infinite words: $\boldsymbol{p}0^\infty$, $\boldsymbol{p}(q-1)0^\infty$, and $\boldsymbol{p}(q-1)^\infty$. This phenomenon is no longer true if an arbitrary factor $\boldsymbol{f}$ is forbidden. For example, if $\boldsymbol{f} = 130$, then the $\prec$-last word in $03|A_4^\infty(\boldsymbol{f})$ is $0300000\ldots$, and the $\prec$-first one in $13|A_4^\infty(\boldsymbol{f})$ is $1313131\ldots$; and 0300000 and 1313131 are consecutive words in $A_4^7(\boldsymbol{f})$, in $\prec$ order. Or, for $\boldsymbol{f} = 223$, the $\prec$-last word in $123|A_4^\infty(\boldsymbol{f})$ is $123300\ldots$ and the $\prec$-first one in $122|A_4^\infty(\boldsymbol{f})$ is $122222\ldots$; and 123300 and 122222 are consecutive words in $A_4^6(\boldsymbol{f})$, in $\prec$ order.

However, the next remark is easy to understand.

Remark 5:

If $\boldsymbol{f} \in A_q^*$ is a forbidden factor ending with a symbol other than 0 or $q-1$, then for any $\boldsymbol{p} \in A_q^*(\boldsymbol{f})$, $q \geq 2$ and even (resp., $q \geq 3$ and odd), the set formed by the $\prec$-first and the $\prec$-last (resp., the $\lhd$-first and the $\lhd$-last) words in $\boldsymbol{p}|A_q^\infty(\boldsymbol{f})$ is $\{\boldsymbol{p}0^\infty, \boldsymbol{p}(q-1)0^\infty\}$.

In other words, the previous remark says that any factor ending with a symbol other than 0 or $q-1$ induces zero periodicity. However, there exist forbidden factors ending with 0 or $q-1$ that do induce zero periodicity. For example, with $\boldsymbol{f} = 120$, the $\prec$-first and $\prec$-last words in $12|A_4^\infty(\boldsymbol{f})$ are $1230000\ldots$ and $1213000\ldots$.

In what follows, we will use (often implicitly) the next straightforward remark which provides the form of the words on the right of a fixed prefix $\boldsymbol{p} \in A_q^*(\boldsymbol{f})$, with respect to the appropriate order. It is obtained from the following observation: the first/last word in $\boldsymbol{p}|A_q^n(\boldsymbol{f})$ is the appropriate prefix of the first/last word in $\boldsymbol{p}|A_q^\infty(\boldsymbol{f})$.

Rank	A_3^3, $\lhd$ order	Hamming distance
0	0 0 0	0
1	0 0 1	1
2	0 0 2	1
3	0 1 0	2
4	0 1 1	1
5	0 1 2	1
6	0 2 2	1
7	0 2 1	1
8	0 2 0	1
9	1 0 0	2
10	1 0 1	1
11	1 0 2	1
12	1 1 0	2
13	1 1 1	1
14	1 1 2	1
15	1 2 2	1
16	1 2 1	1
17	1 2 0	1
18	2 2 0	1
19	2 2 1	1
20	2 2 2	1
21	2 1 2	1
22	2 1 1	1
23	2 1 0	1
24	2 0 2	2
25	2 0 1	1
26	2 0 0	1

(a)

Rank	A_4^4, $\lhd$ order	Hamming distance
0	0 0 0 0	0
1	0 0 0 1	1
2	0 0 0 2	1
3	0 0 0 3	1
4	0 0 1 0	2
5	0 0 1 1	1
6	0 0 1 2	1
7	0 0 1 3	1
8	0 0 2 3	1
9	0 0 2 2	1
⋮	⋮	⋮
15	0 0 3 3	1
16	0 1 0 0	3
⋮	⋮	⋮
63	0 3 3 3	1
64	1 0 0 0	4
⋮	⋮	⋮
247	3 3 1 3	1
248	3 3 2 3	1
249	3 3 2 2	1
250	3 3 2 1	1
251	3 3 2 0	1
252	3 3 3 0	1
253	3 3 3 1	1
254	3 3 3 2	1
255	3 3 3 3	1

(b)

Table 6.1: The list of words in (a) A_3^3, and (b) A_4^4 (shown partially), with respect to Dual RGC order. The Hamming distance column shows the number of differences between a particular sequence and its predecessor. In general, the list for A_q^n in $\lhd$ order is a 2-Gray code for odd q, but it is not Gray code for even q.

Remark 6:

Let q be even, $\boldsymbol{f} \in A_q^*$ be a forbidden factor, $\boldsymbol{p}, \boldsymbol{r} \in A_q^*(\boldsymbol{f})$, and let $\boldsymbol{p}$ have even (resp., odd) parity such that $\boldsymbol{pr} \in A_q^*(\boldsymbol{f})$. Then:

- If $\boldsymbol{pr}$ is a prefix of the $\prec$-first (resp., $\prec$-last) word in $\boldsymbol{p}|A_q^\infty(\boldsymbol{f})$, then $\boldsymbol{r}$ is the smallest word, in $\prec$ order, with this property; that is, if $\boldsymbol{s} \in A_q^*$ with $|\boldsymbol{s}| = |\boldsymbol{r}|$ and $\boldsymbol{s} \neq \boldsymbol{r}$ is such that $\boldsymbol{ps}$ is the prefix of some word in $\boldsymbol{p}|A_q^\infty(\boldsymbol{f})$, then $\boldsymbol{r} \prec \boldsymbol{s}$.
- If $\boldsymbol{pr}$ is a prefix of the $\prec$-last (resp., $\prec$-first) word in $\boldsymbol{p}|A_q^\infty(\boldsymbol{f})$, then $\boldsymbol{r}$ is the largest word, in $\prec$ order, with this property; that is, if $\boldsymbol{s} \in A_q^*$ with $|\boldsymbol{s}| = |\boldsymbol{r}|$ and $\boldsymbol{s} \neq \boldsymbol{r}$ is such that $\boldsymbol{ps}$ is the prefix of some word in $\boldsymbol{p}|A_q^\infty(\boldsymbol{f})$, then $\boldsymbol{s} \prec \boldsymbol{r}$.

Similar results hold when q is odd by replacing $\prec$ by $\lhd$ and considering the words parity as in the definition given after Remark 4.

Remark 7 below specifies the form of the first and last words in A_q^n, subject to the additional constraint that they do not begin by 0 or $q-1$. Later on we will see that when $\boldsymbol{f}$ does not induce zero periodicity, then the possible non-zero periods of the first or the last word in $\boldsymbol{p}|A_q^*(\boldsymbol{f})$ is related to those words. This remark will be used in the proofs of Propositions 16, 18 and 20.

Remark 7:

- For q even, the first word in A_q^n, with respect to $\prec$ order, which does not begin by 0 is $1(q-1)0^{n-2}$.
- For q odd, the first word in A_q^n, with respect to $\lhd$ order, which does not begin by 0 is 10^{n-1}.
- For $q \geq 2$ (even or odd), the last word in A_q^n, with respect to the appropriate order, which does not begin by a $q-1$ is $(q-2)(q-1)0^{n-2}$.

In what follows, we need the technical lemma below.

Lemma 2:

Let $\boldsymbol{u}, \boldsymbol{g} \in A_q^*$ and $\boldsymbol{v} \in A_q^+$ be such that $\boldsymbol{g}$ is a suffix of both $\boldsymbol{u}$ and $\boldsymbol{uv}$. Then there exist a $j \geq 0$ and a (possibly empty) suffix $\boldsymbol{w}$ of $\boldsymbol{v}$ such that $\boldsymbol{g} = \boldsymbol{wv}^j$ (or equivalently, $\boldsymbol{g}$ is a suffix of the left infinite word $\boldsymbol{v}^{-\infty}$).

Proof. We prove the statement by induction on $k = \lfloor \frac{|\boldsymbol{g}|}{|\boldsymbol{v}|} \rfloor$. When $k = 0$, since the length of $\boldsymbol{g}$ is less than that of $\boldsymbol{v}$, and $\boldsymbol{g}$ is a suffix of $\boldsymbol{uv}$, the statement follows by considering $j = 0$ and $\boldsymbol{w} = \boldsymbol{g}$.

Now, let $k = \lfloor \frac{|g|}{|v|} \rfloor > 0$. In this case the length of g is greater than that of v, and it follows that v is a suffix of both g and u. By considering u' and g' such that

- $u = u'v$
- $g = g'v$

we have that g' is a suffix of both u' and $u = u'v$. Since $|g'| = |g| - |v|$, we have that $\lfloor \frac{|g'|}{|v|} \rfloor = k - 1$ and the statement follows by induction on k. □

6.2.1/ FORBIDDEN FACTOR ENDING WITH 0 AND NOT INDUCING ZERO PERIODICITY

We will determine, according to the parity of q, the form of the first and the last words in $p|A_q^\infty(f)$ having no ultimate period 0 for f ending with 0, and consequently the form of the forbidden factors f that do not induce zero periodicity.

q EVEN

Proposition 16:

Let $q \geq 2$ be even and $f \in A_q^+$ be a forbidden factor ending with 0 and not inducing zero periodicity. Also, let $p \in A_q^*(f)$ be such that one of the $\prec$-first or the $\prec$-last word in $p|A_q^\infty(f)$ does not have ultimate period 0, and let a be this word. Then a is ultimately periodic; more precisely, there is an $m \geq 0$ such that either

(i) $a = p0^i(1(q-1)0^m)^\infty$, for some $i \leq m$, or

(ii) $a = p((q-1)0^m1)^\infty$.

Proof. We will show that when p has even (resp., odd) parity, then either

1. a is the $\prec$-first (resp., $\prec$-last) word in $p|A_q^\infty(f)$, and in this case a has the form given in point (i) above, or
2. a is the $\prec$-last (resp., $\prec$-first) word in $p|A_q^\infty(f)$, and in this case a has the form given in point (ii) above.

For the point 1, considering the parity of p, since a does not have ultimate period 0, there is an $i \geq 0$ such that $p0^{i+1}$ contains the factor f, but $p0^i$ does not. Now, by Remark 7, since f ends by a 0, it follows that there is an $m \geq 0$ such that $p0^i\,1(q-1)0^m$ is a prefix of a, but $p0^i1(q-1)0^{m+1}$ is not. Thus, the longest 0 suffix of f is 0^{m+1}, and reasoning in the

same way, it follows that there is an $m' \geq 0$ such that $\boldsymbol{p}0^i\,1(q-1)0^m\,1(q-1)0^{m'}$ is a prefix of $\boldsymbol{a}$, but $\boldsymbol{p}0^i\,1(q-1)0^m\,1(q-1)0^{m'+1}$ is not. Since 0^{m+1} is the longest 0 suffix of $\boldsymbol{f}$, necessarily $m' = m$, and the statement holds by iterating this construction.

Similarly, point 2 holds considering that $\boldsymbol{p}(q-1)$ is a prefix of $\boldsymbol{a}$ and there is an $m \geq 0$ such that $\boldsymbol{p}(q-1)0^{m+1}$ contains the factor $\boldsymbol{f}$.

□

Now we characterize the forbidden factors $\boldsymbol{f} \in A_q^+$ ending with 0, for even $q \geq 2$, and not inducing zero periodicity.

For even $q \geq 2$, we define the set $U_q \subset A_q^+$ as

$$U_q = \bigcup_{m\geq 0}\{\boldsymbol{b}0 \mid \boldsymbol{b} \text{ a suffix of } (1(q-1)0^m)^{-\infty}\}. \tag{6.1}$$

Alternatively, U_q is the set of words of the form $\boldsymbol{b}0$, where $\boldsymbol{b}$ is either empty, or for some $m \geq 0$, a factor of $(1(q-1)0^m)^\infty$ ending with 0^m if $m > 0$ and ending with $q-1$ elsewhere. Clearly, U_q contains exactly n words of length n, for example, $U_4 \cap A_4^5 = \{00000, 30000, 13000, 01300, 13130\}$.

Proposition 17:

For even $q \geq 2$, if a forbidden factor $\boldsymbol{f} \in A_q^+$ ending with 0 does not induce zero periodicity, then $\boldsymbol{f} \in U_q$.

Proof. If $\boldsymbol{f}$ is such a factor, then by Proposition 16 there is a $\boldsymbol{p} \in A^*(\boldsymbol{f})$ and an $m \geq 0$ such that, $\boldsymbol{a}$, the $\prec$-first or the $\prec$-last word in $\boldsymbol{p}|A_q^\infty(\boldsymbol{f})$ is

- $\boldsymbol{a} = \boldsymbol{p}0^i(1(q-1)0^m)^\infty$, for some $i \leq m$, or
- $\boldsymbol{a} = \boldsymbol{p}((q-1)0^m1)^\infty$.

Let $\boldsymbol{g}$ be the word obtained from $\boldsymbol{f}$ after erasing its last 0. In the first case it follows that $\boldsymbol{g}$ is a suffix of both $\boldsymbol{p}0^i\,1(q-1)0^m$ and $\boldsymbol{p}0^i\,1(q-1)0^m\,1(q-1)0^m$, and by Lemma 2 the statement holds. For the second case, the proof is similar. □

Remark 8:

If $\boldsymbol{f} \in U_q$ and q is even, then $\boldsymbol{f}$ does not induce zero periodicity. For example, let $\boldsymbol{f} = \boldsymbol{b}0$ with $\boldsymbol{b}$ a suffix of $(1(q-1)0^m)^{-\infty}$ be as in relation (6.1). Then either the first word in $\boldsymbol{b}|A_q^\infty(\boldsymbol{f})$ when $\boldsymbol{b}$ has even parity, or the last word in $\boldsymbol{b}|A_q^\infty(\boldsymbol{f})$ when $\boldsymbol{b}$ has odd parity, has ultimate period $1(q-1)0^m$.

Example 6:

Let $\boldsymbol{f} = 301300 \in U_4$ be a forbidden factor and consider the prefix $\boldsymbol{p} = 0021301 \in A_4^*$. The $\prec$-first word in $\boldsymbol{p}|A_4^\infty(\boldsymbol{f})$ is $\boldsymbol{p}30(130)^\infty$.

Combining Proposition 17 and Remark 8 we have the following theorem.

Theorem 7:

For even $q \geq 2$, the forbidden factor $\boldsymbol{f} \in A_q^+$ ending with 0 does not induce zero periodicity if and only if $\boldsymbol{f} \in U_q$.

q ODD

Now, we give the odd q counterpart of the previous results.

Proposition 18:

Let $q \geq 3$ be odd and $\boldsymbol{f} \in A_q^+$ be a forbidden factor ending with 0 and not inducing zero periodicity. Also, let $\boldsymbol{p} \in A_q^*(\boldsymbol{f})$ be such that one of the $\lhd$-first or the $\lhd$-last word in $\boldsymbol{p}|A_q^\infty(\boldsymbol{f})$ does not have ultimate period 0, and let $\boldsymbol{a}$ be this word. Then $\boldsymbol{a}$ is ultimately periodic; more precisely, there is an $m \geq 0$ such that either

- $\boldsymbol{a} = \boldsymbol{p}0^i(10^m)^\infty$, for some $i \leq m$, or
- $\boldsymbol{a} = \boldsymbol{p}(q-1)(0^m1)^\infty$.

Proof. The proof is similar to that of Proposition 16 except that the second point of Remark 7 is used instead of the first point. □

Now we characterize the forbidden factor $\boldsymbol{f} \in A_q^+$ ending with 0, for odd $q \geq 3$, and not inducing zero periodicity.

For $q \geq 3$, we define the set $V \subset A_q^+$ as

$$V = \bigcup_{m \geq 0} \{\boldsymbol{b}0 \mid \boldsymbol{b} \text{ a suffix of } (10^m)^{-\infty}\}. \tag{6.2}$$

Alternatively, V is the set of words of the form $\boldsymbol{b}0$, where $\boldsymbol{b}$ is either empty, or for some $m \geq 0$, a factor of $(10^m)^\infty$ ending with 0^m if $m > 0$ (and ending with 1 elsewhere). Notice that V does not depend on q, i.e. $V \subset A_q^+$ for any $q \geq 2$. Clearly, V contains exactly n words of length n, for example, $V \cap A_q^5 = \{00000, 10000, 01000, 10100, 11110\}$, for any $q \geq 2$.

Considering Proposition 18 and the definition of $\lhd$ order relation, with the same arguments as in the proof of Proposition 17 we have the next result.

Proposition 19:

For odd $q \geq 3$, if a forbidden factor $\boldsymbol{f} \in A_q^+$ ending with 0 does not induce zero periodicity, then $\boldsymbol{f} \in V$.

Remark 9:

If $\boldsymbol{f} \in V$ and $q \geq 3$ is odd, then $\boldsymbol{f}$ does not induce zero periodicity on $A_q^\infty(\boldsymbol{f})$. Indeed, let for example $\boldsymbol{f} = \boldsymbol{b}0$ with $\boldsymbol{b}$ a suffix of $(10^m)^{-\infty}$ be as in relation (6.2). Then either the first word in $\boldsymbol{b}|A_q^\infty(\boldsymbol{f})$ when $\boldsymbol{b}$ has even parity, or the last word in $\boldsymbol{b}|A_q^\infty(\boldsymbol{f})$ when $\boldsymbol{b}$ has odd parity, has ultimate period 10^m.

Example 7:

Let $\boldsymbol{f} = 0100010000 \in V$ be the forbidden factor and consider the prefix $\boldsymbol{p} = 430100010 \in A_5^*$. The $\lhd$-last word in $\boldsymbol{p}|A_5^\infty(\boldsymbol{f})$ is $\boldsymbol{p}00(1000)^\infty$.

Combining Proposition 19 and Remark 9, we have the following theorem.

Theorem 8:

For odd $q \geq 3$, the forbidden factor $\boldsymbol{f} \in A_q^+$ ending with 0 does not induce zero periodicity if and only if $\boldsymbol{f} \in V$.

6.2.2/ Forbidden factor ending with $q-1$ and not inducing zero periodicity

The next proposition holds for $q \geq 3$ (even or odd), and the case for $q = 2$ is stated in the remark that follows it.

Proposition 20:

Let $q \geq 3$ (even or odd) and $\boldsymbol{f} \in A_q^+$ be a forbidden factor ending with $q-1$ and not inducing zero periodicity. Also, let $\boldsymbol{p} \in A_q^*(\boldsymbol{f})$ be such that one of the first or the last word in $\boldsymbol{p}|A_q^\infty(\boldsymbol{f})$, with respect to the appropriate order, does not have ultimate period 0, and let $\boldsymbol{a}$ be this word. Then $\boldsymbol{a} = \boldsymbol{p}(q-2)^\infty$.

Proof. Neither $\boldsymbol{p}0$ nor $\boldsymbol{p}(q-1)$ can be a prefix of $\boldsymbol{a}$; otherwise, in the first case $\boldsymbol{a} = \boldsymbol{p}0^\infty$ and in the second one $\boldsymbol{a} = \boldsymbol{p}(q-1)0^\infty$. By the third point of Remark 7 and since $\boldsymbol{f}$ ends by a $q-1$, it follows that $\boldsymbol{p}(q-2)$ is a prefix of $\boldsymbol{a}$, but $\boldsymbol{p}(q-2)(q-1)$ is not (otherwise $\boldsymbol{a} = \boldsymbol{p}(q-2)(q-1)0^\infty$). Again, $\boldsymbol{p}(q-2)(q-2)$ is a prefix of $\boldsymbol{a}$, but $\boldsymbol{p}(q-2)(q-2)(q-1)$ is not; and finally $\boldsymbol{a} = \boldsymbol{p}(q-2)^\infty$. □

When $q = 2$, the ultimate $(q-2)$ period of $\boldsymbol{a}$ in Proposition 20 becomes 0 period, and so, for $q = 2$ any forbidden factor $\boldsymbol{f} \in A_q^+$ ending with $q-1 = 1$ induces zero periodicity. Thus, below we will consider only factors ending with $q-1$ and not inducing zero periodicity only for $q \geq 3$ (even or odd).

For $q \geq 3$, we define the set W_q as

$$W_q = \bigcup_{\ell \geq 0} \{(q-2)^\ell (q-1)\}. \tag{6.3}$$

With the previous terminology, W_q is the set of words of the form $\boldsymbol{b}(q-1)$ with $\boldsymbol{b}$ a suffix of $(q-2)^{-\infty}$. Clearly, W_q contains exactly one word of each length, and for example, $W_4 = \{3, 23, 223, 2223, 22223, \ldots\}$.

Proposition 21:

For $q \geq 3$ (even or odd), if the forbidden factor $\boldsymbol{f} \in A_q^+$ ending with $q-1$ does not induce zero periodicity, then $\boldsymbol{f} \in W_q$.

Proof. Let $\boldsymbol{f}$ be such a factor, and $\boldsymbol{p} \in A_q^*(\boldsymbol{f})$ such that, with respect to the appropriate order, the first word in $\boldsymbol{p}|A_q^\infty(\boldsymbol{f})$ does not have ultimate period 0 (the case of the first word being similar). Also, let $\boldsymbol{g}$ be the (possibly empty) word obtained from $\boldsymbol{f}$ after erasing its last symbol $q-1$. By Proposition 20, $\boldsymbol{g}$ is a suffix of both $\boldsymbol{p}(q-2)$ and $\boldsymbol{p}(q-2)(q-2)$, and by Lemma 2 the statement holds. □

Remark 10:

If $\boldsymbol{f} \in W_q$, then $\boldsymbol{f}$ does not induce zero periodicity. Indeed, let for example $\boldsymbol{b} = (q-2)^\ell$, for some $\ell \geq 0$, and $\boldsymbol{f} = \boldsymbol{b}(q-1)$ be as in relation (6.3). Then the last word in $\boldsymbol{b}|A_q^\infty(\boldsymbol{f})$ has ultimate period $(q-2)$.

Example 8:

Let $\boldsymbol{f} = 223 \in W_4$ be a forbidden factor and consider the prefix $\boldsymbol{p} = 2322 \in A_4^*$. The $\prec$-first word in $\boldsymbol{p}|A_4^\infty(\boldsymbol{f})$ is $\boldsymbol{p}2^\infty$. And when $\boldsymbol{f} = 12 \in W_3$ and $\boldsymbol{p} = 01 \in A_3^*$, the $\lhd$-last word in $\boldsymbol{p}|A_3^\infty(\boldsymbol{f}) = \boldsymbol{p}1^\infty$.

Combining Proposition 21 and Remark 10, we have the following theorem.

Theorem 9:

For $q \geq 3$ (even or odd), the forbidden factor $\boldsymbol{f} \in A_q^+$ ending with $q-1$ does not induce zero periodicity if and only if $\boldsymbol{f} \in W_q$.

Even though we will not use the following remark later, it is worthwhile to mention it.

Remark 11:

For q even (resp., odd), if $\boldsymbol{f}$, $|\boldsymbol{f}| \geq 2$, does not have the form 0^ℓ nor $(q-1)0^\ell$ (resp., the form 0^ℓ) for some $\ell \geq 1$, then for any $\boldsymbol{p} \in A_q^*(\boldsymbol{f})$, at least one among the $\prec$-first and the $\prec$-last word in $\boldsymbol{p}|A_q^\infty(\boldsymbol{f})$ (resp., the $\lhd$-first and the $\lhd$-last word in $\boldsymbol{p}|A_q^\infty$) has ultimate period 0.

6.2.3/ FORBIDDEN FACTOR INDUCING ZERO PERIODICITY

Here we characterize the first and the last words in $\boldsymbol{p}|A_q^\infty(\boldsymbol{f})$ when the forbidden factor $\boldsymbol{f}$ induces zero periodicity; the resulting ultimate 0 periodic words will be used in the next section.

Proposition 22:

Let $q \geq 2$ be even, $\boldsymbol{f} \in A_q^+ \setminus U_q$ be a forbidden factor ending with 0, $\ell \geq 1$ be the length of the maximal 0 suffix of $\boldsymbol{f}$, and $\boldsymbol{p} \in A_q^*(\boldsymbol{f})$. If $\boldsymbol{a}$ is the $\prec$-first or the $\prec$-last word in $\boldsymbol{p}|A_q^\infty(\boldsymbol{f})$, then $\boldsymbol{a}$ has the form

$$\boldsymbol{p}\boldsymbol{r}0^\infty,$$

where

1. $\boldsymbol{r} = \epsilon$ or $\boldsymbol{r} = 0^i 1(q-1)$ for some i, $0 \leq i \leq \ell - 1$, if $\boldsymbol{a}$ is the $\prec$-first (resp., $\prec$-last) word in $\boldsymbol{p}|A_q^\infty(\boldsymbol{f})$ and $\boldsymbol{p}$ has even (resp., odd) parity, or
2. $\boldsymbol{r} = q-1$ or $(q-1)0^{\ell-1}1(q-1)$ if $\boldsymbol{a}$ is the $\prec$-first (resp., $\prec$-last) word in $\boldsymbol{p}|A_q^\infty(\boldsymbol{f})$ and $\boldsymbol{p}$ has odd (resp., even) parity.

Proof. We prove the first point, the second one being similar. Let $\boldsymbol{a}$ be the $\prec$-first (resp., $\prec$-last) word in $\boldsymbol{p}|A_q^\infty(\boldsymbol{f})$ with $\boldsymbol{p}$ having even (resp., odd) parity. Also suppose that $\boldsymbol{r}$ does not have the form described in point 1. Reasoning as in the proof of Proposition 16 it follows that $0^i 1(q-1)0^{\ell-1}1(q-1)0^{\ell-1}$ is a prefix of $\boldsymbol{r}$, for some i, $0 \leq i \leq \ell - 1$, and finally, by Lemma 2 that $\boldsymbol{f} \in U_q$, which leads to a contradiction. □

The proof of the next proposition is similar to that of Proposition 22.

Proposition 23:

Let $q \geq 3$ be odd, $\boldsymbol{f} \in A_q^+ \setminus V$ be a forbidden factor ending with 0, $\ell \geq 1$ be the length of the maximal 0 suffix of $\boldsymbol{f}$, and $\boldsymbol{p} \in A_q^*(\boldsymbol{f})$. If $\boldsymbol{a}$ is the $\lhd$-first or the $\lhd$-last word in $\boldsymbol{p}|A_q^\infty(\boldsymbol{f})$, then $\boldsymbol{a}$ has the form

$$\boldsymbol{p}\boldsymbol{r}0^\infty,$$

where

1. $\boldsymbol{r} = \epsilon$ or $\boldsymbol{r} = 0^i 1$ for some i, $0 \leq i \leq \ell - 1$, if $\boldsymbol{a}$ is the $\lhd$-first (resp., $\lhd$-last) word in $\boldsymbol{p}|A_q^\infty(\boldsymbol{f})$ and $\boldsymbol{p}$ has even (resp., odd) parity, or
2. $\boldsymbol{r} = q - 1$ or $(q-1)0^{\ell-1}1$ if $\boldsymbol{a}$ is the $\lhd$-first (resp., $\lhd$-last) word in $\boldsymbol{p}|A_q^\infty(\boldsymbol{f})$ and $\boldsymbol{p}$ has odd (resp., even) parity.

It is routine to check the next two propositions.

Proposition 24:

Let $q \geq 3$, $\boldsymbol{f} \in A_q^+ \setminus W_q$ be a forbidden factor ending with $q-1$, and $\boldsymbol{p} \in A_q^*(\boldsymbol{f})$. If $\boldsymbol{a}$ is the first or the last word in $\boldsymbol{p}|A_q^\infty(\boldsymbol{f})$ with respect to the appropriate order, then $\boldsymbol{a}$ has the form

$$\boldsymbol{p}\boldsymbol{r}0^\infty,$$

where $\boldsymbol{r}$ is either ϵ, or $q-1$, or $(q-2)(q-1)$.

As mentioned in Remark 5, forbidden factors ending with symbol other than 0 or $q-1$ induce zero periodicity, and we have the following proposition.

Proposition 25:

If $\boldsymbol{f} \in A_q^*$ is a forbidden factor that does not end by 0 nor by $q-1$, then for any $\boldsymbol{p} \in A_q^*(\boldsymbol{f})$, with respect to the appropriate order, both the first and the last word in $\boldsymbol{p}|A_q^\infty(\boldsymbol{f})$ have the form:

$$\boldsymbol{p}\boldsymbol{r}0^\infty,$$

where $\boldsymbol{r}$ is either ϵ or $q-1$.

We will see later that Propositions 22 to 25 above describe sufficient (but not necessary) conditions for the Graycodeness of $A_q^n(\boldsymbol{f})$.

We conclude this section by the next corollary which summarizes the results in Remark 5 and Theorems 7, 8 and 9, and we will refer to it later.

Corollary 1:

The forbidden factor $\boldsymbol{f} \in A_q^*$ induces zero periodicity if and only if either:

- $\boldsymbol{f}$ does not end by 0 nor by $q-1$, or
- $q = 2$ and $\boldsymbol{f} \notin U_2$, or
- $q \geq 4$ is even and $\boldsymbol{f} \notin U_q \cup W_q$, or
- $q \geq 3$ is odd and $\boldsymbol{f} \notin V \cup W_q$.

6.3/ THE GRAY CODES

In this section we show that for forbidden factors $\boldsymbol{f}$ inducing zero periodicity on A_q^∞ (as stated in Corollary 1) consecutive words—in $\prec$ order for q even, or $\lhd$ order for q odd—in $A_q^n(\boldsymbol{f})$, beyond the common prefix, have all symbols 0, except the first few of them; and this ensures that the set $A_q^n(\boldsymbol{f})$ listed in an appropriate order is a Gray code.

Nevertheless, the property of $\boldsymbol{f}$ to induce zero periodicity is not a necessary condition. Indeed, listing the set $A_q^n(\boldsymbol{f})$ in $\prec$ order with:

- $\boldsymbol{f} = 0^\ell$ for any q (not necessarily even), or
- $\boldsymbol{f} = (q-1)0^\ell$ for q even,

where $\ell \geq 1$, yields a 1-Gray code, despite $\boldsymbol{f} \in U_q$ (and so, $\boldsymbol{f}$ does not induce zero periodicity for q even). These particular cases are discussed in Section 6.3.2, and we show that such factors $\boldsymbol{f}$, $\boldsymbol{f} \geq 2$, are the only ones giving Gray codes for forbidden factors not inducing zero periodicity. In particular, the Gray code obtained for $A_q^n(0^\ell)$ is one of those defined in [BBPV13] as a generalization of a Gray code in [Vaj01]. Finally, for forbidden factors $\boldsymbol{f}$ for which $\prec$ nor $\lhd$ does not produce Gray codes on $A_q^n(\boldsymbol{f})$, we give simple transformations of $\boldsymbol{f}$, and eventually obtain Gray codes for $A_q^n(\boldsymbol{f})$ (in an order other than $\prec$ or $\lhd$).

We will make use later of the following property of forbidden factors ending with 0 or $q-1$: for *any* $q \geq 2$, if $\boldsymbol{f}$ ends by 0 or $q-1$, then any two consecutive words in $A_q^n(\boldsymbol{f})$, in both $\prec$ and $\lhd$ order, differ by 1 or -1 in the leftmost position where they differ.

Proposition 26:

Let $q \geq 2$ and $\boldsymbol{f} \in A_q^+$ be a forbidden factor ending with 0 or $q-1$, and $\boldsymbol{a} = a_1 a_2 \dots a_n$ and $\boldsymbol{b} = b_1 b_2 \dots b_n$ be two words in $A_q^n(\boldsymbol{f})$, consecutive with respect to $\prec$ or $\lhd$ order. If k is the leftmost position where $\boldsymbol{a}$ and $\boldsymbol{b}$ differ, then $b_k = a_k + 1$ or $b_k = a_k - 1$.

Proof. If $\boldsymbol{f}$ ends by 0, we assume that $b_k < a_k - 1$. It follows that $\boldsymbol{f}$ is a suffix of $a_1a_2\dots(a_k - 1)$, so $a_k - 1 = 0$ and thus $b_k < 0$, which is a contradiction. The proof when $b_k > a_k + 1$ or when $\boldsymbol{f}$ ends by $q-1$ is similar. □

6.3.1/ FACTORS INDUCING ZERO PERIODICITY

We show that for factors $\boldsymbol{f}$ as in Corollary 1 the set $A_q^n(\boldsymbol{f})$ listed in $\prec$ or $\lhd$ order is a Gray code.

Proposition 27:

If q is even (resp., odd) and $\boldsymbol{f} \in A_q^+$ does not end by 0 nor $q-1$, then $A_q^n(\boldsymbol{f})$, $n \geq 1$, listed in $\prec$ (resp., $\lhd$) order is a 2-adjacent Gray code.

Proof. Let $\boldsymbol{a}, \boldsymbol{b} \in A_q^n(\boldsymbol{f})$, $\boldsymbol{a} = a_1a_2\dots a_n$ and $\boldsymbol{b} = b_1b_2\dots b_n$ be two consecutive words with respect to the appropriate order, and k be the leftmost position where they differ. Since $\boldsymbol{f}$ does not end by 0 nor by $q-1$, it follows that $q \geq 3$, and considering the definitions of $\prec$ and $\lhd$ order, we have in both cases (see Remark 4) $\{a_{k+1}, b_{k+1}\} \subset \{0, q-1\}$ and $a_{k+2}\dots a_n = b_{k+2}\dots b_n = 0^{n-k-1}$. In any case, $\boldsymbol{a}$ and $\boldsymbol{b}$ differ in position k and possibly in position $k+1$. □

Theorem 10:

If $q \geq 2$ is even, $\boldsymbol{f} \in A_q^+ \setminus U_q$ ends by 0, and ℓ is the length of the maximal 0 suffix of $\boldsymbol{f}$, then $A_q^n(\boldsymbol{f})$, $n \geq 1$, listed in $\prec$ order is an at most $(\ell+2)$-close 3-Gray code.

Proof. Let $\boldsymbol{a}, \boldsymbol{b} \in A_q^n(\boldsymbol{f})$, $\boldsymbol{a} = a_1a_2\dots a_n$ and $\boldsymbol{b} = b_1b_2\dots b_n$ be two consecutive words with respect to the appropriate order, and k be the leftmost position where $\boldsymbol{a}$ and $\boldsymbol{b}$ differ. By Proposition 26, $b_k = a_k + 1$ or $b_k = a_k - 1$ and so the prefixes $\boldsymbol{a}' = a_1a_2\dots a_k$ and $\boldsymbol{b}' = b_1b_2\dots b_k$ have different parity. Two cases arise according to the parity of $\boldsymbol{a}'$.

• $\boldsymbol{a}'$ has even parity, and so $\boldsymbol{b}'$ has odd parity. By point 2 of Proposition 22

$$\boldsymbol{a} = \boldsymbol{a}'\boldsymbol{x},$$

and

$$\boldsymbol{b} = \boldsymbol{b}'\boldsymbol{y},$$

with $\boldsymbol{x}$ and $\boldsymbol{y}$ being the $n-k$ prefixes of $\boldsymbol{r}0^\infty$ and of $\boldsymbol{r}'0^\infty$, where $\{\boldsymbol{r}, \boldsymbol{r}'\} \subset \{(q-1), (q-1)0^{\ell-1}1(q-1)\}$. Thus $\boldsymbol{a}$ and $\boldsymbol{b}$ differ in position k and possibly in positions $k+\ell+1$ and $k+\ell+2$ if $k+\ell+1 \geq n$.

• $\boldsymbol{a}'$ has odd parity, and so $\boldsymbol{b}'$ has even parity. By point 1 of Proposition 22 either

(i) $a_{k+1}a_{k+2}\dots a_n = b_{k+1}b_{k+2}\dots b_n = 0^{n-k}$, or

(ii) at least one of $a_{k+1}a_{k+2}\dots a_n$ or $b_{k+1}b_{k+2}\dots b_n$ is the $n-k$ prefix of a word of the form $0^i1(q-1)0^\infty$.

In case (i), $\boldsymbol{a}$ and $\boldsymbol{b}$ differ only in position k. And in case (ii) we suppose that $a_{k+1}a_{k+2}\dots a_n$ is the length $n-k$ prefix of $0^i1(q-1)0^\infty$ (the corresponding case for $b_{k+1}b_{k+2}\dots b_n$ being similar). Considering that $b_k = a_k+1$ or $b_k = a_k-1$ it follows that $b_{k+1}b_{k+2}\dots b_n$ is the length $n-k$ prefix of 0^∞ and so $\boldsymbol{a}$ and $\boldsymbol{b}$ differ in positions k, and (possibly) $k+i+1$ and $k+i+2$.

In any case, $\boldsymbol{a}$ and $\boldsymbol{b}$ differ in at most three positions which are at most $\ell+2$ apart from each other. □

Example 9:

The words 00230130 and 00330000 are consecutive in $A_4^8(2300)$ listed in $\prec$ order. They differ in 3 positions which are 4-close, and are in the worst case since the list is a 4-close 3-Gray code.

Considering the possible values of $\boldsymbol{r}$ in Proposition 23 it is easy to see that for $\boldsymbol{f} \notin V$ ending with 0 and q odd, the set $A_q^n(\boldsymbol{f})$ listed in $\lhd$ order is a 4-Gray code. The next theorem gives a more restrictive result.

Theorem 11:

If $q \geq 3$ is odd, $\boldsymbol{f} \in A_q^+ \setminus V$ ends by 0, and ℓ is the length of the maximal 0 suffix of $\boldsymbol{f}$, then $A_q^n(\boldsymbol{f})$, $n \geq 1$, listed in $\lhd$ order is an at most $(\ell+1)$-close 3-Gray code.

Proof. Let $\boldsymbol{a}, \boldsymbol{b} \in A_q^n(\boldsymbol{f})$, $\boldsymbol{a} = a_1a_2\dots a_n$ and $\boldsymbol{b} = b_1b_2\dots b_n$ be two consecutive words, in $\lhd$ order, and k be the leftmost position where $\boldsymbol{a}$ and $\boldsymbol{b}$ differ. If $\boldsymbol{a}'$ and $\boldsymbol{b}'$ are the length k prefix of $\boldsymbol{a}$ and $\boldsymbol{b}$, by Proposition 23

$$\boldsymbol{a} = \boldsymbol{a}'\boldsymbol{x},$$

and

$$\boldsymbol{b} = \boldsymbol{b}'\boldsymbol{y},$$

with $\boldsymbol{x}$ and $\boldsymbol{y}$ being the $n-k$ prefixes of $\boldsymbol{r}0^\infty$ and of $\boldsymbol{r}'0^\infty$, where $\{\boldsymbol{r}, \boldsymbol{r}'\} \subset \{\epsilon, 0^i1, (q-1), (q-1)0^{\ell-1}1\}$, for some i, $0 \leq i \leq \ell-1$. The statement holds by showing that $\{\boldsymbol{r}, \boldsymbol{r}'\} \subset \{0^i1, (q-1)0^{\ell-1}1\}$ is not possible. Indeed, let us suppose that $\boldsymbol{r} = 0^i1$ for some i, $0 \leq i \leq \ell-1$, and $\boldsymbol{r}' = (q-1)0^{\ell-1}1$ (the case $\boldsymbol{r} = (q-1)0^{\ell-1}1$ and $\boldsymbol{r}' = 0^i1$ being similar). This happens when both $\boldsymbol{a}'$ and $\boldsymbol{b}'$ have both odd parity. By Proposition 26, $b_k = a_k+1$ or $b_k = a_k-1$, and since $a_1a_2\dots a_{k-1} = b_1b_2\dots b_{k-1}$ it follows that $a_k = 1$ and $b_k = 0$. Since $\boldsymbol{r}' = (q-1)0^{\ell-1}1$, the factor $\boldsymbol{f}$ must end by $(q-1)0^\ell$ and since $a_k = 1$ it follows that $\boldsymbol{r} = \epsilon$, which leads to a contradiction. □

Example 10:

By Theorem 11, the sets $A_5^9(31000)$ and $A_5^9(24000)$ listed in $\lhd$ order are 4-close 3-Gray codes. However, it is easy to check that in particular, $A_5^9(31000)$ is a 3-close 3-Gray code, and $A_5^9(24000)$ is 4-close 2-Gray code. For example:

- the words 001304000 and 001310010 are consecutive in $A_5^9(31000)$ when listed in $\lhd$ order; they differ in 3 positions which are 3-close; and
- the words 001140000 and 001240010 are consecutive in $A_5^9(24000)$ when listed in $\lhd$ order; they differ in 2 positions which are 4-close.

Theorem 12:

If q is even (resp., odd) and $\boldsymbol{f} \in A_q^+ \setminus W_q$ ends by $q-1$, then $A_q^n(\boldsymbol{f})$ listed in $\prec$ order (resp., $\lhd$ order) is a 2-close 3-Gray code (that is, a 3-adjacent Gray code).

Proof. Let k be the leftmost position where two consecutive words $\boldsymbol{a} = a_1a_2\dots a_n$ and $\boldsymbol{b} = b_1b_2\dots b_n$, in $A_q^n(\boldsymbol{f})$ differ. By considering all the possible values of a_k and b_k, and since $\boldsymbol{f} \notin W_q$ ends by $q-1$, we conclude that $a_i = b_i = 0$ for all $i > k+2$ (see also Proposition 24). □

6.3.2/ PARTICULAR CASES

As mentioned before, there are two cases when $\boldsymbol{f} \in U_q$, $q \geq 2$ and even, but $A_q^n(\boldsymbol{f})$ listed in $\prec$ order is a Gray code; these are $\boldsymbol{f} = 0^\ell$ and $\boldsymbol{f} = (q-1)0^\ell$, $\ell \geq 1$. Moreover, it turns out that $A_q^n(0^\ell)$, $q \geq 3$ and odd, also gives Gray code if listed in $\prec$ order. A similar phenomenon does not occur for $\boldsymbol{f} \in V$, i.e., the set $A_q^n(\boldsymbol{f})$ listed in $\lhd$ order is not a Gray code for any $\boldsymbol{f} \in V$, $|\boldsymbol{f}| \geq 2$ and $q \geq 3$ odd, see for example Remark 12.

Before discussing these particular forbidden factors we introduce some notations.

Let $q \geq 2$, $\ell \geq 1$, and define the infinite words:

$$\begin{aligned} \boldsymbol{u} &= (0^{\ell-1}1(q-1))^\infty, \\ \boldsymbol{v} &= ((q-1)0^{\ell-1}1)^\infty. \end{aligned} \tag{6.4}$$

Note that $\boldsymbol{u}$ and $\boldsymbol{v}$ are suffixes of each other, and they are related with the infinite words occurring in Proposition 16. It is easy to see that $\boldsymbol{u}$ and $\boldsymbol{v}$ are, respectively, the $\prec$-first and $\prec$-last word in $A_q^\infty(0^\ell)$ for even q; and thus the length n prefix of $\boldsymbol{u}$ and $\boldsymbol{v}$ are, respectively, the $\prec$-first and $\prec$-last word in $A_q^n(0^\ell)$.

Moreover, for any $\boldsymbol{p} \in A_q^k(0^\ell)$ with $1 \leq k \leq n$ and q even

- the $\prec$-first (resp., $\prec$-last) word in $\boldsymbol{p}|A_q^n(0^\ell)$ is $\boldsymbol{pv'}$ if $\boldsymbol{p}$ has an odd (resp., even) parity, where $\boldsymbol{v'}$ is the length $n-k$ prefix of $\boldsymbol{v}$;
- if $\boldsymbol{p}$ does not end by 0, then the $\prec$-first (resp., $\prec$-last) word in $\boldsymbol{p}|A_q^n(0^\ell)$ is $\boldsymbol{pu'}$ if $\boldsymbol{p}$ has an even (resp., odd) parity, where $\boldsymbol{u'}$ is the length $n-k$ prefix of $\boldsymbol{u}$.

Now let $q \geq 3$ and odd, $\ell \geq 1$, and define the infinite words:

$$\begin{aligned} \boldsymbol{s} &= 0^{\ell-1}1(q-1)^\infty, \\ \boldsymbol{t} &= (q-1)^\infty. \end{aligned} \tag{6.5}$$

It is easy to see that $\boldsymbol{s}$ and $\boldsymbol{t}$ have similar property as $\boldsymbol{u}$ and $\boldsymbol{v}$ for q odd and with same $\prec$ order.

THE CASE $\boldsymbol{f} = 0^\ell$

Proposition 28:

For $q \geq 2$ (even or odd), and $\ell, n \geq 1$, the set $A_q^n(0^\ell)$ listed in $\prec$ order is a Gray code where two consecutive words differ in one position and by 1 or -1 in this position.

Proof. Let $\boldsymbol{a}$ and $\boldsymbol{b}$ be two consecutive words, in $\prec$ order, in $A_q^n(0^\ell)$, $\boldsymbol{a'} = a_1a_2\ldots a_k$ and $\boldsymbol{b'} = b_1b_2\ldots b_k$ be the length k prefix of $\boldsymbol{a}$ and $\boldsymbol{b}$, with k the leftmost position where $\boldsymbol{a}$ and $\boldsymbol{b}$ differ.

When q is even, with $\boldsymbol{u'}$ and $\boldsymbol{v'}$ the length $(n-k)$ prefix of $\boldsymbol{u}$ and $\boldsymbol{v}$ defined in relation (6.4), we have

- $\boldsymbol{a} = \boldsymbol{a'v'}$ and $\boldsymbol{b} = \boldsymbol{b'v'}$ if $\boldsymbol{a'}$ has an even parity (and so, by Proposition 26, $\boldsymbol{b'}$ has odd parity);
- $\boldsymbol{a} = \boldsymbol{a'u'}$ and $\boldsymbol{b} = \boldsymbol{b'u'}$, elsewhere, since $a_k \neq 0$ and $b_k \neq 0$ by considering the parity of the common length $k-1$ prefix of $\boldsymbol{a}$ and $\boldsymbol{b}$.

Similarly, when q is odd, with $\boldsymbol{s'}$ and $\boldsymbol{t'}$ the length $(n-k)$ prefix of $\boldsymbol{s}$ and $\boldsymbol{t}$ defined in relation (6.5), we have

- $\boldsymbol{a} = \boldsymbol{a't'}$ and $\boldsymbol{b} = \boldsymbol{b't'}$ if $\boldsymbol{a'}$ has an even parity (given by $\sum_{i=1}^k a_i$);
- $\boldsymbol{a} = \boldsymbol{a's'}$ and $\boldsymbol{b} = \boldsymbol{b's'}$ (since, $a_k \neq 0$ and $b_k \neq 0$), elsewhere.

In both cases, $\boldsymbol{a}$ and $\boldsymbol{b}$ differ only in position k. □

THE CASE $f = (q-1)0^\ell$ FOR q EVEN

Let $q \geq 2$ be even, $1 \leq k \leq n$, and $\boldsymbol{v}'$ be the $n-k$ prefix of $\boldsymbol{v}$ defined in relation (6.4). For any $\boldsymbol{p} \in A_q^k((q-1)0^\ell)$ with $\ell \geq 1$ and $1 \leq k \leq n$

- the $\prec$-first (resp., $\prec$-last) word in $\boldsymbol{p}|A_q^n((q-1)0^\ell)$ is $\boldsymbol{p}\boldsymbol{v}'$ if $\boldsymbol{p}$ has odd (resp., even) parity;
- if $\boldsymbol{p}$ does not end by 0 nor by $q-1$, then the $\prec$-first (resp., $\prec$-last) word in $\boldsymbol{p}|A_q^n((q-1)0^\ell)$ is $\boldsymbol{p}0^{n-k}$ if $\boldsymbol{p}$ has even (resp., odd) parity.

Proposition 29:

For $q \geq 2$ even, and $\ell, n \geq 1$, the set $A_q^n((q-1)0^\ell)$ listed in $\prec$ order is a Gray code where two consecutive words differ in one position and by 1 or -1 in this position.

Proof. Let $\boldsymbol{a}$ and $\boldsymbol{b}$ be two consecutive words, in $\prec$ order, in $A_q^n((q-1)0^\ell)$, $\boldsymbol{a}' = a_1 a_2 \dots a_k$ and $\boldsymbol{b}' = b_1 b_2 \dots b_k$ be the length k prefix of $\boldsymbol{a}$ and $\boldsymbol{b}$ with k the leftmost position where $\boldsymbol{a}$ and $\boldsymbol{b}$ differ.

If $\boldsymbol{a}'$ has even parity (and so, by Proposition 26, $\boldsymbol{b}'$ has odd parity), then by the above considerations $\boldsymbol{a} = \boldsymbol{a}'\boldsymbol{v}'$ and $\boldsymbol{b} = \boldsymbol{b}'\boldsymbol{v}'$.

If $\boldsymbol{a}'$ has odd parity, by considering the parity of the common length $k-1$ prefix of $\boldsymbol{a}$ and $\boldsymbol{b}$ it follows that $a_k \neq q-1$ and $b_k \neq q-1$, and again, by the above considerations we have $\boldsymbol{a} = \boldsymbol{a}'0^{n-k}$ and $\boldsymbol{b} = \boldsymbol{b}'0^{n-k}$.

In both cases, $\boldsymbol{a}$ and $\boldsymbol{b}$ differ only in position k. □

6.3.3/ FACTORS PREVENTING GRAYCODENESS

A consequence of the next remark and proposition, is Corollary 2 below. Proposition 30 is similar to Remark 11 and says that if $\boldsymbol{f}$, $|\boldsymbol{f}| \geq 2$ ($|\boldsymbol{f}| = 1$ being trivial), does not induce zero periodicity (see Corollary 1), and it is not in one of the two particular cases above, then consecutive words, with respect to the appropriate order, in $A_q^n(\boldsymbol{f})$ can differ in arbitrarily many positions for large enough n. One of these particular cases is explained below.

Remark 12:

For $q \geq 3$ and odd, $\ell \geq 2$ and $\boldsymbol{f} = 0^\ell$, the set $A_q^n(\boldsymbol{f})$ listed in $\lhd$-order is not a Gray code. Indeed, for example, the words $02z'$ and $1z''$ are consecutive in $\lhd$-order in $A_q^n(\boldsymbol{f})$, where z' and z'' are appropriate length prefixes of $(0^{\ell-1}1)^\infty$, and they differ in arbitrarily many positions for large enough n.

Proposition 30:

Let $\boldsymbol{f} \in A_q^+$, $q \geq 2$ and $|\boldsymbol{f}| \geq 2$, be a forbidden factor not inducing zero periodicity, other than 0^ℓ or $(q-1)0^\ell$, $\ell \geq 1$. Also, let $\boldsymbol{a}$ and $\boldsymbol{b}$ be two consecutive words, in appropriate order, in $A_q^n(\boldsymbol{f})$, $n \geq 1$, and let k be the leftmost position where $\boldsymbol{a}$ and $\boldsymbol{b}$ differ. If

- $\boldsymbol{a}' = a_1 a_2 \dots a_k$ and $\boldsymbol{b}' = b_1 b_2 \dots b_k$ are, respectively, the length k prefix of $\boldsymbol{a}$ and $\boldsymbol{b}$, and
- $\boldsymbol{a}''$ and $\boldsymbol{b}''$ are, respectively, the last word in $\boldsymbol{a}'|A_q^\infty(\boldsymbol{f})$ and the first word in $\boldsymbol{b}'|A_q^\infty(\boldsymbol{f})$, in appropriate order,

then at most one among $\boldsymbol{a}''$ and $\boldsymbol{b}''$ does not have ultimate period 0.

Proof. Since $\boldsymbol{f}$ does not induce zero periodicity, we prove the statement according to whether $\boldsymbol{f}$ belongs to U_q, V or W_q (see Corollary 1), and supposing that $\boldsymbol{a}''$ does not have ultimate period 0 (the corresponding case for $\boldsymbol{b}''$ being similar).

If $\boldsymbol{f} \in U_q$, $q \geq 2$ and $\boldsymbol{f}$ does not have the form 0^ℓ nor $(q-1)0^\ell$:

- When $\boldsymbol{a}'$ has odd parity, since a_k must be a symbol of $\boldsymbol{f}$, it follows that $a_k \in \{0, 1, q-1\}$. From the parity of $\boldsymbol{a}'$, it follows that $a_k = 0$ implies that $b_k = a_k - 1$, and $a_k = q-1$ that $b_k = a_k + 1$, which are not possible, and necessarily $a_k = 1$. Thus, either $\boldsymbol{a}'' = \boldsymbol{a}'0^\infty$ (which is a contradiction with the non-zero periodicity of $\boldsymbol{a}$) or $\boldsymbol{b}'' = \boldsymbol{b}'0^\infty$.
- When $\boldsymbol{a}'$ has even parity, then $\boldsymbol{a}'' = \boldsymbol{a}'\boldsymbol{v}$ and since $b_k \neq a_k$, $\boldsymbol{b}'' = \boldsymbol{b}'(q-1)0^\infty$, with $\boldsymbol{v}$ defined in relation (6.4).

If $\boldsymbol{f} \in V$, $q \geq 3$ and odd, and $\boldsymbol{f}$ does not have the form 0^ℓ:

- $\boldsymbol{a}'$ can not have even parity, otherwise $\boldsymbol{a}'' = \boldsymbol{a}'(q-1)0^\infty$, which is a contradiction with the non-zero periodicity of $\boldsymbol{a}$;
- When $\boldsymbol{a}'$ has odd parity, the symbol a_k must be a symbol in the forbidden factor, so $a_k \in \{0, 1\}$. But $a_k = 0$, implies $b_k = a_k - 1$, which again is not possible; and $a_k = 1$ implies $b_k = 0$, and so $\boldsymbol{b}'' = (q-1)0^\infty$, which does not contain the factor $\boldsymbol{f}$ if it is different from 0^ℓ.

Finally, when $\boldsymbol{f} \in W_q$, $q \geq 3$ (even or odd) then $\boldsymbol{a}'' = \boldsymbol{a}'(q-2)^\infty$ and $\boldsymbol{b}''$ is either $\boldsymbol{b}'0^\infty$ (this can occur if q is odd) or $\boldsymbol{b}'(q-1)0^\infty$. □

Table 6.2 summarizes the cases occurring in Proposition 30.

A consequence of Remark 12, Propositions 28 to 30 and Corollary 1, is the corollary below.

Corollary 2:

- For even $q \geq 2$ and $|\boldsymbol{f}| \geq 2$, the set $A_q^n(\boldsymbol{f})$ listed in $\prec$ order is a Gray code for any $n \geq 1$ if and only if $\boldsymbol{f} \in \{0^\ell, (q-1)0^\ell\}_{\ell \geq 1} \cup W_2 \cup (A_q^* \setminus (U_q \cup W_q))$.
- For odd $q \geq 3$ and $|\boldsymbol{f}| \geq 2$, the set $A_q^n(\boldsymbol{f})$ listed in $\lhd$ order is a Gray code for any $n \geq 1$ if and only if $\boldsymbol{f} \in A_q^* \setminus (V \cup W_q)$.

q	Order relation	The set for forbidden factor $\boldsymbol{f}$	The set of ultimate periods of the last word in $\boldsymbol{a}\|A_q^\infty(\boldsymbol{f})$ and the first one in $\boldsymbol{b}\|A_q^\infty(\boldsymbol{f})$	Graycodeness of $A_q^n(\boldsymbol{f})$
even	$\prec$	$U_q \setminus \{0^\ell, (q-1)0^\ell\}_{\ell \geq 1}$	$\{1(q-1)0^{\ell-1}, 0\}$	Not Gray code
even	$\prec$	$\{0^\ell, (q-1)0^\ell\}_{\ell \geq 1}$	$\{1(q-1)0^{\ell-1}\}$	1-Gray code
odd	$\prec$	$\{0^\ell\}_{\ell \geq 1}$	$\{(q-1)\}$	1-Gray code
odd	$\lhd$	$V \setminus \{0^\ell\}_{\ell \geq 1}$	$\{10^{\ell-1}, 0\}$	Not Gray code
odd	$\lhd$	$\{0^\ell\}_{\ell \geq 2}$	$\{10^{\ell-1}\}$	Not Gray code
$q \geq 3$ even (resp., odd)	$\prec$ (resp., $\lhd$)	$W_q \cap A_q^{\geq 2}$	$\{(q-2), 0\}$	Not Gray code

Table 6.2: The Graycodeness of $A_q^n(\boldsymbol{f})$ listed in appropriate order together with the ultimate periods of the last word in $\boldsymbol{a}|A_q^\infty(\boldsymbol{f})$ and the first word in $\boldsymbol{b}|A_q^\infty(\boldsymbol{f})$, when at least one of them does not have ultimate period 0, and $\boldsymbol{a}$ and $\boldsymbol{b}$ are consecutive words; and $A_q^{\geq 2}$ is the set of words on A_q of length at least two. These summarize Propositions 28 to 30, and Corollary 2.

6.3.4/ Obtaining Gray code if $\boldsymbol{f}$ does not induce zero periodicity and is not one of the particular cases

According to the previous results, if the forbidden factor $\boldsymbol{f}$ does not induce zero periodicity, then the set $A_q^n(\boldsymbol{f})$ listed in $\prec$ or $\lhd$ order is not a Gray code, except for the two particular cases in Section 6.3.2. Now we show how a simple transformation makes it possible to define Gray codes, with the same Hamming distance and closeness properties as for factors that induce zero periodicity, when $\boldsymbol{f}$ does not have this property. By Theorems 7, 8 and 9, the last symbol of a factor that does not induce zero periodicity is either 0, or $q-1$ when $q \geq 3$.

We define the transformation $\phi : A_q \to A_q$ depending on $\boldsymbol{f}$ as:

- When the last symbol of $\boldsymbol{f}$ is 0, then $\phi(0) = 1$, $\phi(1) = 0$, and $\phi(x) = x$ if $x \notin \{0, 1\}$; and

- When the last symbol of $\boldsymbol{f}$ is $q-1$ (and so, $q \geq 3$), then $\phi(q-2) = q-1$, $\phi(q-1) = q-2$, and $\phi(x) = x$ if $x \notin \{q-2, q-1\}$.

In both cases, ϕ is an involution, that is, $\phi^{-1} = \phi$. By abuse of notation, for $\boldsymbol{w} \in A_q^*$, $\phi(\boldsymbol{w})$ is the word obtained from $\boldsymbol{w}$ by replacing each of its symbols x by $\phi(x)$, and for a list $\mathcal{L}$ of words, $\phi(\mathcal{L})$ is the list obtained from $\mathcal{L}$ by replacing each word $\boldsymbol{w}$ in $\mathcal{L}$ by $\phi(\boldsymbol{w})$.

If $\boldsymbol{f}$ is a forbidden factor that does not induce zero periodicity, then $\phi(\boldsymbol{f})$ does not end by 0 nor by $q-1$, and so it induces zero periodicity, see Remark 5. In this case $\phi(\mathcal{L})$ is a Gray code for the set $A_q^n(\boldsymbol{f})$, where $\mathcal{L}$ is the set $A_q^n(\phi(\boldsymbol{f}))$ listed in $\prec$ order for q even, and in $\lhd$ order for q odd.

6.4/ ALGORITHMIC CONSIDERATIONS

We design the efficient generating algorithm by first introducing the initial algorithm (a "naive" algorithm), in which the factor matching procedure is not efficient; then we improve the algorithm by applying efficient factor matching procedure.

6.4.1/ INITIAL GENERATING ALGORITHM

To generate factor avoiding words, a factor matching procedure is needed in the generating algorithm. Figure 6.1 shows a "naive" factor matching function FACTORMATCH, which returns *true* if the factor $\boldsymbol{f} = f_1 f_2 \dots f_\ell$ is equal to the suffix of a given word $\boldsymbol{w} = w_1 w_2 \dots w_k$, or returns $false$ otherwise. One can easily see that the worst case complexity for FACTORMATCH is given by $O(k\ell)$.

This function is applied in the generating procedure GENAVOID in Figure 6.2. GENAVOID recursively expands the current generated prefix $w_1 w_2 \dots w_{k-1}$ (k being the first parameter of GENAVOID) to $w_1 w_2 \dots w_{k-1} j$, with j covering the alphabet A_q in increasing or decreasing order, according to the value of $dir \in \{0, 1\}$, the second parameter of the procedure, which is the parity of the word $w_1 w_2 \dots w_{k-1}$. The restriction is given by function call FACTORMATCH(k) in line 8 and 14, in order to avoid generating the words containing the given factor. Since FACTORMATCH runs in $O(k\ell)$ time, one can easily conclude that GENAVOID is not efficient.

Actually, if function call FACTORMATCH(k) in line 8 and 14 are removed, GENAVOID generates efficiently (i.e., in CAT) the list for the set A_q^n in $\prec$ order or in $\lhd$ order, depending whether q is even or odd. To improve the algorithm, we modify GENAVOID to apply the efficient factor matching technique.

```
1  function FACTORMATCH(k)
2  while i ≤ k and j ≤ ℓ do
3      if w[i] = f[j] then
4          i := i + 1; j := j + 1;
5      else i := i − j + 1;
6          j := 1;
7  return i = ℓ
8  end.
```

Figure 6.1: Algorithm for checking if factor f match to the suffix of word w.

```
 1  procedure GENAVOID(k, dir)
 2  if k = n + 1 then PRINT(w);
 3  else if dir = 0 then
 4          for j := 0 to q − 1 do
 5              w[k] := j;
 6              m := (dir + j) mod 2;
 7              if q is odd and j ≠ 0 then m := (m + 1) mod 2;
 8              if FACTORMATCH(k)= false then
 9                  GENAVOID(k + 1, m);
10      else for j := q − 1 downto 0 do
11              w[k] := j;
12              m := (dir + j) mod 2;
13              if q is odd and j ≠ 0 then m := (m + 1) mod 2;
14              if FACTORMATCH(k)= false then
15                  GENAVOID(k + 1, m);
16  end.
```

Figure 6.2: Algorithm producing the set $A_q^n(f)$, listed in $\prec$ order if q is even or in $\lhd$ order if q is odd. The initial call is GENAVOID$(1, 0)$.

6.4.2/ IMPROVED GENERATING ALGORITHM

EFFICIENT FACTOR MATCHING TECHNIQUE

Given two words $\boldsymbol{w}$ and $\boldsymbol{f}$, the *overlap* of $\boldsymbol{w}$ over $\boldsymbol{f}$ is the longest proper suffix of $\boldsymbol{w}$ that is also a proper prefix of $\boldsymbol{f}$. The *border* of a non empty word $\boldsymbol{w}$ is the overlap of $\boldsymbol{w}$ and itself (see for instance [Lot05]). Thus it is the longest word $\boldsymbol{u}$ which is both a proper prefix and a proper suffix of $\boldsymbol{w}$. The overlap of $\boldsymbol{w}$ over $\boldsymbol{f}$ is denoted by overlap$(\boldsymbol{w}, \boldsymbol{f})$. Notice that $overlap(\boldsymbol{w}, \boldsymbol{f})$ is not necessarily equal to $overlap(\boldsymbol{f}, \boldsymbol{w})$. The border of $\boldsymbol{w}$ is denoted by $border(\boldsymbol{w})$. Thus, $border(\boldsymbol{w}) = overlap(\boldsymbol{w}, \boldsymbol{w})$.

```
1  procedure MAKEBORDER()
2  b[0] := −1;
3  i := 0;
4  for j := 1 to (ℓ − 1)
5      b[j] := i;
6      while (i ≥ 0 and f[j + 1] ≠ f[i + 1])
7          i := b[i];
8      i := i + 1;
9  b[ℓ] := i;
10 end.
```

Figure 6.3: Procedure computing the border array b of length ℓ forbidden factor $\boldsymbol{f}$, and used by MAKEARRAY.

Procedure MAKEBORDER in Figure 6.3, initializes array $b = b_0 b_1 b_2 \dots b_\ell$, the *border* array of $\boldsymbol{f} = f_1 f_2 \dots f_\ell$, and by convenience $b_0 = -1$. For example if $\ell = 8$ and $\boldsymbol{f} = 01001010$, then $b_0 b_1 \dots b_8 = -100112323$; and for instance, $b_5 = 2$ since 01 is the longest proper prefix which is also a suffix of $f_1 f_2 \dots f_5 = 01001$. Actually, the border array b is a main ingredient for Knuth-Morris-Pratt word matching algorithm in [KMP77] and it is initialized by an $O(\ell)$ time complexity preprocessing step, see again [Lot05].

GENERATING ALGORITHM WITH EFFICIENT FACTOR MATCHING TECHNIQUE

Here we give a generating algorithm for the set $A_q^n(\boldsymbol{f})$, $n \geq 1$, for any forbidden factor $\boldsymbol{f} \in A_q^\ell$, $\ell \geq 2$ (the case $\ell = 1$ being trivial). This generating algorithm recursively produces prefixes of words in $A_q^n(\boldsymbol{f})$, in $\prec$ order if q is even, or in $\lhd$ order if q is odd, and in particular, it generates the previously discussed Gray codes for $A_q^n(\boldsymbol{f})$. We will show that this algorithm is efficient, except for the trivial factors of the form $00\dots01$ or $11\dots10$, for which a simple transformation of them makes the generating algorithm efficient.

When the length $(\ell - 1)$ prefix of $\boldsymbol{f} = f_1 f_2 \dots f_\ell$ is a suffix of $w_1 w_2 \dots w_{k-1}$, the value f_ℓ is

skipped for j in order not to produce the forbidden factor. To do this efficiently, the third parameter, i, of procedure GENAVOID is the length of the maximal prefix of the forbidden factor $\boldsymbol{f}$ which is also a suffix of the current generated word $w_1 w_2 \dots w_{k-1}$; and h in this procedure is the length of the maximal suffix of $w_1 w_2 \dots w_{k-1} j$ which is also a prefix of $\boldsymbol{f}$, and it is given by $M_{i,j}$. So, when h is equal to ℓ (the length of the forbidden factor), the current value of j is skipped for the prefix expansion.

Now, we explain in more detail the array M used by algorithm GENAVOID. For a forbidden factor $\boldsymbol{f} = f_1 f_2 \dots f_\ell \in A_q^\ell$, the $\ell \cdot q$ size two dimensional array M is defined as: for $i \in \{0, 1, \dots, \ell - 1\}$ and $j \in \{0, 1, \dots, q - 1\} = A_q$, $M_{i,j}$ is the length of the maximal suffix of $f_1 f_2 \dots f_i j$ which, is also a prefix $\boldsymbol{f}$. For instance, for $q = 4$ and $\boldsymbol{f} = 012011 \in A_4^6$, we have

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 2 & 0 & 0 \\ \mathbf{1} & \mathbf{0} & \mathbf{3} & 0 \\ 4 & 0 & 0 & 0 \\ 1 & 5 & 0 & 0 \\ 1 & 6 & 3 & 0 \end{bmatrix},$$

and, for example (see the entries in boldface in M),

- $M_{2,0} = \mathbf{1}$, since the length of the longest suffix of $f_1 f_2 0 = 010$ which is a prefix of $\boldsymbol{f}$ is 1,
- $M_{2,1} = \mathbf{0}$, since there is no suffix of $f_1 f_2 1 = 011$ which is a prefix of $\boldsymbol{f}$,
- $M_{2,2} = \mathbf{3}$, since $f_1 f_2 2 = 012$ (of length 3) is a prefix of $\boldsymbol{f}$.

The array M is initialized, in an $O(\ell \cdot q)$ time preprocessing step, by procedure MAKEARRAY in Figure 6.5, which in turn uses array $b = b_0 b_1 b_2 \dots b_\ell$, the *border* array of $\boldsymbol{f}$ initialized by procedure MAKEBORDER in Figure 6.3.

```
 1  procedure GENAVOID(k, dir, i)
 2  if k = n + 1 then PRINT(w);
 3  else if dir = 0 then
 4          for j := 0 to q − 1 do
 5              h := M[i, j];
 6              if h ≠ ℓ then
 7                  w[k] := j;
 8                  m := 0;
 9                  if q is odd and j ≠ 0 then m := 1;
10                  GENAVOID(k + 1, (dir + j + m) mod 2,h);
11      else for j := q − 1 downto 0 do
12              h := M[i, j];
13              if h ≠ ℓ then
14                  w[k] := j;
15                  m := 0;
16                  if q is odd and j ≠ 0 then m := 1;
17                  GENAVOID(k + 1, (dir + j + m) mod 2,h);
18  end.
```

Figure 6.4: Algorithm producing the set $A_q^n(\boldsymbol{f})$, listed in $\prec$ order if q is even or in $\lhd$ order if q is odd. The initial call is GENAVOID$(1,0,0)$, and it uses array M, initialized in a preprocessing step by MAKEARRAY; and $\mathcal{S}$ is the list of symbols in the alphabet A_q in increasing or decreasing order.

```
1  procedure MAKEARRAY()
2  for j := 0 to q − 1
3      for i := 0 to ℓ − 1
4          if f[i + 1] = j then M[i, j] := i + 1;
5          else if i > 0 then M[i, j] := M[b[i], j];
6                  else M[i, j] := 0;
7  end.
```

Figure 6.5: Algorithm initializing the array M.

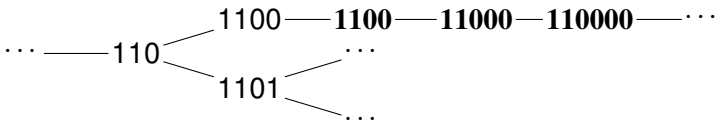

Figure 6.6: In boldface a 'branch' of words produced by consecutive degree-one calls in the generating tree of $A_2^n(001)$.

ALGORITHM ANALYSIS

Before analyzing the time complexity of the generating algorithm GENAVOID we show that, if in the underlying tree induced by recursive calls of GENAVOID there are degree-one successive calls, then $q = 2$ and the forbidden factor has the form $00\ldots01$ or $11\ldots10$. See Figure 6.6 for words in $A_2^n(001)$ produced by degree-one consecutive calls of GENAVOID.

For length $\ell \geq 2$ forbidden factor $\boldsymbol{f}$ let $\boldsymbol{w} \in A_q^*(\boldsymbol{f})$ and $i, j \in A_q$ such that $\boldsymbol{w}ij \in A_q^*(\boldsymbol{f})$ and:

- $\boldsymbol{w}k$ ends by $\boldsymbol{f}$ for any $k \in A_q$, $k \neq i$, and
- $\boldsymbol{w}ik$ ends by $\boldsymbol{f}$ for any $k \in A_q$, $k \neq j$.

In other words, when the current word is $\boldsymbol{w}$ as above, then the call of GENAVOID is a degree-one call (producing $\boldsymbol{w}i$) which in turn produces a degree-one call (producing $\boldsymbol{w}ij$). By the two conditions above, it follows that $q = 2$ and $i = j$. When $i = j = 0$, the length $\ell - 1$ suffix of $\boldsymbol{w}$ is equal to the $\ell - 1$ suffix of $\boldsymbol{w}0$, which in this case must be $0^{\ell-1}$, and finally $\boldsymbol{f} = 0^{\ell-1}1$. Similarly, when $i = j = 1$, it follows that $\boldsymbol{f} = 1^{\ell-1}0$.

Let now $\boldsymbol{f}$ be a length $\ell \geq 2$ forbidden factor, and either $q \geq 3$ or $q = 2$ and $\boldsymbol{f}$ is not $0^{\ell-1}1$ nor $1^{\ell-1}0$. In this case, by the previous considerations, each recursive call of GENAVOID is either:

- a terminal call, or
- a call producing at least two recursive calls, or
- a call producing one recursive call, which in turn is in one of the two cases above,

and by Ruskey's CAT principle in [Rus03], it follows that, with the previous restrictions on q and $\boldsymbol{f}$, GENAVOID runs in constant amortized time, and so is an efficient generating algorithm.

Nevertheless, for the particular factors above, when $\ell = 2$, $A_2^n(1^{\ell-1}0)$ is trivially the set $\{0^n, 0^{n-1}1, 0^{n-2}11, \ldots, 1^n\}$, and $A_2^n(0^{\ell-1}1)$ the set $\{0^n, 10^{n-1}, 110^{n-2}, \ldots, 1^n\}$. And for $\ell \geq 3$, both sets $A_2^n(1^{\ell-1}0)$ and $A_2^n(0^{\ell-1}1)$ can be generated efficiently in Gray code order. Indeed, for $A_2^n(1^{\ell-1}0)$ with $\ell \geq 3$ it is enough to generate (efficiently) the Gray code for $A_2^n(01^{\ell-1})$ (see Theorem 12) and then reverse each generated word; and for $A_2^n(0^{\ell-1}1)$ it is enough

to generate the Gray code for $A_2^n(1^{\ell-1}0)$ as previously, then complement each symbol in each word. The following scheme describes this method (see the example in Table 6.3):

$$A_2^n(01^{\ell-1}) \xrightarrow{\text{Reverse}} A_2^n(1^{\ell-1}0) \xrightarrow{\text{Complement}} A_2^n(0^{\ell-1}1).$$

Finally, notice that the generating order ($\prec$ or $\lhd$ in our case) does not affect the efficiency of the generating algorithm, which can obviously be modified to produce same set of factor avoiding words in lexicographical order.

$A_2^4(011)$ (a)	$A_2^4(110)$ (b)	$A_2^4(001)$ (c)
0 0 0 0	0 0 0 0	1 1 1 1
0 0 0 **1**	**1** 0 0 0	**0** 1 1 1
0 0 **1 0**	**0 1** 0 0	**1 0** 1 1
0 **1 0 1**	**1 0 1** 0	**0 1 0** 1
0 1 0 **0**	**0** 0 1 0	**1** 1 0 1
1 1 0 0	0 0 1 **1**	1 1 0 **0**
1 1 0 **1**	**1** 0 1 1	**0** 1 0 0
1 1 **1** 1	1 **1** 1 1	0 **0** 0 0
1 1 1 **0**	**0** 1 1 1	**1** 0 0 0
1 **0** 1 0	0 1 **0** 1	1 0 **1** 0
1 0 **0 1**	**1 0** 0 1	**0 1** 1 0
1 0 0 **0**	**0** 0 0 1	**1** 1 1 0

Table 6.3: (a) The set $A_2^4(011)$ listed in $\prec$ order, inducing 3-adjacent Gray code; (b) the reverse of the list in (a), giving Gray code for $A_2^4(110)$; (c) the complement of the list in (b), giving Gray code for $A_2^4(001)$. The changed symbols are in bold.

6.5/ IMPLEMENTATION ON CROSS-BIFIX-FREE SET

If necessary, the readers are advised to recall Definition 3 for general definitons of cross-bifix-free set and Section 4.3 for some related contemporary results. Following the authors of [CKPW13], the cross-bifix-free set we consider here is denoted by $S_{n,q}^{(k)}$. It is formed by length n words over the q-ary alphabet $A_q = \{0, 1, \ldots, q-1\}$ containing a particular sub-word avoiding k consecutive 0s. Now we briefly summarize its definition and we refer the reader to [CKPW13] for more details about its features.

Definition 11:

Let $n \geq 3$, $q \geq 2$ and $1 \leq k \leq n-2$. The cross-bifix-free set $S_{n,q}^{(k)}$ is the set of all length n words $s_1 s_2 \cdots s_n$ over A_q satisfying:

- $s_1 = \cdots = s_k = 0$;
- $s_{k+1} \neq 0$;
- $s_n \neq 0$;
- the sub-word $s_{k+2} \ldots s_{n-1}$ does not contain k consecutive 0s.

Below we recall the cardinality of $S_{n,q}^{(k)}$: let

$$f_{n,q}^{(k)} = \begin{cases} q^n & \text{if } 0 \leq n < k, \\ (q-1)\left(f_{n-1,q}^{(k)} + f_{n-2,q}^{(k)} + \ldots + f_{n-k,q}^{(k)}\right) & \text{if } n \geq k, \end{cases} \tag{6.6}$$

be the sequence enumerating the words of n length words over A_q avoiding k consecutive zeros [Lud81, Sch08] (observe that in the particular case of $q = 2$, the well known *k-generalized Fibonacci sequences* [Knu73] are obtained). It is not difficult to realize that, from the above description of $S_{n,q}^{(k)}$ we have:

$$|S_{n,q}^{(k)}| = (q-1)^2 f_{n-k-2,q}^{(k)}. \tag{6.7}$$

For the sake of clearness we would like to point out that the cardinality of these sets depends on n, q and k. For any fixed n and q, the largest size of $|S_{n,q}^{(k)}|$, denoted by $S(n,q)$, is obtained by a particular value of k which, at the present, is given by the following empirical expression as in [CKPW13]:

$$S(n,q) = \max\{(q-1)^2 f_{n-k-2,q}^{(k)} : 2 \leq k \leq n-2\}. \tag{6.8}$$

In Table 6.4 the values of $S(n,q)$ are shown, for $2 \leq q \leq 5$ and $4 \leq n \leq 20$. The values of k giving $S(n,2)$ increase with the growth of n. The value of k giving the remaining columns $S(n,3)$, $S(n,4)$ and $S(n,5)$ is 2 all the times: this is the reason why we do not show the respective columns with the values of k (which are expected to increase for some $n > 20$).

6.5.1/ The Gray code

For constructive reasons we give the Gray code for the set $S_{n+k,q}^{(k)}$ starting from particular list of length n words, then prepending a 0^k prefix to each word in the set. We denote the

n	$S(n,2)$	k	$S(n,3)$	$S(n,4)$	$S(n,5)$
4	1	2	4	9	16
5	2	2	12	36	80
6	3	2	32	135	384
7	5	2	88	513	1 856
8	8	2	240	1 944	8 960
9	13	2	656	7 371	43 264
10	24	3	1 792	27 945	208 896
11	44	3	4 896	105 948	1 008 640
12	81	3	13 376	401 679	4 870 144
13	149	3	36 544	1 522 881	23 515 136
14	274	3	99 840	5 773 680	113 541 120
15	504	3	272 768	21 889 683	548 225 024
16	927	3	745 216	82 990 089	2 647 064 576
17	1 705	3	2 035 968	314 639 316	12 781 158 400
18	3 136	3	5 562 368	1 192 888 215	61 712 891 904
19	5 768	3	15 196 672	4 522 582 593	297 976 201 216
20	10 671	4	41 518 080	17 146 412 424	1 438 756 372 480

Table 6.4: The values of $S(n,q)$, for $2 \leq q \leq 5$ and $4 \leq n \leq 20$.

obtained Gray code by $\mathcal{S}^{(k)}_{n+k,q}$. Our strategy in order to obtain $\mathcal{S}^{(k)}_{n+k,q}$ is the following:

1. restrict the list in $\prec$ order for $A^n_q(0^k)$ such that for $w_1 w_2 \dots w_n \in A^n_q(0^k)$, $w_1 \neq 0$ and $w_n \neq 0$. We denote the obtained list as $\mathcal{T}^{(k)}_{n,q}$.

2. prepend a 0^k prefix to each word in $\mathcal{T}^{(k)}_{n,q}$, so that the desired Gray code $\mathcal{S}^{(k)}_{n+k,q}$ is obtained.

It is easy to check that $\mathcal{T}^{(k)}_{n,q}$ is the Gray code for the set of q-ary words with no 0^k factors which begin and end by a non-zero symbol, as formalized in the following proposition:

Proposition 31:

The list $\mathcal{T}^{(k)}_{n,q}$ is a 1-Gray code for the set of q-ary words with no 0^k factors which begin and end by a non-zero symbol.

Proof. The proof is straightforward, since if we restrict a symbol in any position, then the resulting list is still a Gray code. □

Since $0^k \cdot \mathcal{T}^{(k)}_{n,q} = \mathcal{S}^{(k)}_{n+k,q}$, we have the following:

Corollary 3:

The list $\mathcal{S}^{(k)}_{n+k,q}$ is a 1-Gray code.

6.5.2/ GENERATING ALGORITHM

Cross-bifix-free words are more restrictive factor avoiding q-ary words. In order to obtain the generating algorithm for them, some modifications are applied to the algorithm GENAVOID already given in Figure 6.4.

First, we remove line 8, 9, 15, and 16 in GENAVOID, since the words are only generated with respect to RGC order (applying those lines will generate the words in RGC order for even q, and in Dual RGC order for odd q). Next, we insert in line 3 and 4 two statements to prevent assigning zero to w_1 and w_n. The resulting algorithm is GENJ, as shown in Figure 6.7.

```
1  procedure GENJ(k, dir, i)
2  if k = n + 1 then TYPE;
3  else low = 0;
4       if k = 1 or k = n then low = 1;
5       if dir = 0 then
6         for j := low to q − 1 do
7              h := M[i, j];
8              if h ≠ ℓ then
9                w[k] := j;
10               GENJ(k + 1, (dir + j) mod 2,h);
11      else for j := q − 1 downto low do
12             h := M[i, j];
13             if h ≠ ℓ then
14               w[k] := j;
15               GENJ(k + 1, (dir + j) mod 2,h);
16 end.
```

Figure 6.7: Algorithm GENJ, producing $\mathcal{J}_{n,q}^{(k)}$.

Removing line 8, 9, 15, and 16 in GENAVOID does not change its CAT complexity, since those lines contain statements running in $O(1)$. And the statements in line 3 and 4 in GENJ are obviously run in $O(1)$. These modifications implies GENJ to remain CAT.

6.6/ SUMMARY

We introduce two order relations on the set of length n q-ary words, and show that the set of words avoiding any from among the q^ℓ factors of length $\ell \geq 2$, except $\ell - 1$ or ℓ of them according to the parity of q, when listed in the appropriate order is an (at most) 3-Gray code. For each of the factors listed as exceptions, we give a simple transformation makes it possible to eventually obtain similar Gray codes. An efficient generating algorithm for

the derived Gray codes is given.

Finally, we investigate the Graycodeness of the set of cross-bifix-free words $\mathcal{S}^{(k)}_{n+k,q}$, and give an efficient generating algorithm.

7

GRAY CODES FOR SOME PATTERN AVOIDING PERMUTATIONS

In this chapter, we will investigate the Graycodeness of certain pattern avoiding permutations, listed according to Steinhaus-Johnson-Trotter (SJT) algorithm. If the permutations in an SJT-list are represented by inversion tables (which is actually the subexcedant sequences), then we obtain the list of subexcedant sequences with respect to RGC order (as shown implicitly by Dijkstra [Dij76]). Our approach is based on the idea that restricting permutations by "avoiding a set of patterns" is equivalent to restricting their corresponding inversion tables to "a restriction rule". For this purpose, we introduce the notion of restricted inversion tables. We narrow our study by considering pattern avoiding permutations that can be constructed by *succession functions*.

7.1/ STEINHAUS-JOHNSON-TROTTER ALGORITHM

SJT algorithm has come through improvement or simpler versions, such as speeding up version by Even [Eve73], loopless version by Ehrlich [Ehr73]. To give a simpler explanation of the algorithm, in the following we describe a version using 'directed integer' by Dershowitz [Der75]:

1. Generate initial permutation $12\ldots n$, and set all integers (except the smallest) to be active, and to have direction left (i.e., to face left).
2. Transpose the largest active integer L with the adjacent integer where L is facing. Having done that, if L reach the leftmost or the rightmost position, or L is facing integer larger than itself, deactivate L.
3. Reverse the direction of all integers larger than L, and activate them.
4. Back to step 2 until there are no active integers.

Table 7.1 illustrates how these steps work on generating length-4 permutations.

1	$1\ \overleftarrow{2}\ \overleftarrow{3}\ \dot{\overleftarrow{4}}$	13	$\dot{\overrightarrow{4}}\ \overrightarrow{3}\ 2\ 1$
2	$1\ \overleftarrow{2}\ \dot{\overleftarrow{4}}\ \overleftarrow{3}$	14	$\overrightarrow{3}\ \dot{\overrightarrow{4}}\ 2\ 1$
3	$1\ \dot{\overleftarrow{4}}\ \overleftarrow{2}\ \overleftarrow{3}$	15	$\overrightarrow{3}\ 2\ \dot{\overrightarrow{4}}\ 1$
4	$4\ 1\ \overleftarrow{2}\ \dot{\overleftarrow{3}}$	16	$\dot{\overrightarrow{3}}\ 2\ 1\ 4$
5	$\dot{\overrightarrow{4}}\ 1\ \overleftarrow{3}\ \overleftarrow{2}$	17	$2\ \overrightarrow{3}\ 1\ \dot{\overleftarrow{4}}$
6	$1\ \dot{\overrightarrow{4}}\ \overleftarrow{3}\ \overleftarrow{2}$	18	$2\ \overrightarrow{3}\ \dot{\overleftarrow{4}}\ 1$
7	$1\ \overleftarrow{3}\ \dot{\overrightarrow{4}}\ \overleftarrow{2}$	19	$2\ \dot{\overleftarrow{4}}\ \overrightarrow{3}\ 1$
8	$1\ \dot{\overleftarrow{3}}\ \overleftarrow{2}\ 4$	20	$4\ 2\ \dot{\overrightarrow{3}}\ 1$
9	$3\ 1\ \overleftarrow{2}\ \dot{\overleftarrow{4}}$	21	$\dot{\overrightarrow{4}}\ 2\ 1\ 3$
10	$3\ 1\ \dot{\overleftarrow{4}}\ \overleftarrow{2}$	22	$2\ \dot{\overrightarrow{4}}\ 1\ 3$
11	$3\ \dot{\overleftarrow{4}}\ 1\ \overleftarrow{2}$	23	$2\ 1\ \dot{\overrightarrow{4}}\ 3$
12	$4\ 3\ 1\ \dot{\overleftarrow{2}}$	24	$2\ 1\ 3\ 4$

Table 7.1: The steps in generating all length-4 permutations according to SJT algorithm. The initial permutation is 1234. The active integers are marked by arrows according to their directions, and the largest among them are marked by a dot. To obtain the next permutation, the dotted integer is transposed to its adjacent integer within its direction. The steps terminate when there are no active integers.

7.2/ ADDITIONAL NOTIONS

We denote by P_n the set of length-n permutations, $\mathcal{P}_n$ the SJT-list for P_n. For a set of patterns T, $P_n(T)$ denotes the set of length-n permutations that avoid T. $\mathcal{P}_n(T)$ is the (possibly scattered) sublist of $\mathcal{P}_n$, and it consists of permutations avoiding any pattern in T (recall Definition 4 for pattern avoiding permutations).

For further discussion, we will be using some notions already defined in Chapter 5: SE_n, the set of length-n subexcedant sequences; and $\mathcal{SE}_n$, the list for SE_n with respect to RGC order. The reader is suggested to recall Definition 7 about RGC order and the $\prec$ notation.

7.2.1/ THE SITES OF A PERMUTATION

The *sites* of $\pi \in P_n$ are the positions between two adjacent symbols in π, including those before the first and after the last symbol; and they are numbered, from right to left, from 1 to $n+1$, as described in the following:

$$\underset{(n+1)}{\_\_}\ \pi_1\ \underset{(n)}{\_\_}\ \pi_2\ \underset{(n-1)}{\_\_}\ \cdots\ \underset{(2)}{\_\_}\ \pi_n\ \underset{(1)}{\_\_}$$

For a permutation $\pi \in P_n(T)$, i is an *active site* in π if inserting $n+1$ into the i-th site of π yields a permutation in $P_{n+1}(T)$.

Example 11:

$13254 \in P_5(321)$ has six sites, but only site number 1 and 2 are active. Here is the permutation 13254 with all the sites being shown:
$\underset{(6)}{\_\_} 1 \underset{(5)}{\_\_} 3 \underset{(4)}{\_\_} 2 \underset{(3)}{\_\_} 5 \underset{(2)}{\_\_} 4 \underset{(1)}{\_\_}$

In particular, $132546, 132564 \in P_6(321)$. But $132\mathbf{6}54$, $13\mathbf{6}254$, $1\mathbf{6}3254$, $\mathbf{6}13254 \notin P_6(321)$, since 654 is order isomorphic to the avoided pattern 321.

In this case, we say that 13254 has two *children*, that are 132546 and 132564.

7.2.2/ BIJECTION FROM SE_n TO P_n

For a permutation $\pi = \pi_1\pi_2\ldots\pi_n \in P_n$, we call (i, j) an *inversion* if $i < j$ and $\pi_i > \pi_j$. A sequence $s = s_1 s_2 \ldots s_n$ is called the *inversion table* of a permutation $\pi = \pi_1\pi_2\ldots\pi_n \in P_n$ if $s_i = card(\{(i, j)|i < j \text{ and } \pi_i > \pi_j\})$. For example, 0102 is the inversion table of permutation 2413. It is easy to check that an inversion table is also a subexcedant sequence; and so, we denote by SE_n the set of inversion table of length n. Note that we will use the two terms interchangeably based on the context.

We use a bijection $\psi : SE_n \to P_n$, which maps an inversion table to its corresponding permutation. The bijection is defined as: $\psi(s_1) = \psi(0) = 1$, and $\psi(s_1 \ldots s_{n-1}s_n)$ is a permutation obtained by inserting n to the $(s_n + 1)$-th active site of $\psi(s_1 \ldots s_{n-2}s_{n-1})$. The proof is quite easy: since n is greater than any symbol in permutation $\pi \in P_{n-1}$, inserting n to $(s_n + 1)$-th site of $\pi \in P_{n-1}$ implies n to have s_n inversions, and vice versa.

For example, the recursive mapping $\psi(0102)$ is described as follows:

- $\psi(s_1) = \psi(0) = 1$ (with 2 sites, $\underset{(2)}{\_\_} 1 \underset{(1)}{\_\_}$)
- $\psi(s_1 s_2) = \psi(01) = 21$ (i.e., 2 is inserted in the $(s_2 + 1)$-th, or the second, site of permutation 1, the result is permutation 21 with 3 sites, $\underset{(3)}{\_\_} 2 \underset{(2)}{\_\_} 1 \underset{(1)}{\_\_}$)
- $\psi(s_1 s_2 s_3) = \psi(010) = 213$ (3 is inserted in the (s_3+1)-th, or the first, site of permutation 21, the result is permutation 213 with 4 sites, $\underset{(4)}{\_\_} 2 \underset{(3)}{\_\_} 1 \underset{(2)}{\_\_} 3 \underset{(1)}{\_\_}$)
- $\psi(s_1 s_2 s_3 s_4) = \psi(0102) = 2413$ (4 is inserted in the $(s_4 + 1)$-th, or the third, site of permutation 213, the result is permutation 2413).

Dijkstra implicitly used a relation from SE_n to P_n to design generating algorithm for P_n, so that the next permutation is obtained by an adjacent transposition to the current gen-

erated permutation [Dij76]. Dijkstra's algorithm generates looplessly SE_n according to RGC order. Each generated sequence is converted into the corresponding permutation. More precisely, it initally generates length-n sequence $0\ldots0$, which mapped to the initial permutation $12\ldots n$; and to obtain the next permutation, the change between two successive subexcedant sequences is converted as transposition between two successive permutations. With this scheme, the output is exactly the same as that from SJT algorithm. Although it was not stated explicitly, actually the Dijkstra's list for P_n is induced by bijection $\psi(SE_n)$, according to RGC order.

In other words, the list in RGC order for the set SE_n corresponds with the SJT-list for P_n. That is, if all sequences in SE_n is mapped by ψ according to RGC order, then the result is SJT-list for P_n. See Table 7.2 for example with $n = 4$.

Rank	SE_n, RGC order	$\psi(SE_n)$	Rank	SE_n, RGC order	$\psi(SE_n)$
0	0 0 0 0	1 2 3 4	12	0 **1** 2 3	4 3 **2 1**
1	0 0 0 **1**	1 2 **4 3**	13	0 1 2 **2**	**3 4** 2 1
2	0 0 0 **2**	1 **4 2** 3	14	0 1 2 **1**	3 **2 4** 1
3	0 0 0 **3**	**4 1** 2 3	15	0 1 2 **0**	3 2 **1 4**
4	0 0 **1** 3	4 1 **3 2**	16	0 1 **1** 0	**2 3** 1 4
5	0 0 1 **2**	**1 4** 3 2	17	0 1 1 **1**	2 3 **4 1**
6	0 0 1 **1**	1 **3 4** 2	18	0 1 1 **2**	2 **4 3** 1
7	0 0 1 **0**	1 3 **2 4**	19	0 1 1 **3**	**4 2** 3 1
8	0 0 **2** 0	**3 1** 2 4	20	0 1 **0** 3	4 2 **1 3**
9	0 0 2 **1**	3 1 **4 2**	21	0 1 0 **2**	**2 4** 1 3
10	0 0 2 **2**	3 **4 1** 2	22	0 1 0 **1**	2 **1 4** 3
11	0 0 2 **3**	**4 3** 1 2	23	0 1 0 **0**	2 1 **3 4**

Table 7.2: The list for SE_n in RGC order, and its corresponding permutation $\psi(SE_n)$. The list for $\psi(SE_n)$ is actually an SJT-list for P_4.

Remark 13:

If two sequences $s, t \in SE_n$ differ in one position, and by $+1$ or -1 in this position, then $\psi(s)$ and $\psi(t)$ differ by one transposition (i.e., they differ in two positions).

7.2.3/ REGULAR PATTERN AND SUCCESSION FUNCTION

Definition 12: Regular pattern

A set of patterns T is said to be *regular* if it satisfies all of the following considerations:

1. $P_1(T) = \{1\}$ and $P_2(T) = \{12, 21\}$.
2. All active sites are right justified (i.e., all the sites to the right of an active sites are also active).
3. For any $\pi \in P_n(T)$, the number of active sites of π can be determined by $\chi_T(i,k)$, where:
   - k is the number of children of π's parent (i.e., π and its *siblings*), and
   - i is the position (from the right) of the largest symbol in π.

Equivalently, $\chi_T(i,k)$ is the number of π's children. We say that $\chi_T(i,k)$ is a *succession function*.

We consider the set of regular patterns presented in Table 7.3 along with their corresponding succession functions. Note that some of these succession functions (particularly for T being T_1, T_2, or T_3) do not depend on k, and for the generality, we keep k being the parameter of the function in those cases .

Set of patterns T	Succession function $\chi_T(i,k)$	Cardinality of $P_n(T)$
$T_1 = \{312, 321\}$	$\chi_{T_1}(i,k) = 2$	2^{n-1}
$T_2 = \{321, 3412, 4123\}$	$\chi_{T_2}(i,k) = \begin{cases} 3 & \text{if } i = 1, \\ 2 & \text{otherwise.} \end{cases}$	n-th Pell number
$T_3 = \{312, 3421, 4321\}$	$\chi_{T_3}(i,k) = \begin{cases} 3 & \text{if } i = 2, \\ 2 & \text{otherwise.} \end{cases}$	n-th Pell number
$T_4 = \{p12\ldots(p-1), 321, 231\}$	$\chi_{T_4^{(p)}}(i,k) = \begin{cases} k+1 & \text{if } i = 1 \text{ and } k < p-1, \\ k & \text{if } i = 1 \text{ and } k = p-1, \\ 1 & \text{otherwise.} \end{cases}$	$(n+p-2)$-th Fibonacci number of order $(p-1)$

Table 7.3: Succession functions $\chi_T(i,k)$ for the considered set of patterns T.

7.2.4/ THE RESTRICTED INVERSION TABLES

We denote by $SE_n(T)$ the set of inversion tables of $P_n(T)$. Also, we denote by $\theta_T(u,k)$ the *restriction function* of $SE_n(T)$ such that for any $s = s_1 s_2 \ldots s_{n-1} u \in SE_n(T)$, we have $sx \in SE_{n+1}(T)$ for any x, $0 \leq x \leq \theta_T(u,k)$. The parameter k represents the number of

children of s's parent (i.e., π and its siblings), and $k = 2$ if $n = 1$ (recall that k is defined analogously in the context of succession function $\chi_T(i,k)$). Although it might happen that for some T, θ_T does not depend on k, and for the reason of generality, we keep k being function parameter of such θ_T.

The restriction function $\theta_T(u,k)$ is obtained from its corresponding succession function $\chi_T(i,k)$ by replacing i by u, and subtract all the assigned values by 1 (this due to the property that the site numbers in the permutation start from 1, while the symbols in the inversion table start from 0). See Table 7.4 for restriction functions for our considered $SE_n(T)$. Figure 7.1 shows the scheme we use to generate $P_n(T)$, that is by restricting SE_n subject to θ_T, and then mapping $SE_n(T)$ to $P_n(T)$ by ψ.

Set of patterns T	Restriction function $\theta_T(u,k)$
$T_1 = \{312, 321\}$	$\theta_{T_1}(u,k) = 1.$
$T_2 = \{321, 3412, 4123\}$	$\theta_{T_2}(u,k) = \begin{cases} 2 & \text{if } u = 0, \\ 1 & \text{otherwise.} \end{cases}$
$T_3 = \{312, 3421, 4321\}$	$\theta_{T_3}(u,k) = \begin{cases} 2 & \text{if } u = 1, \\ 1 & \text{otherwise.} \end{cases}$
$T_4^{(p)} = \{p12\ldots(p-1), 321, 231\}$	$\theta_{T_4^{(p)}}(u,k) = \begin{cases} k+1 & \text{if } u = 0 \text{ and } k < p-2 \\ k & \text{if } u = 0 \text{ and } k = p-2 \\ 0 & \text{otherwise.} \end{cases}$

Table 7.4: Restriction functions for the set of patterns T.

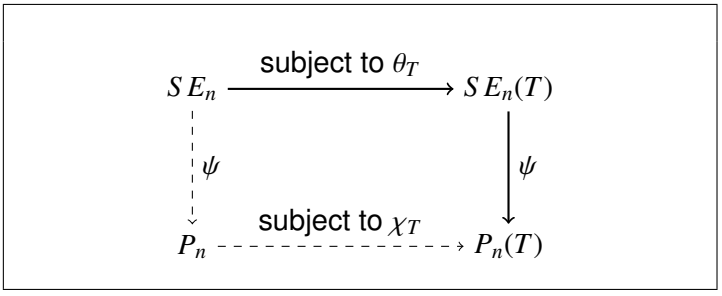

Figure 7.1: Our scheme for generating $P_n(T)$ (shown by thick arrows), that is, by restricting SE_n subject to θ_T, and then mapping $SE_n(T)$ to $P_n(T)$ by ψ.

7.3/ THE GRAY CODES

7.3.1/ THE RESULTS FOR REGULAR PATTERNS

RGC order induces a prefix partitioned list, that is, the sequences having the same prefix are contiguous; and we denote by $\texttt{last}(s_1 s_2 \dots s_m \,|\, \mathcal{L})$ (resp. $\texttt{first}(s_1 s_2 \dots s_m \,|\, \mathcal{L})$) the last (resp. the first) sequence having prefix $s_1 s_2 \dots s_m$ in the list $\mathcal{L}$. Recall that $\mathcal{SE}_n$ denotes the list of length-n subexcedant sequences with respect to RGC order. Now, we consider T being either T_1, T_2, T_3, or $T_4^{(p)}$. If $\boldsymbol{s}$ and $\boldsymbol{t}$, with $\boldsymbol{s} \prec \boldsymbol{t}$, are successive sequences in $\mathcal{SE}_n(T)$ having the longest common prefix $s_1 s_2 \dots s_{m-1}$, then $|s_m - t_m| = 1$, and for all i, $m+1 \le i \le n$, we have the following property

- $\boldsymbol{s} = \texttt{last}(s_1 s_2 \dots s_{m-1} s_m \,|\, \mathcal{SE}_n(T))$, where $s_i = \begin{cases} \theta_T(s_{i-1}, k) & \text{if } \sum_{j=1}^{i-1} s_j \text{ is even,} \\ 0 & \text{otherwise;} \end{cases}$
- $\boldsymbol{t} = \texttt{first}(t_1 t_2 \dots t_{m-1} t_m \,|\, \mathcal{SE}_n(T))$, where $t_i = \begin{cases} \theta_T(t_{i-1}, k) & \text{if } \sum_{j=1}^{i-1} t_j \text{ is odd,} \\ 0 & \text{otherwise;} \end{cases}$

Note that $s_1 s_2 \dots s_{m-1} = t_1 t_2 \dots t_{m-1}$.

The property above determines the form of two successive sequences $\boldsymbol{s}$ and $\boldsymbol{t}$ in $\mathcal{SE}_n(T)$, according to a particular restriction function $\theta_T(u, k)$ considered in Table 7.4. Remark 14, 15, and 16 in the following give the form of $\boldsymbol{s}$ and $\boldsymbol{t}$ in $\mathcal{SE}_n(T_2)$, $\mathcal{SE}_n(T_3)$, $\mathcal{SE}_n(T_4^{(p)})$, respectively. In $\mathcal{SE}_n(T_4^{(p)})$, there are several possible forms for the pair of $\boldsymbol{s}$ and $\boldsymbol{t}$; and in this case we only consider the worst case form (i.e., that give the largest Hamming distance).

In particular, $\texttt{first}(00 \,|\, \mathcal{SE}_n(T)) = 00 \dots 0$ and $\texttt{last}(01 \,|\, \mathcal{SE}_n(T)) = 010 \dots 0$; and by Remark 13, it follows that the first and the last permutations in $\mathcal{P}_n(T)$ differ in one transposition (which are, respectively, $123 \dots n$ and $2134 \dots n$). Thus, if $\mathcal{P}_n(T)$ is a Gray code, then it is also a circular Gray code.

In general, our strategy to investigate the Graycodeness are done in two steps: firstly by determining the worst case form of successive sequences $\boldsymbol{s}, \boldsymbol{t} \in \mathcal{SE}_n(T)$; and then find the form of their corresponding permutations. By finding the latter, the Graycodeness of $\mathcal{P}_n(T)$ can be determined. In the upcoming explanation, we will be using notation a^∞, which denotes the infinite copy of symbol a, i.e., $a^\infty = aaa \dots$.

PERMUTATIONS AVOIDING $T_1 = \{312, 321\}$

Proposition 32:

$\mathcal{SE}_n(T_1)$ is a 1-Gray code.

Proof. $\mathcal{SE}_n(T_1)$ is equal to the list of length-n binary sequences that begin with 0, with respect to RGC order. So that two successive sequences in $\mathcal{SE}_n(T_1)$ differ in one position

(and by 1 or -1 in this position). □

Combining Proposition 32 and Remark 13, we have the following proposition.

Proposition 33:

$\mathcal{P}_n(T_1)$ is a 2-Gray code.

PERMUTATIONS AVOIDING $T_2 = \{321, 3412, 4123\}$

Remark 14:

Let s, t be two successive sequences in $\mathcal{SE}_n(T_2)$. Let a be the longest common prefix of both s and t. Then, we have $\{s, t\} = \{au, av\}$, where u and v are prefixes of appropriate length of 0210^∞ and 1100^∞, respectively.

Proposition 34:

$\mathcal{P}_n(T_2)$ is a 4-Gray code.

Proof. Let s, t be two successive sequences in $\mathcal{SE}_n(T_2)$, and $s \prec t$. It follows that $\sigma = \psi(s)$ and $\tau = \psi(t)$ are successive permutations in $\mathcal{P}_n(T_2)$. Let m be the leftmost position where s, t differ; this implies $\psi(s_1 s_2 \dots s_{m-1}) = \psi(t_1 t_2 \dots t_{m-1}) = \sigma_1 \sigma_2 \dots \sigma_{m-1}$. We suppose that $\sum_{i=1}^{m-1} s_i$ is even. By Remark 14, we have:

- $\sigma = \psi(s_1 s_2 \dots s_{m-1} 0210 \dots 0) = \sigma_1 \sigma_2 \dots \sigma_{m-2}(m+1)\sigma_{m-1}(m+2)(m)(m+3)(m+4)\dots n$, and
- $\tau = \psi(t_1 t_2 \dots t_{m-1} 110 \dots 0) = \sigma_1 \sigma_2 \dots \sigma_{m-2}(m)(m+1)\sigma_{m-1}(m+2)(m+3)(m+4)\dots n$.

It follows that σ and τ differ in at most 4 positions.

The case when $\sum_{i=1}^{m-1} s_i$ being odd is proven analogously by exchanging the form of σ and τ. □

PERMUTATIONS AVOIDING $T_3 = \{312, 4321, 3421\}$

Remark 15:

Let s, t be two successive sequences in $\mathcal{SE}_n(T_3)$. Let a be the longest common prefix of both s and t. Then, we have $\{s, t\} \in \{au, av\}$, where u and v are prefixes of appropriate length of 1210^∞ and 0100^∞, respectively.

Proposition 35:

$\mathcal{P}_n(T_3)$ is a 4-Gray code.

Proof. Let $\boldsymbol{s}, \boldsymbol{t}$ be two successive sequences in $\mathcal{SE}_n(T_3)$ and $\boldsymbol{s} \prec \boldsymbol{t}$. It follows that $\boldsymbol{\sigma} = \psi(\boldsymbol{s})$ and $\boldsymbol{\tau} = \psi(\boldsymbol{t})$ are successive permutations in $\mathcal{P}_n(T_3)$. Let m be the leftmost position where $\boldsymbol{s}, \boldsymbol{t}$ differ; this implies $\psi(s_1 s_2 \dots s_{m-1}) = \psi(t_1 t_2 \dots t_{m-1}) = \sigma_1 \sigma_2 \dots \sigma_{m-1}$. We suppose that $\sum_{i=1}^{m-1} s_i$ is even. By Remark 15, we have:
• $\boldsymbol{\sigma} = \psi(s_1 s_2 \dots s_{m-1} 0100 \dots 0) = \sigma_1 \sigma_2 \dots \sigma_{m-2} \sigma_{m-1} (m+1)(m)(m+2)(m+3)(m+4) \dots n$, and
• $\boldsymbol{\tau} = \psi(t_1 t_2 \dots t_{m-1} 1210 \dots 0) = \sigma_1 \sigma_2 \dots \sigma_{m-2} (m+1)(m)(m+2) \sigma_{m-1} (m+3)(m+4) \dots n$.
It follows that $\boldsymbol{\sigma}$ and $\boldsymbol{\tau}$ differ in at most 4 positions.

The case when $\sum_{i=1}^{m-1} s_i$ being odd is proven analogously by exchanging the form of $\boldsymbol{\sigma}$ and $\boldsymbol{\tau}$. □

PERMUTATIONS AVOIDING $T_4^{(p)} = \{p12 \dots (p-1), 321, 231\}$

By the definition of $\theta_{T_4^{(p)}}(u, k)$ we have the fact that $\boldsymbol{s} = s_1 s_2 \dots s_{n-1} u \in SE_n(T_4^{(p)})$ has several children if $u = 0$, otherwise $\boldsymbol{s}$ has only one child. It follows that if $\boldsymbol{s}$ and $\boldsymbol{t}$ are successive sequences in $\mathcal{SE}_n(T_4^{(p)})$, then their longest common prefix is ended by 0. To investigate the Graycodeness of $\mathcal{P}(T_4^{(p)})$, we priorly need the form of $\boldsymbol{s}$ and $\boldsymbol{t}$ for $p = 3$ and $p \geq 4$.

Remark 16:

Let $\boldsymbol{s}, \boldsymbol{t}$ be two successive sequences in $\mathcal{SE}_n(T_4^{(3)})$. Let $\boldsymbol{a}$ be the longest common prefix of both $\boldsymbol{s}$ and $\boldsymbol{t}$. Then, we have $\{\boldsymbol{s}, \boldsymbol{t}\} = \{\boldsymbol{au}, \boldsymbol{av}\}$, where $\boldsymbol{u}$ and $\boldsymbol{v}$ are prefixes of appropriate length of 1010^∞ and 0100^∞, respectively.

Remark 17:

Let $\boldsymbol{s}, \boldsymbol{t}$ be two successive sequences in $\mathcal{SE}_n(T_4^{(p)})$, for a fixed $p \geq 4$. Let $\boldsymbol{a}$ be the longest common prefix of both $\boldsymbol{s}$ and $\boldsymbol{t}$. Then, their largest Hamming distance happens when $\{\boldsymbol{s}, \boldsymbol{t}\} = \{\boldsymbol{au}, \boldsymbol{av}\}$, where $\boldsymbol{u}$ and $\boldsymbol{v}$ are, respectively, prefixes of appropriate length of $0(p-2)010^\infty$ and 10100^∞ if $p \geq 4$ and even, or $0(p-3)010^\infty$ and 10100^∞ if $p \geq 5$ and odd.

Proposition 36:

$\mathcal{P}_n(T_4^{(3)})$ is a 4-Gray code.

Proof. Let $\boldsymbol{s}, \boldsymbol{t}$ be two successive sequences in $\mathcal{SE}_n(T_4^{(3)})$ and $\boldsymbol{s} \prec \boldsymbol{t}$. It follows that $\boldsymbol{\sigma} = \psi(\boldsymbol{s})$ and $\boldsymbol{\tau} = \psi(\boldsymbol{t})$ are successive permutations in $\mathcal{P}_n(T_4^{(3)})$. Let m be the leftmost position where $\boldsymbol{s}, \boldsymbol{t}$ differ; this implies $\psi(s_1 s_2 \dots s_{m-1}) = \psi(t_1 t_2 \dots t_{m-1}) = \sigma_1 \sigma_2 \dots \sigma_{m-1}$. We suppose that $\sum_{i=1}^{m-1} s_i$ is even. By Remark 16, we have:
• $\boldsymbol{\sigma} = \psi(s_1 s_2 \dots s_{m-1} 0100 \dots 0) = \sigma_1 \sigma_2 \dots \sigma_{m-2} \sigma_{m-1} (m+1)(m)(m+2)(m+3)(m+4) \dots n$, and
• $\boldsymbol{\tau} = \psi(t_1 t_2 \dots t_{m-1} 1010 \dots 0) = \sigma_1 \sigma_2 \dots \sigma_{m-2} (m) \sigma_{m-1} (m+2)(m+1)(m+3)(m+4) \dots n$.

It follows that σ and τ differ in at most 4 positions.

The case when $\sum_{i=1}^{m-1} s_i$ being odd is proven analogously by exchanging the form of σ and τ. □

Proposition 37:

For $p \geq 4$, $\mathcal{P}_n(T_4^{(p)})$ is a $2 \cdot \lfloor \frac{p}{2} \rfloor$-Gray code.

Proof. Let s, t be two successive sequences in $\mathcal{SE}_n(T_4^{(p)})$ and $s \prec t$. It follows that $\sigma = \psi(s)$ and $\tau = \psi(t)$ are successive permutations in $\mathcal{P}_n(T_4^{(p)})$. Let m be the leftmost position where s, t differ; this implies $\psi(s_1 s_2 \dots s_{m-1}) = \psi(t_1 t_2 \dots t_{m-1}) = \sigma_1 \sigma_2 \dots \sigma_{m-1}$. We suppose that $\sum_{i=1}^{m-1} s_i$ is even, and that $p \geq 4$ and p is even. By Remark 17, we have:
• $\sigma = \psi(s_1 s_2 \dots s_{m-1} 0(p-2)010 \dots 0) = \sigma_1 \sigma_2 \dots \sigma_{m-(p-2)}(m+1)\sigma_{m-(p-2)-1} \dots \sigma_{m-2}\sigma_{m-1}(m)(m+3)(m+2)(m+4)\dots n$, and
• $\tau = \psi(t_1 t_2 \dots t_{m-1} 1010 \dots 0) = \sigma_1 \sigma_2 \dots \sigma_{m-(p-2)}\sigma_{m-(p-2)-1} \dots \sigma_{m-2}(m)\sigma_{m-1}(m+2)(m+1)(m+3)(m+4)\dots n$.
Based on the worst case form above, σ and τ differ in at most p positions if $p \geq 4$ and p is even.

Now, we will prove the case when $p \geq 5$ and p is odd. Let $p = q + 1$, where $q \geq 4$ and q is even; let s, t be two successive sequences in $\mathcal{SE}_n(T_4^{(p)})$ and $s \prec t$; and let σ, τ, m be defined analogously as in the first paragraph of this proof. We suppose that $\sum_{i=1}^{m-1} s_i$ is even. By Remark 17, we have $\sigma = \psi(s_1 s_2 \dots s_{m-1} 0(p-3)010 \dots 0)$ and $\tau = \psi(t_1 t_2 \dots t_{m-1} 1010 \dots 0)$. Since $p = q + 1$, we can express σ in terms of q, that is, $\sigma = \psi(s_1 s_2 \dots s_{m-1} 0(q-2)010 \dots 0)$. It follows that σ and τ have the same form as described in the first paragraph; and so, they differ in at most q positions. This confirms that σ and τ differ in at most $p-1$ positions if $p \geq 5$ and p is odd.

The case when $\sum_{i=1}^{m-1} s_i$ being odd is proven analogously by exchanging the form of σ and τ. □

7.3.2/ THE RESULTS FOR SOME REVERSE AND INVERSE REGULAR PATTERNS

Firstly, we denote by T^{rev} the set of patterns obtained by reversing the patterns in T, and by T^{inv} that obtained by inversing the patterns in T. More precisely, $T_1^{\text{rev}} = \{213, 123\}$, $T_2^{\text{rev}} = \{123, 2143, 3214\}$, and $T_4^{(p)\text{inv}} = \{312, 321, 23 \dots p1\}$. In this section, we will show that permutations avoiding either T_1^{rev}, T_2^{rev}, or $T_4^{(p)\text{inv}}$ also yield Gray code.

Since the active sites in permutations avoiding T_1 or T_2 are right justified (recall Definition 12), it follows that reversing T_1 or T_2 produces active sites which are left justified. Based on this property, the form of successive sequences in $\mathcal{SE}_n(T^{\text{rev}})$ is given by complementing the form of s and t in $\mathcal{SE}_n(T)$ provided in Section 7.3.1.

Formally, if $\boldsymbol{s}$ and $\boldsymbol{t}$, with $\boldsymbol{s} \prec \boldsymbol{t}$, are successive sequences in $\mathcal{SE}_n(T^{\text{rev}})$ having the longest common prefix $s_1 s_2 \dots s_{m-1}$, then we have the property that $|s_m - t_m| = 1$, and

- $\boldsymbol{s} = \texttt{last}(s_1 s_2 \dots s_{m-1} s_m \,|\, \mathcal{SE}_n(T^{\text{rev}}))$,
  where $s_i = \begin{cases} i-1 & \text{if } \sum_{j=1}^{i-1} s_j \text{ is even,} \\ i-1-\theta_T((i-1)-s_{i-1},k) & \text{otherwise;} \end{cases}$
- $\boldsymbol{t} = \texttt{first}(t_1 t_2 \dots t_{m-1} t_m \,|\, \mathcal{SE}_n(T^{\text{rev}}))$,
  where $t_i = \begin{cases} i-1 & \text{if } \sum_{j=1}^{i-1} t_j \text{ is odd,} \\ i-1-\theta_T((i-1)-t_{i-1},k) & \text{otherwise;} \end{cases}$

for all i, $m+1 \le i \le n$.

The property above determines the form of two successive sequences $\boldsymbol{s}$ and $\boldsymbol{t}$ in $\mathcal{SE}_n(T_1^{\text{rev}})$ and $\mathcal{SE}_n(T_2^{\text{rev}})$, as shown in Remark 18 and 19, respectively.

In [Bar06] was shown that $P_n(T_4^{(p)\text{inv}})$ correspond with the set of length-$(n-1)$ binary words avoiding $p-1$ consecutive ones (see also earlier result in [Vaj01]). If we prepend such binary words by a 0, then we obtain the set of length-n binary words beginning by 0 and avoiding $p-1$ consecutive ones, which are identical with $SE_n(T_4^{(p)\text{inv}})$, the inversion tables of $P_n(T_4^{(p)\text{inv}})$. Based on this restriction, we can easily determine the form of two successive sequences in $\mathcal{SE}_n(T_4^{(p)\text{inv}})$, as shown in Proposition 40. The set $P_n(T_4^{(p)\text{inv}})$ is enumerated by $(n+p-2)$-th Fibonacci number of order $(p-1)$.

PERMUTATIONS AVOIDING $T_1^{\text{rev}} = \{213, 123\}$

The following proposition gives the forms of any two successive words in $SE_n(T_1^{\text{rev}})$ listed in RGC order.

Remark 18:

Let $\boldsymbol{s}, \boldsymbol{t}$ be two successive sequences in $\mathcal{SE}_n(T_1^{\text{rev}})$, and m is the leftmost position where they differ. Let $\boldsymbol{a}$ be the longest common prefix of both $\boldsymbol{s}$ and $\boldsymbol{t}$. Then, we have $\{\boldsymbol{s}, \boldsymbol{t}\} = \{\boldsymbol{a}(m-1)\boldsymbol{u}, \boldsymbol{a}(m-2)\boldsymbol{u}\}$, where $\boldsymbol{u}$ is the prefix of appropriate length of $(m-1)m(m+1)(m+2)\dots$.

Combining Remark 18 and Remark 13, we have the following proposition.

Proposition 38:

$\mathcal{P}_n(T_1^{\text{rev}})$ is a 2-Gray code.

PERMUTATIONS AVOIDING $T_2^{\text{rev}} = \{123, 2143, 3214\}$

The following proposition gives the forms of any two successive words in $SE_n(T_2^{\text{rev}})$ listed in RGC order.

Remark 19:

Let $\boldsymbol{s}, \boldsymbol{t}$ be two successive sequences in $\mathcal{SE}_n(T_2^{\text{rev}})$, and m is the leftmost position where they differ. Let $\boldsymbol{a}$ be the longest common prefix of both $\boldsymbol{s}$ and $\boldsymbol{t}$. Then, we have $\{\boldsymbol{s}, \boldsymbol{t}\} = \{\boldsymbol{au}, \boldsymbol{av}\}$, where $\boldsymbol{u}$ and $\boldsymbol{v}$ are prefixes of appropriate length of $(m-1)(m-2)(m+1)\boldsymbol{w}$ and $(m-2)(m-1)(m)\boldsymbol{w}$, respectively. In addition, $\boldsymbol{w} = (m+2)(m+1)(m+4)(m+3)(m+6)(m+5)\ldots$.

Proposition 39:

$\mathcal{P}_n(T_2^{\text{rev}})$ is a 4-Gray code.

Proof. Let $\boldsymbol{s}, \boldsymbol{t}$ be two successive sequences in $\mathcal{SE}_n(T_2^{\text{rev}})$ and $\boldsymbol{s} \prec \boldsymbol{t}$. It follows that $\boldsymbol{\sigma} = \psi(\boldsymbol{s})$ and $\boldsymbol{\tau} = \psi(\boldsymbol{t})$ are successive permutations in $\mathcal{P}_n(T_2^{\text{rev}})$. Let m be the leftmost position where $\boldsymbol{s}, \boldsymbol{t}$ differ; this implies $\psi(s_1 s_2 \ldots s_{m-1}) = \psi(t_1 t_2 \ldots t_{m-1}) = \sigma_1 \sigma_2 \ldots \sigma_{m-1}$. We suppose that $n = m+2$ and $\sum_{i=1}^{m-1} s_i$ is even. By Remark 19, we have:

- $\boldsymbol{\sigma} = \psi(s_1 s_2 \ldots s_{m-1}(m-2)(m-1)(m)) = \sigma_1(m+2)(m+1)(m)\sigma_2 \ldots \sigma_{m-1}$, and
- $\boldsymbol{\tau} = \psi(t_1 t_2 \ldots t_{m-1}(m-1)(m-2)(m+1)) = (m+2)(m)\sigma_1(m+1)\sigma_2 \ldots \sigma_{m-1}$.

Based on the worst case form above, $\boldsymbol{\sigma}$ and $\boldsymbol{\tau}$ differ in at most 4 positions. It is routine to check that if $n > m+2$, the the two successive permutations differ also in at most 4 positions. □

PERMUTATIONS AVOIDING $T_4^{(p)\text{inv}} = \{312, 321, 23\ldots p1\}$

The following proposition gives the forms of any two successive words in $SE_n(T_4^{(p)\text{inv}})$ listed in RGC order.

Proposition 40:

Let $\boldsymbol{s}, \boldsymbol{t}$ be two successive sequences in $\mathcal{SE}_n(T_4^{(p)\text{inv}})$. Let $\boldsymbol{a}$ be the longest common prefix of both $\boldsymbol{s}$ and $\boldsymbol{t}$. The worst case form of $\boldsymbol{s}, \boldsymbol{t}$ is given by $\{\boldsymbol{s}, \boldsymbol{t}\} = \{\boldsymbol{au}, \boldsymbol{av}\}$, where $\boldsymbol{u}$ and $\boldsymbol{v}$ are prefixes of appropriate length of 1010^{∞} and 0100^{∞}, respectively.

Proof. Suppose that $\sum_{i=1}^{|\boldsymbol{a}|} a_i$ is even. Clearly, $\boldsymbol{s} = \texttt{last}(\boldsymbol{a} \,|\, \mathcal{SE}_n(T_4^{(p)\text{inv}})) = \boldsymbol{a}0100\ldots0$, and $\boldsymbol{t} = \texttt{first}(\boldsymbol{a} \,|\, \mathcal{SE}_n(T_4^{(p)\text{inv}}))$. Since the sequence avoids $p-1$ consecutive ones, there are two possibilities for $\boldsymbol{t}$: if 1^{p-3} is the suffix of $\boldsymbol{a}$, then $\boldsymbol{t} = \boldsymbol{a}1010\ldots0$, otherwise $\boldsymbol{t} = \boldsymbol{a}110\ldots0$. It is routine to check that 1^{p-2} can not be suffix of $\boldsymbol{a}$, since in this case, $\boldsymbol{a}$ will be no longer the longest common prefix of $\boldsymbol{s}$ and $\boldsymbol{t}$. □

The forms of two successive sequences in $\mathcal{SE}_n(T_4^{(p)\text{inv}})$ are exactly the same with those of $\mathcal{SE}_n(T_4^{(3)})$ (see Remark 16). This similarity implies the following proposition, whose proof

is similar with that of Proposition 36.

Proposition 41:

For a $p \geq 3$, $\mathcal{P}_n(T_4^{(p)\text{inv}})$ is a 4-Gray code.

It is easy to check that $\texttt{first}(00\,|\,\mathcal{SE}_n(T_4^{(p)\text{inv}})) = 00\ldots0$ and $\texttt{last}(01\,|\,\mathcal{SE}_n(T)) = 010\ldots0$; and this property implies $\mathcal{P}_n(T_4^{(p)\text{inv}})$ being also a circular Gray code.

7.4/ ALGORITHMIC CONSIDERATIONS

The algorithm GENRSE (GENerate Restricted SubExcedant sequences) in Figure 7.2a generates $SE_k(T)$ (the inversion tables of permutations in $P_k(T)$), with k goes from 1 to n, with respect to RGC order. GENRSE is actually a slight modification of GEN1 (see Figure 5.1 in Chapter 5). The main difference is that GENRSE uses particular function THETA_X (line 6), instead of OMEGA_X as that in GEN1. We have shown that GEN1 is efficient as long as OMEGA_X runs in $O(1)$. In the current case, all the particular functions THETA_$T1$, THETA_$T2$, THETA_$T3$, and THETA_$T4$ presented in Figure 7.2b runs in $O(1)$. Analogously, as long as THETA_X runs in $O(1)$, one can verify that GENRSE satisfies the CAT principle stated in Section 3.3.1, and so it is an efficient generating algorithm.

The algorithm GENPAP (GENerate Pattern Avoiding Permutations) in Figure 7.3 is actually GENRSE with some additional commands so that the output is the SJT-list for $P_n(T)$. This is done by adding line 4, 11, and 14. The current generated inversion table is mapped by function PSI (works as function ψ defined in Section 7.2.2) to its corresponding permutation. The function UNDOPSI is needed, since the insertion of an integer (done by PSI) in a particular site (thus, a particular permutation is generated) must be undone before the same integer is inserted in the next site (in order to generate the next permutation). Skipping UNDOPSI will generate permutations having the same integer appearing more than once, which is forbidden. GENPAP remains CAT as long as the functions PSI and UNDOPSI runs in $O(1)$.

The main call of both algorithms are, respectively, GENRSE$(1,0,0,0)$ and GENPAP$(1,0,0,0)$.

```
 1  procedure GENRSE(k, x, dir, v)
 2    global: n, s;
 3    s_k := x;
 4    if k = n then PRINT(s);
 5    else if k = 1 then u := 2;
 6      else u :=THETA_X(x, v);
 7      if dir mod 2 = 0
 8      then for i := 0 to u do
 9             GENRSE(k + 1, i, i, u);
10      else for i := u downto 0 do
11             GENRSE(k + 1, i, i + 1, u);
12  end.
```

(a)

```
(i) 1  function THETA_T1(s, k)
    2    return 1;
    3  end.

(ii) 1  function THETA_T2(s, k)
    2    if s = 0 then u := 1;
    3    else u := 0;
    4    return u;
    5  end.

(iii) 1  function THETA_T3(s, k)
    2    if s = 1 then u := 2;
    3    else u := 1;
    4    return u;
    5  end.

(iv) 1  function THETA_T4(s, k)
    2    if s = 0 and k < p − 2
    3      then u := k + 1;
    4    else if s = 0 and
    5            k = p − 2
    6      then u := k;
    7    else u := 0;
    8    return u;
    9  end.
```

(b)

Figure 7.2: (a) Algorithm GENRSE, generating the list $\mathcal{SE}_n(T)$; (b) Particular function THETA_X called by GENRSE, and returning the value for $\theta_T(s_n, k)$, if T is one of the sets of patterns: (i) T_1, (ii) T_2, (iii) T_3, and (iv) $T_4^{(p)}$. The initial call is GENRSE$(1, 0, 0, 0)$.

```
 1  procedure GENPAP(k, x, dir, v)
 2    global: n, s, p;
 3    s_k := x;
 4    p :=PSI(s_k, k);
 5    if k = n then PRINT(p);
 6    else if k = 1 then u := 2;
 7         else u :=THETA_X(x, v);
 8         if dir mod 2 = 0
 9         then for i := 0 to u do
10                GENPAP(k + 1, i, i, u);
11                p :=UNDOPSI(s_{k+1});
12         else for i := u downto 0 do
13                GENPAP(k + 1, i, i + 1, u);
14                p :=UNDOPSI(s_{k+1});
15  end.
```

Figure 7.3: Algorithm GENPAP, generating the list $\mathcal{P}_n(T)$. The initial call is GENPAP$(1, 0, 0, 0)$.

7.5/ GRAPH THEORETIC CONSEQUENCES

The k-th power *transposition graph* of a set of permutation P_n is a graph whose vertices represent permutations in P_n, and edges connecting the vertices for permutations differing by at most k transposition. Particularly for $k = 1$, we refer *the first power transposition graph* of a set of permutation P_n simply as the *transposition graph*, that is, a graph whose vertices represent permutations in P_n, and edges connecting the vertices for permutations differing by a transposition.

A circular Gray code is associated with a Hamiltonian cycle in a transposition graph. If a list of permutations $(\boldsymbol{\pi}_1, \boldsymbol{\pi}_2, \ldots, \boldsymbol{\pi}_N)$ is a circular k-Gray code, then the Hamiltonian cycle in their corresponding k-th power transposition graph is given by $\boldsymbol{\pi}_1\boldsymbol{\pi}_2 \ldots \boldsymbol{\pi}_N\boldsymbol{\pi}_1$.

It is routine to check that if two permutations $\boldsymbol{s}$ and $\boldsymbol{t}$ differ in k position, then it needs at most $k-1$ transposition for transforming $\boldsymbol{s}$ to $\boldsymbol{t}$. Recall the third paragraph of Section 7.3.1 and the paragraph after the proof of Proposition 41, that $\mathcal{P}_n(T)$ is a circular Gray code for T being either T_1, T_2, T_3, $T_4^{(p)}$, or $T_4^{(p)\text{inv}}$; and since a circular Gray code is associated with a Hamiltonian cycle in a transposition graph, it follows that the following transposition graphs are Hamiltonian:

- the transposition graphs of permutations $P_n(T_1)$,
- the third power of the transposition graphs of permutations $P_n(T)$, where T is either T_2, T_3, $T_4^{(3)}$, or $T_4^{(p)\text{inv}}$ (with $p \geq 3$), and
- the $(2 \cdot \lfloor \frac{p}{2} \rfloor - 1)$-th power (with $p \geq 4$) of the transposition graphs of permutations $P_n(T_4^{(p)})$.

Figure 7.4 and 7.7 show the transposition graphs of $P_4(T_1)$ and $P_5(T_4^{(3)})$, respectively. Their Hamiltonian cycles are obtained according to the Gray codes shown in Table 7.5.

$\mathcal{P}_4(T_1)$	$\mathcal{P}_5(T_4^{(3)})$
1234	12345
1243	12354
1342	12435
1324	13254
2314	13245
2341	21435
2143	21354
2134	21345

Table 7.5: The list $\mathcal{P}_4(T_1)$ and $\mathcal{P}_5(T_4^{(3)})$, which is a 2-Gray code and a 4-Gray code, respectively.

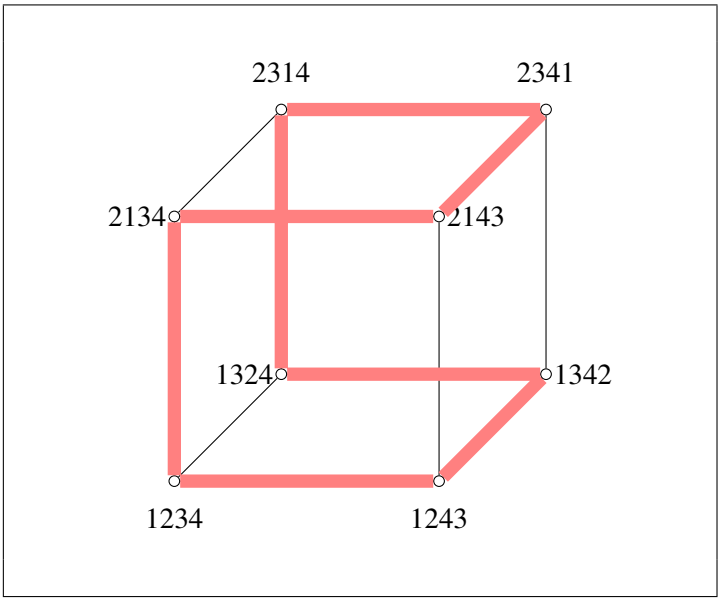

Figure 7.4: Transposition graph of $P_4(312, 321)$, with Hamiltonian cycle.

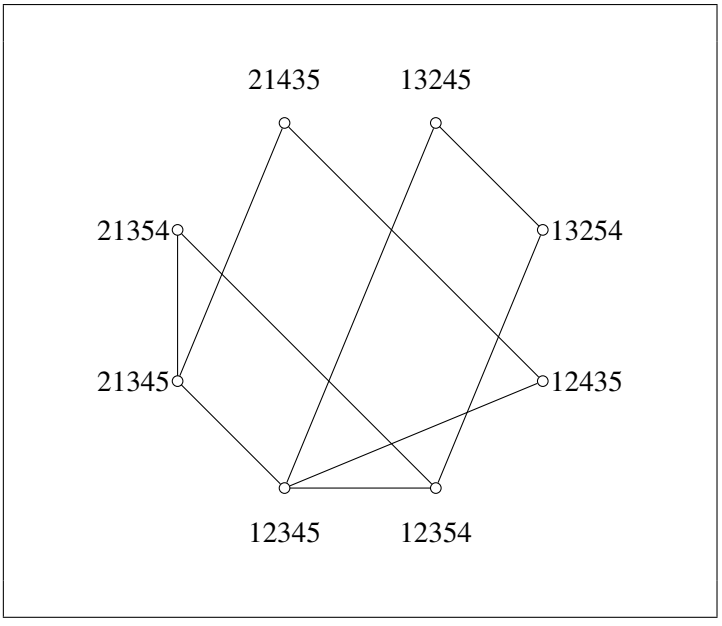

Figure 7.5: Transposition graph of $P_5(312, 321, 231)$.

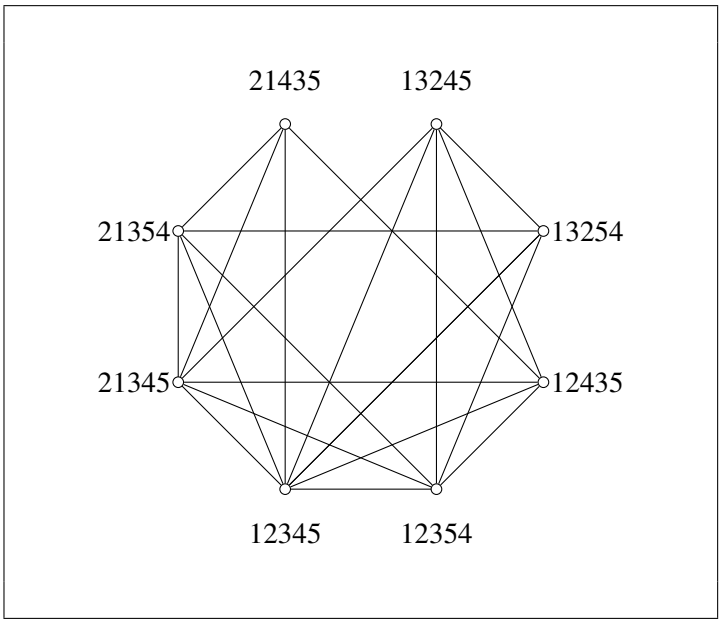

Figure 7.6: 2nd power transposition graph of $P_5(312, 321, 231)$.

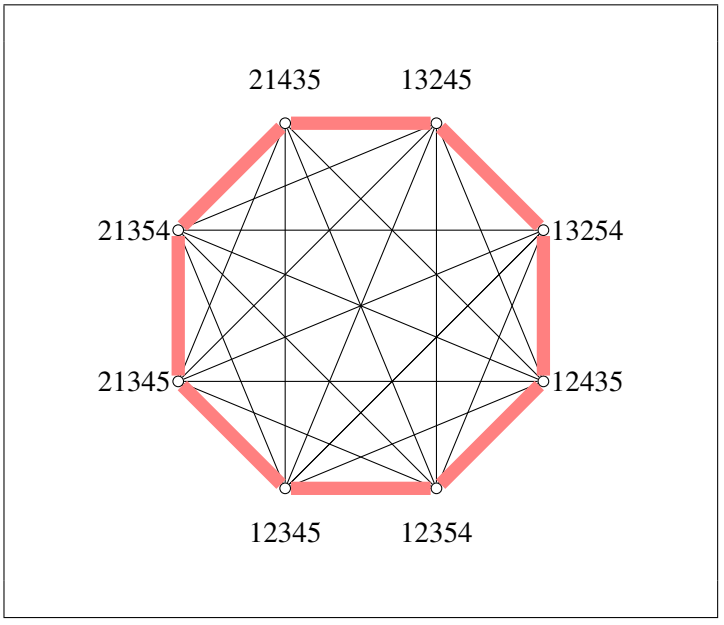

Figure 7.7: 3rd power transposition graph of $P_5(312, 321, 231)$, with Hamiltonian cycle.

7.6/ SUMMARY

We use the inversion tables to investigate the Graycodeness of an SJT-list of pattern avoiding permutations. Our approach works for some regular patterns and some of their reverse or inverse. More precisely, we show the Graycodeness of the SJT-list for $P_n(T)$, where T is either T_1, T_2, T_3, $T_4^{(p)}$, T_1^{rev}, T_2^{rev}, or $T_4^{(p)\text{inv}}$. For this purpose we introduce the notion "restricted inversion tables", which is an alternative representation of pattern avoiding permutations. The results are resumed in Table 7.6.

Set of patterns T	Graycodeness of $\mathcal{P}_n(T)$	Cardinality of $P_n(T)$
$T_1 = \{312, 321\}$	2-Gray code	2^{n-1}
$T_2 = \{321, 3412, 4123\}$	4-Gray code	n-th Pell number
$T_3 = \{312, 3421, 4321\}$	4-Gray code	n-th Pell number
$T_4^{(3)} = \{312, 321, 231\}$	4-Gray code	$(n+1)$-th Fibonacci
$T_4^{(p)} = \{p12\ldots(p-1), 321, 231\}$, $p \geq 4$	$2 \cdot \lfloor \frac{p}{2} \rfloor$-Gray code	$(n+p-2)$-th Fibonacci of order $(p-1)$
$T_1^{\text{rev}} = \{213, 123\}$	2-Gray code	2^{n-1}
$T_2^{\text{rev}} = \{123, 2143, 3214\}$	4-Gray code	n-th Pell number
$T_4^{(p)\text{inv}} = \{312, 321, 23\ldots p1\}$, $p \geq 3$	4-Gray code	$(n+p-2)$-th Fibonacci of order $(p-1)$

Table 7.6: The Graycodeness of $\mathcal{P}_n(T)$.

We also provide two efficient generating algorithms: GENRSE, that generates permutations represented by inversion tables; and GENPAP that generates permutations represented by one line notations. The two algorithms are based on efficient algorithm GEN1 discussed previously in Chapter 5.

Finally, we show the graph theoretic consequence for T being either T_1, T_2, T_3, $T_4^{(p)}$, or $T_4^{(p)\text{inv}}$. More precisely, if $\mathcal{P}_n(T)$ is a k-Gray code and circular, then the $(k-1)$-th power transposition graph of $P_n(T)$ is Hamiltonian.

III

CONCLUSION

8

CONCLUSION AND FUTURE WORKS

We investigate the RGC-based order relations on restricted classes of words and permutations as the generalization of the original RGC order. The variations of RGC order we consider in our work might have the advantage that they induce more restrictive Gray codes and more performable generating algorithms than those produced by the original RGC order. Even in some cases, the RGC based order might yield Gray code for a combinatorial class when other techniques fail to do so (such as, for example, the reflectable languages technique). In the case of statistics-defined restricted growth sequences, the Co-RGC order induces a more restrictive Gray code for ascent sequences and restricted growth functions, compared to that induced by the classical RGC order. It also allows the possibility for designing faster generating algorithm (for instance, by parallelization). In the case of factor avoiding q-ary words, the application of both RGC order for even q and Dual RGC order for odd q yield Gray codes for them; and for factors belonging to a few particular cases, the desired Gray code is obtained by applying a simple transformation technique to such factors.

In the same spirit, we use the bijective approach to define Gray code for pattern avoiding permutations. By representing permutations as inversion tables, the Steinhaus-Johnson-Trotter list of permutations of a given length can be viewed as a list of subexcedant sequences with respect to RGC order. In particular, our technique applies for some regular patterns. The main ingredients we are using in the investigation of the Graycodeness are the restriction function, the classical bijection from inversion tables to permutations, and the list of inverison tables with respect to RGC order.

The subject of defining Gray code with respect to the RGC-based order relations remains widely open. Some of the contemporary results are already given in a brief state of the art. Further works can be established such as defining Gray code for other classes of statistics-defined restricted growth sequences (for instance, for statistics ‘level’ or ‘descent’; or even other combinatorial classes) by using existing RGC-based order as we present in this thesis, or else by introducing new variations of RGC order if considered necessary. Some alternative further directions are to investigate other aspects of enumerative combinatorics (such as ranking/unranking objects, random generation, and object

searching) based on the obtained Gray code, and their implementations to other research domains.

BIBLIOGRAPHY

[Baj07] D. Bajic. On construction of cross-bifix-free kernel sets. In *2nd MCM COST 2100*, TD(07)237, Lisbon, Portugal, February 2007.

[Baj14] D. Bajic. A simple suboptimal construction of cross-bifix-free codes. *Crypt. and Communications*, 6(1):27–37, 2014.

[Bar06] J.-L. Baril. ECO generation for p generalized Fibonacci and Lucas permutations. *J. Pure Math. and Applications (Pu.M.A.)*, 17:19–37, 2006.

[Bar07] J.-L. Baril. Gray code for permutations with a fixed number of cycles. *Discr. Mathematics*, 30(13):1559–1571, 2007.

[Bar09] J.-L. Baril. More restrictive Gray codes for some classes of pattern avoiding permutations. *Inf. Proc. Letters*, 109(14):799–804, 2009.

[Bar13] J.-L. Baril. Gray code for permutations with a fixed number of left-to-right minima. *Ars Combinatoria*, 111:225–239, 2013.

[BBP06] E. Barcucci, A. Bernini, and M. Poneti. From Fibonacci to Catalan permutations. *Pu.M.A.*, 17(1–2):1–17, 2006.

[BBPV13] A. Bernini, S. Bilotta, R. Pinzani, and V. Vajnovszki. Two Gray codes for q-ary k-generalized Fibonacci strings. In *ICTCS13*, Palermo, Italy, September 2013.

[BER76] J.R. Bitner, G. Ehrlich, and E.M. Reingold. Efficient generation of the Binary Reflected Gray Code and its applications. *Comm. of the ACM*, 19:517–521, 1976.

[BGPP07] A. Bernini, E. Grazzini, E. Pergola, and R. Pinzani. A general exhaustive generation algorithm for Gray structures. *Acta Informatica*, 44(5):361–376, 2007.

[Big79] N.L. Biggs. The roots of combinatorics. *Historia Mathematica*, 6:109–136, 1979.

[BLPP99] E. Barcucci, A. Del Lungo, E. Pergola, and R. Pinzani. ECO: A methodology for the enumeration of combinatorial objects. *J. of Diff. Eq. and Applications*, 5:435–490, 1999.

[BMCDK10] M. Bousquet-Mélou, A. Claesson, M. Dukes, and S. Kitaev. $(2 + 2)$-free posets, ascent sequences and pattern avoiding permutations. *J. of Comb. Theo. Ser. A*, 7:884–909, 2010.

[BMPP12] S. Bilotta, D. Merlini, E. Pergola, and R. Pinzani. Pattern $1^{j+1}0^j$ avoiding binary words. *Fundamenta Informaticae*, 117:35–55, 2012.

[BPP12a] S. Bilotta, E. Pergola, and R. Pinzani. A construction for a class of binary words avoiding 1^j0^i. *Pu.M.A.*, 23(2):81–102, 2012.

[BPP12b] S. Bilotta, E. Pergola, and R. Pinzani. A new approach to cross-bifix-free sets. *IEEE Trans. on Inf. Theory*, 58:4058–4063, 2012.

[BPR09] J. Berstel, D. Perrin, and C. Reutenauer. *Codes and Automata (Encyclopedia of Mathematics and its Applications)*. Cambridge University Press, Cambridge, 2009.

[BS00] E. Babson and E. Steingrímsson. Generalized permutation patterns and a classification of the Mahonian statistics. *Sém. Lothar. de Combinatoire*, 44, 2000.

[BV04] J.-L. Baril and V. Vajnovszki. Gray code for derangements. *Discr. App. Mathematics*, 140:207–221, 2004.

[BV05] J.-L. Baril and V. Vajnovszki. Minimal change list for Lucas strings and some graph theoretic consequences. *Theor. Comp. Science*, 346:189–199, 2005.

[CDD+13] W.Y.V. Chen, A.Y.L. Dai, T. Dokos, T. Dwyer, and B.E. Sagan. On 021-avoiding ascent sequences. *Electr. J. of Combinatorics*, 20:P76, 2013.

[Cha89] P.J. Chase. Combination generation and Graylex ordering. *Congr. Numer.*, 69:215–242, 1989.

[CHL07] M. Crochemore, C. Hancart, and T. Lecroq. *Algorithms on Strings*. Cambridge University Press, Cambridge, 2007.

[CKPW13] Y.M. Chee, H.M. Kiah, P. Purkayastha, and C. Wang. Cross-bifix-free codes within a constant factor of optimality. *IEEE Trans. on Inf. Theory*, 59:4668–4674, 2013.

[Com74] L. Comtet. *Advanced Combinatorics, The Art of Finite and Infinite Expansions*. D. Reidel, Dordrecht, Netherlands, 1974.

[Der75] N. Dershowitz. A simplified loop-free algorithm for generating permutations. *Nordisk Tidskr. Informationsbehandling (BIT)*, 15(2):158–164, 1975.

[DF98] I. Dutour and J.-M. Fédou. Object grammars and random generation. *Discr. Math. and Theor. Comp. Science*, 2:47–61, 1998.

[DFMV08] W.M.B. Dukes, M.F. Flanagan, T. Mansour, and V. Vajnovszki. Combinatorial Gray code for classes of pattern avoiding permutations. *Theor. Comp. Science*, 396:35–49, 2008.

[Dij76] E.W. Dijkstra. On a gauntlet thrown by David Gries. *Acta Informatica*, 6:357–359, 1976.

[dLvWW00] A.J. de Lind van Wijngaarden and T.J. Willink. Frame synchronization using distributed sequences. *IEEE Trans. on Communications*, 48:2127–2138, 2000.

[DS11] P. Duncan and E. Steingrímsson. Pattern avoidance in ascent sequences. *Electr. J. of Combinatorics*, 18:P226, 2011.

[Ehr73] G. Ehrlich. Loopless algorithm for generating permutations, combinations, and other combinatorial configurations. *J. ACM*, 20(3):500–513, 1973.

[EM05] E.S. Egge and T. Mansour. Restricted permutations, Fibonacci numbers, and k-generalized Fibonacci numbers. *Integers: Electr. J. of Comb. Numb. Theory*, 5:A1, 2005.

[Er84] M.C. Er. On generating the n-ary reflected Gray code. *IEEE Trans. on Computers*, 33(8):739–741, 1984.

[Eve73] S. Even. *Algorithmic Combinatorics*. Macmillan, New York, 1973.

[FH09] D. Foata and G.-N. Han. New permutation coding and equidistribution of set-valued statistics. *Theor. Comp. Science*, 410(38–40):3743–3750, 2009.

[GO81a] L.J. Guibas and A.M. Odlyzko. Periods in strings. *J. of Comb. Theo. Ser. A*, 30:19–42, 1981.

[GO81b] L.J. Guibas and A.M. Odlyzko. String overlaps, pattern matching, and nontransitive games. *J. of Comb. Theo. Ser. A*, 30:183–208, 1981.

[Gra53] F. Gray. Pulse code communication. U.S. Patent 2632058, 1953.

[Hea63] B.R. Heap. Permutations by interchanges. *The Comp. Journal*, 6:293–294, 1963.

[HM09] S. Heubach and T. Mansour. *Combinatorics of compositions and words*. Chapman & Hall/CRC an imprint of Taylor & Francis LLC, 2009.

[JJW80] D. E. White J. Joichi and S. G. Williamson. Combinatorial Gray codes. *SIAM J. on Computing*, 9:130–141, 1980.

[Joh63] S.M. Johnson. Generation of permutations by adjacent transposition. *Math. Comp.*, 17:282–285, 1963.

[Kay76] R. Kaye. A Gray code for set partitions. *Inf. Proc. Letters*, 5(6):171–173, 1976.

[Kit11] S. Kitaev. *Patterns in Permutations and Words*. Springer-Verlag, Berlin, 2011.

[Kli81] P. Klingsberg. A Gray code for compositions. *J. of Algorithms*, 3(1):41–44, 1981.

[KMP77] D.E. Knuth, J.H. Morris, and V.R. Pratt. Fast pattern matching in strings. *SIAM J. on Computing*, 6:323–350, 1977.

[Knu68] D.E. Knuth. *The Art of Computer Programming, Volume 1*. Addison-Wesley, 1968.

[Knu73] D.E. Knuth. *The Art of Computer Programming, Volume 3*. Addison-Wesley, 1973.

[Knu11] D.E. Knuth. *The Art of Computer Programming, Volume 4A*. Addison Wesley, 2011.

[Kor01] J.F. Korsh. Loopless generation of up-down permutations. *Discr. Mathematics*, 240(1–3):97–122, 2001.

[KR92] C.W. Ko and F. Ruskey. Generating permutations of a bag by interchanges. *Inf. Proc. Letters*, 41(5):263–269, 1992.

[KT12] M.T. Keller and W.T. Trotter. *Applied Combinatorics*. Georgia Institute of Technology, 2012.

[Lai88] C.-A Laisant. Sur la numération factorielle, application aux permutations. *Bulletin de la Soc. Math. de France*, 16:176–183, 1888.

[Leh60] D.H. Lehmer. Teaching combinatorial tricks to a computer. *Proc. Sympos. Appl. Math. Combinatorial Analysis, Amer. Math. Soc.*, 10:179–193, 1960.

[Lot83] M. Lothaire. *Combinatorics on Words (Encyclopedia of Mathematics and its Applications)*. Addison-Wesley, Reading, Massachussets, 1983.

[Lot05] M. Lothaire. *Applied Combinatorics on Words*. Cambridge University Press, New York, 2005.

[LS09] Y. Li and J. Sawada. Gray codes for reflectable languages. *Inf. Proc. Letters*, 109(5):296–300, 2009.

[Lud81] J.E. Ludman. Gray code generation for MPSK signals. *IEEE Trans. on Communications*, 29:1519–1522, 1981.

[Maz10] D.R. Mazur. *Combinatorics: A Guided Tour*. The Mathematical Association of America, Washington DC, USA, 2010.

[MN08] T. Mansour and G. Nassar. Gray codes, loopless algorithm and partitions. *J. Math. Model Algor.*, 7:291–310, 2008.

[MNV11] T. Mansour, G. Nassar, and V. Vajnovzski. Loop-free Gray code algorithm for the e-restricted growth functions. *Inf. Proc. Letters*, 111:541–544, 2011.

[MS14] T. Mansour and M. Shattuck. Some enumerative results related to ascent sequences. *Discr. Mathematics*, 315–316:29–41, 2014.

[MV13] T. Mansour and V. Vajnovszki. Efficient generation of restricted growth words. *Inf. Proc. Letters*, 113(17):613–616, 2013.

[Nie73] P.T. Nielsen. A note on bifix-free sequences. *IEEE Trans. on Inf. Theory*, 29:704–706, 1973.

[NW75] S. Nijenhuis and H.S. Wilf. *Combinatorial Algorithms*. Academic Press, New York, 1975.

[NW78] S. Nijenhuis and H.S. Wilf. *Combinatorial Algorithms for Computers and Calculators*. Academic Press, New York, 1978.

[Pud14] L.K. Pudwell. Ascent sequences and the binomial convolution of Catalan numbers. *arXiv:1408.6823v1*, 2014.

[Rot81] D. Rotem. Stack sortable permutations. *Discr. Mathematics*, 33:185–196, 1981.

[RS94] F. Ruskey and C. Savage. Gray codes for set partitions and restricted growth tails. *The Austr. J. of Combinatorics*, 10:85–96, 1994.

[RS03] F. Ruskey and J. Sawada. An efficient algorithm for generating necklaces with fixed density. *SIAM J. on Computing*, 29:107–112, 2003.

[RSW12] F. Ruskey, J. Sawada, and A. Williams. Binary bubble languages and cool-lex order. *J. of Comb. Theory*, 119(1):155–169, 2012.

[Rus93] F. Ruskey. Simple Gray codes constructed by reversing sublists. *4th ISAAC (International Symposium on Algorithms and Computation), and Lecture Notes in Computer Science*, 762:201–208, 1993.

[Rus03] F. Ruskey. *Combinatorial Generation*. Dept. of Computer Science University of Victoria, Victoria, 2003.

[RW05] F. Ruskey and A. Williams. Generating combinations by prefix shifts. In *International Computing and Combinatorics Conference*, LNCS 3595, pages 570–576, Kunming, China, August 2005.

[RW09] F. Ruskey and A. Williams. The coolest way to generate combinations. *Discr. Mathematics*, 309:5305–5320, 2009.

[Sag] http://www.sagemath.org/. Accessed: 2014-09-27.

[Sav97] C. Savage. A survey of combinatorial Gray codes. *SIAM Review*, 39(4):605–629, 1997.

[Sch08] M. Schork. The r-generalized Fibonacci numbers and Polynomial coefficients. *Int. J. Contemp. Math. Sciences*, 3:1157–1163, 2008.

[Sed77] R. Sedgewick. Permutation generation methods. *Computing Survey*, 9(2):137–164, 1977.

[Squ96] M. Squire. Gray codes for A-free strings. *The Electr. J. of Combinatorics*, 3(1), 1996.

[SS85] R. Simion and F.W. Schmidt. Restricted permutations. *Europ. J. of Combinatorics*, 6:383–406, 1985.

[Sta97] R. Stanley. *Enumerative Combinatorics*, volume 1. Cambridge University Press, Cambridge, England, 1997.

[Sta99] R. Stanley. *Enumerative Combinatorics*, volume 2. Cambridge University Press, New York, 1999.

[Ste64] H. Steinhaus. *One Hundred Problems in Elementary Mathematics*. Basic Books, New York, 1964.

[Tro62] H.F. Trotter. Algorithm 115: Perm. *Comm. of the ACM*, 5(8):434–435, 1962.

[Vaj01] V. Vajnovszki. A loopless generation of bitstrings without p consecutive ones. *Discr. Math. and Theor. Comp. Science*, 2001.

[Vaj03] V. Vajnovszki. A loopless algorithm for generating the permutations of a multiset. *Theor. Comp. Science*, 307:415–431, 2003.

[Vaj07] V. Vajnovszki. Gray code order for Lyndon words. *Discr. Math. and Theor. Comp. Science*, 9(2):145–152, 2007.

[Vaj08a] V. Vajnovszki. More restrictive Gray codes for necklaces and Lyndon words. *Inf. Proc. Letters*, 106:96–99, 2008.

[Vaj08b] V. Vajnovszki. Simple Gray codes constructed by ECO method (extended abstract). In *JMIT*, Mons, Belgium, August 2008.

[Vaj12a] V. Vajnovszki. ECO-based Gray codes generation for particular classes of word (extended abstract). In *GASCom 2012*, Bordeaux, France, June 2012.

[Vaj12b] V. Vajnovszki. Lehmer code transforms and Mahonian statistics on permutations. In *Permutation Patterns*, Strathclyde, Scotland, June 2012.

[vBR92] D.R. van Baronaigien and F. Ruskey. Generating permutations with given ups and downs. *Discr. Appl. Mathematics*, 36(1):57–65, 1992.

[VV11] V. Vajnovszki and R. Vernay. Restricted compositions and permutations: from old to new Gray codes. *Inf. Proc. Letters*, 111:650–655, 2011.

[Wal00] T. Walsh. Loop-free sequencing of bounded integer compositions. *J. of Comb. Math. and Comb. Computing*, 33:323–345, 2000.

[Wal01] T. Walsh. Gray codes for involutions. *J. Comb. Math. and Comb. Computing*, 36:95–118, 2001.

[Wal08] T. Walsh. Generating Gray codes in $O(1)$ worst-case time per word. In *4th Discrete Mathematics and Theoretical Computer Science Conference*, LNCS 2731, pages 71–88, Dijon, France, July 2008.

[Wel61] M.B. Wells. Generation of permutations by transposition. *Math. Comp.*, 15:192–195, 1961.

[Wik] Sage (mathematics software). http://en.wikipedia.org/wiki/Sage_%28mathematics_software%29. Accessed: 2014-09-27.

[Wil85] S. G. Williamson. *Combinatorics for computer science*. Computer Science Press, Rockville, Maryland, 1985.

[Wil88] H.S. Wilf. Generalized Gray codes. In *SIAM Conference on Discrete Mathematics*, 1988.

[Wil89] H.S. Wilf. *Combinatorial Algorithms: an Update*. SIAM, 1989.

[Wil09a] A. Williams. Loopless generation of multiset permutations by prefix shifts. In *Symposium on Discrete Algorithms (SODA)*, New York, USA, January 2009.

[Wil09b] A. Williams. *Shift Gray Codes*. PhD thesis, University of Victoria, 2009.

LIST OF FIGURES

List of Tables

IV

APPENDICES

A

Software Implementation

Sage (previously SAGE, System for Algebra and Geometry Experimentation) is mathematical software with features covering many aspects of mathematics, including algebra, combinatorics, numerical mathematics, number theory, and calculus [Wik]. Sage is a free open-source mathematics software system licensed under the GPL. It builds on top of many existing open-source packages: NumPy, SciPy, matplotlib, Sympy, Maxima, GAP, FLINT, R and many more. Its mission: Creating a viable free open source alternative to Magma, Maple, Mathematica and Matlab [Sag].

Sage also provides cloud computing (https://cloud.sagemath.com/) that one can access freely. All of our efficient algorithms in this thesis have been tested and implemented by using Sage. In the following we give the Sage code for them.

A.1/ Generating restricted growth sequences

A.1.1/ With respect to $\prec$ order: Algorithm Gen1

```
#Generating restricted growth sequences in RGC order

#initialization
reset()
n=6  # length of the sequences, changeable
b=[]
for i in range(n+1): b.append(0)
L=[]

def type():
    t=tuple(b[1:n+1])
    L.append(t)
    print L.index(t)+1,t
```

```
def Omega_SE(w,p):
    return p

def Omega_A(w,p):
    if p>1 and b[p]>b[p-1]: return w+2
    else: return w+1

def Omega_R(w,p):
    if p>1 and b[p]>w: return w+2
    else: return w+1

def Omega_S(w,p):
    return b[p]+1

#Generating procedure, change XX in the function call
#with SE, A, R, or S to generate the desired list
def Gen1(k,x,drc,v):
    b[k]=x
    if k==n:
        type()
    else:
        u=Omega_XX(v,k)
        if drc%2==0:
            for i in range(u+1):
                Gen1(k+1,i,i,u-1)
        else:
            for i in range(u,-1,-1):
                Gen1(k+1,i,i+1,u-1)

#Main call
Gen1(1,0,0,0)
```

A.1.2/ WITH RESPECT TO $\prec_c$ ORDER: ALGORITHM GEN2

```
#Generating restricted growth sequences in Co-RGC order

#initialization
reset()
n=5   # length of the sequences, changeable
```

```
b=[]
for i in range(n+2): b.append(i-1) #(i=1;i<=n;i++) b[i]=i-1;
L=[]

#type the sequence
def type():
    t=tuple(b[1:n+1])
    L.append(t)
    print L.index(t)+1,t

#functions on subexcedant sequences
def mu_SE(k,i,w):
    if (k==n): return n-1
    else: return (w-1)

def DegreeOne_SE(m,v):
    if m==2:return 1
    else: return 0

def Lowest_SE(m):
    return 0

def SecLargest_SE(m,w):
    return m-3

#functions on ascent sequences
def mu_A(k,i,w):
    if i>=w: return i
    elif i>=b[k+1]: return w
    elif i< b[k+1]: return w-1

def DegreeOne_A(m,v):
    if v==m-1 or (v==m-2 and b[m]==0):
        b[m-1]=m-2
        return 1
    else: return 0

def Lowest_A(m):
    return 0

def SecLargest_A(m,w):
```

```
        if (w==m-2 and b[m]>0 and b[m]<m-1):
            return b[m]-1
        else: return (m-3)

    #functions on restricted growth functions
    def mu_R(k,i,w):
        if i>=w: return i
        elif b[k+1]<w: return w
        else: return w-1

    def DegreeOne_R(m,v):
        if (v==m-1) or (v==m-2 and b[m]<m-2):
            b[m-1]=m-2
            return 1
        else: return 0

    def Lowest_R(m):
        return 0

    def SecLargest_R(m, w):
        return m-3

    #functions on staircase words
    def mu_S(k,i,w):
        return i

    def DegreeOne_S(m,v):
        if b[m]==m-1:
            b[m-1]=m-2
            b[m-2]=m-3
            return 1
        else: return 0

    def Lowest_S(m):
        if b[m]>1 and b[m]<=m-2: return b[m]-1
        else: return 0

    def SecLargest_S(m,w):
        return m-3
```

```
#Generating procedure, change XX in the function call
#with SE, A, R, or S to generate the desired list
def Gen2(k,x,drc,v):
    b[k]=x
    u=mu_XX(k,x,v)
    if DegreeOne_XX(k,u)==1:
        type()
    else:
        c=Lowest_XX(k)
        d=SecLargest_XX(k,u)
        if drc%2==1:
            for i in [c..d]:
                Gen2(k-1,i,i,u)
        Gen2(k-1,k-2,k-1-(drc%2),u)
        if drc%2==0:
            for i in reversed [c..d]:
                Gen2(k-1,i,i+1,u)

#Main call
Gen2(n+1,0,0,0)
```

A.2/ GENERATING FACTOR AVOIDING q-ARY WORDS

A.2.1/ ALGORITHM GENAVOID

```
#Generating $q$-ary words avoiding a given factor f

#initialization
reset()
n=6 # length of the sequences, changeable
q=3 # -arity, changeable
factor=[0,1,1,2] #forbidden factor, changeable
#prepend it with a 0
#ex: if  f=112, then factor=[0,1,1,2]
size_f=len(factor)-1
border=[-1]
for i in [1..size_f]:
    border.append(0)
M=matrix(size_f,q,0)
b=[]
```

```
for i in range(n+1):
    b.append(0)
L=[]

def type():
    t=Sequence(b[1:n+1])
    L.append(t)
    print L.index(t)+1,t

def make_border():
    i=0
    for j in [1..size_f-1]:
        border[j]=i
        while i>=0 and factor[j+1]!=factor[i+1]:
            i=border[i]
        i=i+1
    border[size_f]=i

def make_array():
    for j in [0..q-1]:
        for i in [0..size_f-1]:
            if factor[i+1]==j:
                M[i,j]=i+1
            elif i>0:
                M[i,j]=M[border[i],j]
            else:
                M[i,j]=0

def GenAvoid(j,drc,i):
    if(j==n+1):
        type()
    else:
        if i==size_f-1:
            u=factor[size_f]
        else:
            u=-1
        if drc==0:
            for k in [0..q-1]:
                if k!=u:
                    b[j]=k
                    m=0
```

```
                    if q%2==1 and k!=0:
                        m=1
                    GenAvoid(j+1,(drc+k+m)%2, M[i,k])
            else:
                for k in reversed [0..q-1]:
                    if k!=u:
                        b[j]=k
                        m=0
                        if q%2==1 and k!=0:
                            m=1
                        GenAvoid(j+1,(drc+k+m)%2, M[i,k])

#Main call
make_border()
make_array()
GenAvoid(1,0,0)
```

A.2.2/ ALGORITHM GENJ

```
#Generating cross-bifix-free set

reset()
n=3 # length of the sequences, changeable
q=4 # -arity, changeable
factor=[0,0,0] # forbidden factor, changeable
# prepend it with a 0
# ex: if  f=00, then factor=[0,0,0]
size_f=len(factor)-1
border=[-1]
for i in [1..size_f]:
    border.append(0)
M=matrix(size_f,q,0)
b=[]
for i in range(n+1):
    b.append(0)
L=[]

def type():
    t=factor[1:len(factor)]+(Sequence(b[1:n+1]))
    L.append(t)
```

```
        print L.index(t)+1,t

    def make_border():
        i=0
        for j in [1..size_f-1]:
            border[j]=i
            while i>=0 and factor[j+1]!=factor[i+1]:
                i=border[i]
            i=i+1
        border[size_f]=i

    def make_array():
        for j in [0..q-1]:
            for i in [0..size_f-1]:
                if factor[i+1]==j:
                    M[i,j]=i+1
                elif i>0:
                    M[i,j]=M[border[i],j]
                else:
                    M[i,j]=0

    def GenJ(j,drc,i):
        if(j==n+1):
            type()
        else:
            if i==size_f-1:
                u=factor[size_f]
            else:
                u=-1
            if j==1 or j==n:
                lowest=1
            else:
                lowest=0
            if drc==0:
                for k in [lowest..q-1]:
                    if k!=u:
                        b[j]=k
                        m=0
                        if q%2==1 and k!=0:
                            m=1
                        GenJ(j+1,(drc+k+m)%2, M[i,k])
```

```
        else:
            for k in reversed [lowest..q-1]:
                if k!=u:
                    b[j]=k
                    m=0
                    if q%2==1 and k!=0:
                        m=1
                    GenJ(j+1,(drc+k+m)%2, M[i,k])

#Main call
make_border()
make_array()
GenJ(1,0,0)
```

A.3/ GENERATING PATTERN AVOIDING PERMUTATIONS

A.3.1/ ALGORITHM GENPAP

```
#Generating several classes of pattern avoiding permutations

reset()
n=5    # length of the permutations, changeable
s=[]
for i in range(n+1): s.append(0)  #initial inversion table
L=[]                              #initial list of inversion tables
perm=[]
for i in range(n+1): perm.append(0) #initial permutation
P=[]                              #initial list of permutations

def type_inv_tab():
    t=Sequence(s[1:n+1])
    L.append(t)
    print L.index(t)+1,t

def type_perm():
    perm2=Permutation(perm[0:n])
    perm1=Permutation(perm2).reverse()
    P.append(perm1)
    print P.index(perm1)+1,perm1
```

```
def Psi(s,k):
    perm.insert(s,k)

def UndoPsi(s):
    perm.pop(s)

#T1={312,321}
def theta_T1(s,k):
    return 1

#T2={321,3412,4123}
def theta_T2(s,k):
    if s==0:
        u=2
    else:
        u=1
    return u

#T3={312,3421,4321}
def theta_T3(s,k):
    if s==1:
        u=2
    else:u=1
    return u

#T4={p12...(p-1),321,231}
def theta_T4(s,k):
    p=5  # changeable
    if s==0 and k<p-2: u=k+1
    elif s==0 and k==p-2:u=k
    else:u=0
    return u

#Generating procedure, change XX in the function call
#with T1,T2,T3, or T4 to generate the desired list
def GenPAP(k,x,drc,v):
    s[k]=x
    Psi(s[k],k)
    if k==n:
        type_inv_tab()   #choose output: inversion tables, or
```

```
            #type_perm()     #permutations  (uncomment the desired output)
        else:
            if k==1:u=1
            else: u=theta_XX(x,v)
            if drc%2==0:
                for i in [0..u]:
                    GenPAP(k+1,i,i,u)
                    UndoPsi(s[k+1])
            else:
                for i in reversed[0..u]:
                    GenPAP(k+1,i,i+1,u)
                    UndoPsi(s[k+1])

    #Main call
    GenPAP(1,0,0,0)
```

B

Our publications

Part of the results presented in this thesis have been published in journals or presented in a conference. Here are the list of our articles related to this thesis.

1. A. Sabri, V. Vajnovszki. Two Reflected Gray Code based orders on some restricted growth sequences. To appear in *The Computer Journal* (published online on 18 March 2014, doi:10.1093/comjnl/bxu018).

2. A. Bernini, S. Bilotta, R. Pinzani, A. Sabri, V. Vajnovszki. Gray code orders for q-ary words avoiding a given factor. To appear in *Acta Informatica*, (published online on 7 March 2015, doi:10.1007/s00236-015-0225-2).

3. A. Bernini, S. Bilotta, R. Pinzani, A. Sabri, V. Vajnovszki. Prefix partitioned Gray codes for particular cross-bifix–free sets. *Cryptography and Communications*, 6(4):359–369, 2014.

4. A. Sabri, V. Vajnovszki. Restricted Steinhaus-Johnson-Trotter list. In *9th International colloquium on graph theory and combinatorics (ICGT)*, Grenoble, France, 30 June–4 July 2014.

Document generated with LaTeX and:
the LaTeX style for PhD Thesis created by S. Galland — http://www.multiagent.fr/ThesisStyle
the tex-upmethodology package suite — http://www.arakhne.org/tex-upmethodology/

Abstract:

We consider Gray codes and efficient exhaustive generating algorithms for the sets belonging to three major classes of restricted words, that are: (1) restricted growth sequences, (2) factor avoiding q-ary words, and (3) pattern avoiding permutations. For the first two classes, our Gray codes (and thus, our generating algorithms) are based on order relations obtained by specializing known order relations; namely Reflected Gray Code (RGC) order and its variations, and we call them *Reflected Gray Code based orders*. The Gray code and the generating algorithm for the third class are based on *Steinhaus-Johnson-Trotter* order, that is, order relation induced by Steinhaus-Johnson-Trotter Gray code for permutations.
In the first results, we define Gray codes and give efficient generating algorithms for the class of restricted growth sequences that satisfy our prescribed properties. In particular, we focus on four mainstream subclasses: subexcedant and ascent sequences, restricted growth functions and staircase words. The results are given in two parts: by using original RGC order and *Co-RGC order*, which generates prefix (and suffix, respectively) partitioned Gray codes; and we give comparison between the two results. In addition, we investigate the Graycodeness of the restricted ascent sequences.
In the second results, we define Gray codes and give an efficient generating algorithm for the class of factor avoiding q-ary words. Among the involved tools, we make use of original RGC order for even q and *Dual RGC order* for odd q, the *zero periodicity property*, and word matching techniques adapted from that of Knuth-Morris-Pratt. We give the implementation of these results to define Gray code and generating algorithm for cross-bifix-free sets.
In the third results, we define Gray codes and give efficient generating algorithms for the class of pattern avoiding permutations. In particular, we show that the Steinhaus-Johnson-Trotter Gray code for permutations, when restricted by avoiding some set of patterns, still remains a (possibly less restricted) Gray code. The main ingredients we are using in the investigation of the Graycodeness are: succession functions, the classical bijection from inversion tables to permutations, and the list of inversion tables with respect to RGC order. We give additional results on graph theoretic consequences.

Keywords: Gray code, generating algorithm, order relation, restricted words

Résumé :

Nous introduisons des codes de Gray et des algorithmes efficaces de génération exhaustive pour trois classes de mots: (1) suites à croissance restreinte, (2) mots évitant un facteur spécifié, (3) permutations à motif exclus.
Pour les deux premières classes, nos codes de Gray (et les algorithmes de génération qui en découlent) sont basés sur des relations d'ordre obtenues par la spécialisation de l'ordre du code de Gray réfléchi.
Pour la troisième classe, les codes de Gray et les algorithmes de génération correspondants sont basés sur l'ordre induit par l'algorithme de Steinhaus-Johnson-Trotter pour la génération des permutations.
Concernant les suites à croissance restreinte, nous définissons un code de Gray et donnons un algorithme de génération exhaustive pour ce code. En particulier, nous considérons les suites sous-excédantes et ascendantes, les fonctions à croissance restreinte et les mots 'escalier'. Les relations d'ordre considérées sont *RGC* et *Co-RGC*, qui sont des relations partitionnant les listes selon, respectivement, le préfixe et le suffixe. De plus, nous explorons la possibilité pour l'obtention des codes de Gray pour les suites ascendantes restreintes.
Pour les mots de q-aires à facteur interdit nous donnons deux codes de Gray et les algorithmes de génération correspondants. Les relations d'ordre considérées sont *RGC*, pour q pair, et *Dual RGC* pour q impair. Parmi les notions utilisées, citons la périodicité zéro et un algorithme classique de recherche de motif du à Knuth, Morris et Pratt. Comme application, nous considérons les ensembles 'cross-bifix-free'.
Finalement, des résultats similaires sont obtenus pour certaines classes de permutations à motif interdit. Plus précisément, nous montrons que la restriction du code de Gray de Steinhaus-Johnson-Trotter aux ensembles de permutations évitant certains motifs reste un code de Gray (moins restrictif). Parmi les techniques utilisées, nous mentionnons la fonction de succession et une bijection classique entre permutations et tableaux d'inversions, et donnons quelques conséquences en théorie des graphes.

Mots-clés : Code de Gray, algorithme de génération, relation d'ordre, croissante restreinte

■ École doctorale SPIM - Université de Bourgogne/UFR ST BP 47870 F - 21078 Dijon cedex
■ tél. +33 (0)3 80 39 59 10 ■ ed-spim@univ-fcomte.fr ■ www.ed-spim.univ-fcomte.fr