Novel universal cycle constructions for a variety of combinatorial objects

by

Chi Him Wong

A Thesis
Presented to
The University of Guelph

In partial fulfilment of requirements
for the degree of
Doctor of Philosophy
in
Computer Science

Guelph, Ontario, Canada

© Chi Him Wong, April, 2015
Novel universal cycle constructions for a variety of combinatorial objects

Chi Him Wong
University of Guelph, 2015

Advisor:
Dr. Joe Sawada

The cyclic sequence 0000100110101111 has the unlikely property that the 16 unique binary substrings of length 4 appear exactly once in the sequence as a substring. This sequence is an example of a universal cycle. A universal cycle for an arbitrary set \( \mathcal{S} \) is a cyclic sequence of length \( |\mathcal{S}| \) whose substrings of length \( n \) encode \( |\mathcal{S}| \) distinct instances in \( \mathcal{S} \). When \( \mathcal{S} \) is the set of \( k \)-ary strings of length \( n \), these sequences are commonly studied under the name de Bruijn sequence.

Universal cycles have been studied for a wide variety of combinatorial objects including permutations, partitions, subsets, multisets, labeled graphs, various functions and passwords. The study of universal cycles has a long history with applications including dynamic connections in overlay networks, genomics, software calculation of the ruler function in computer words, and indexing a 1 in a computer word. There are standard proofs to demonstrate the existence of universal cycles for a variety of combinatorial objects; however, only a small number of universal cycles can be constructed efficiently and practically. This thesis provides novel efficient constructions to generate universal cycles for a variety of combinatorial objects.

Our research leads to several new results. Firstly, we propose a novel shift rule to construct de Bruijn sequence for length \( n \) binary strings. The shift rule provides a simple and efficient successor rule to find the next bit in the de Bruijn sequence and is applicable to all values of \( n \). We then extend the shift rule to construct de Bruijn sequence for \( k \)-ary strings of length \( n \). Secondly, we generalize the Fredricksen-Kessler-Maiorana (FKM) construction and Greedy construction to generate universal cycles for a broad class of \( k \)-ary strings. We also prove that the universal cycles produced
are the lexicographically smallest for the sets. Lastly, we provide the first known efficient construction to generate universal cycles for length $n$ binary strings where the number of 1s range from $c$ to $d$ given $0 \leq c < d \leq n$. We also prove the existence of universal cycles for other combinatorial objects including subsets of passwords and labeled graphs.
Acknowledgments

First and foremost, I would like to express my sincere gratitude to my advisor Joe Sawada for his continuous support of my graduate study and research, for his patience, motivation, enthusiasm, immense knowledge, and for the opportunity he gave me to pursue my graduate study and work with him. Without his supervision and constant help this thesis would not be possible.

Besides my advisor, I would also like to thank Aaron Williams (my “academic uncle”) for the interesting and stimulating discussions, and insights he has shared. This thesis would also not be possible without standing on the shoulder of a giant like Frank Ruskey (my “academic grandfather”) whose research works have provided many of the foundations of our research.

My sincere thanks also go to my committee, Steve Gismondi, Charlie Obimbo and Yang Xiang, and the external examiner Anant Godbole for their encouragements and insightful comments. I am also in debt to Fangju Wang for his advices and his help on various aspects of my graduate study. I am also grateful to have met lots of wonderful friends in Guelph over the years, and I would like to thank them for their constant support and encouragement.

Last but not the least, I would like to thank my parents and brother for supporting me in this intellectual venture.
# Table of Contents

List of Figures vii

List of Tables viii

1 Introduction 1

1.1 Universal cycles and de Bruijn sequences . . . . . . . . . . . . . . . . . . 3
1.2 Thesis statement . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5
1.3 Overview of thesis . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6

2 Literature review and relevant results 7

2.1 Notations and definitions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7
2.2 De Bruijn graphs and existence results of universal cycles . . . . . . . . 9
2.3 De Bruijn sequence constructions . . . . . . . . . . . . . . . . . . . . . . . . . 12
   2.3.1 Greedy construction . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 13
   2.3.2 Feedback shift register sequence . . . . . . . . . . . . . . . . . . . . . 13
   2.3.3 Lempel’s recursive construction . . . . . . . . . . . . . . . . . . . . . 15
   2.3.4 The FKM construction . . . . . . . . . . . . . . . . . . . . . . . . . . . . 18
   2.3.5 Doubly recursive approach . . . . . . . . . . . . . . . . . . . . . . . . 19
2.4 Other universal cycle construction . . . . . . . . . . . . . . . . . . . . . . 20
   2.4.1 The cool-daddy construction . . . . . . . . . . . . . . . . . . . . . . . 20
2.5 Applications . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 22
2.6 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 24

3 A simple de Bruijn sequence construction 25

3.1 Shift rule to generate de Bruijn sequence over a binary alphabet . . . 25
3.2 Proving the correctness of the shift rule over a binary alphabet . . . 27
3.3 Efficient implementation for the shift rule over a binary alphabet . . 30
3.4 Translate the shift rule over a binary alphabet into NLFSR . . . . . 34
3.5 Generalizing the shift rule to general alphabets . . . . . . . . . . . . . 35
3.6 Proving the correctness of the shift rule over a general alphabet . . 36
3.7 Efficient implementation for the shift rule over a general alphabet . 41
3.8 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 48

4 Generalizations of the FKM construction and greedy construction 49

4.1 A broad class of $k$-ary strings $C(n, k)$ . . . . . . . . . . . . . . . . . . 49
4.2 $k$-suffix languages and $k$-suffix posets . . . . . . . . . . . . . . . . . . 52
4.3 The generalized FKM construction . . . . . . . . . . . . . . . . . . . . . . 55
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.4</td>
<td>Greedy approach</td>
<td>60</td>
</tr>
<tr>
<td>4.5</td>
<td>Combinatorial objects in $\mathcal{C}(n,k)$</td>
<td>62</td>
</tr>
<tr>
<td>4.6</td>
<td>The generalized FKM algorithm</td>
<td>64</td>
</tr>
<tr>
<td>4.7</td>
<td>An even broader language $\mathcal{C'}(n,k)$</td>
<td>66</td>
</tr>
<tr>
<td>4.8</td>
<td>Summary</td>
<td>71</td>
</tr>
<tr>
<td>5</td>
<td>An efficient universal cycle construction for weight-range binary strings</td>
<td>73</td>
</tr>
<tr>
<td>5.1</td>
<td>Definitions</td>
<td>73</td>
</tr>
<tr>
<td>5.2</td>
<td>Gluing universal cycles</td>
<td>74</td>
</tr>
<tr>
<td>5.3</td>
<td>Existence of weight-range universal cycle</td>
<td>75</td>
</tr>
<tr>
<td>5.4</td>
<td>Construction of weight-range universal cycle</td>
<td>76</td>
</tr>
<tr>
<td>5.5</td>
<td>Other applications of the Gluing lemma</td>
<td>86</td>
</tr>
<tr>
<td>5.5.1</td>
<td>Passwords</td>
<td>87</td>
</tr>
<tr>
<td>5.5.2</td>
<td>Strings with content-range and sum-range over a general alphabet</td>
<td>88</td>
</tr>
<tr>
<td>5.5.3</td>
<td>Labeled graphs</td>
<td>88</td>
</tr>
<tr>
<td>5.6</td>
<td>Summary</td>
<td>89</td>
</tr>
<tr>
<td>6</td>
<td>Summary and open problems</td>
<td>90</td>
</tr>
<tr>
<td>6.1</td>
<td>Open problems related to shift rule</td>
<td>91</td>
</tr>
<tr>
<td>6.2</td>
<td>Open problems related to the FKM and greedy constructions</td>
<td>92</td>
</tr>
<tr>
<td>6.3</td>
<td>Other open problems</td>
<td>93</td>
</tr>
<tr>
<td>Bibliography</td>
<td></td>
<td>97</td>
</tr>
<tr>
<td>A</td>
<td>C code of optimized shift-based algorithm to generate a de Bruijn sequence over a binary alphabet</td>
<td>105</td>
</tr>
<tr>
<td>B</td>
<td>C code of optimized shift-based algorithm to generate a de Bruijn sequence over a general alphabet</td>
<td>107</td>
</tr>
<tr>
<td>C</td>
<td>C code of optimized weighted FKM construction over a binary alphabet</td>
<td>109</td>
</tr>
</tbody>
</table>
List of Figures

2.1 The de Bruijn graph $G(B(4))$. .................................................. 10
2.2 Concatenation of aperiodic prefixes of length 8 binary necklaces with weight 4 in reverse cool-lex ordering. The resulting sequence is a shorthand universal cycle for length 8 binary strings with weight 4. ....... 21

3.1 A robot finding an Euler cycle in a de Bruijn network. ................. 26
3.2 The incoming edges and outgoing edges of two vertices in the de Bruijn graph for $T(6, 6)$ with respect to the application of $f_k$. ................. 38

4.1 The Hasse diagram of $\text{Poset}(4, 2)$ with an ideal in bold. ....... 53
4.2 The Hasse diagram of $\text{NPoset}(3, 3)$ with an ideal in bold. The ideal is the necklaces with sum at least 6. The set of rotations of the ideal corresponds to the strings in $T(3, 3)$ with sum at least 6. ............... 54

5.1 Illustrating the de Bruijn graph corresponding to the universal cycles of the 4 necklace equivalence classes $\text{Neck}(000111)$, $\text{Neck}(001011)$, $\text{Neck}(001101)$ and $\text{Neck}(010101)$ that make up $B_3(6)$. ............... 79
List of Tables

3.1 The cyclic order of $B(6)$ starting from 000000 induced by the function $f$. The rows break down the order based on when $f$ applies a complemented reversal. ................................................................. 32
3.2 The cyclic order of $T(4, 3)$ starting from 1111 induced by the function $f_k$. The underlined strings are of the form $\alpha = a_1a_2\cdots a_n$ such that $\alpha$ is not a necklace but $a_1a_2\cdots a_{n-1}(a_n + 1)$ is a necklace. ..................... 43
Chapter 1

Introduction

In the book Magical Mathematics [22], Diaconis and Graham introduced a card trick called de Bruijn card trick which can accurately predict successive cards in a seemingly random deck by only knowing the color of cards. The card trick starts with a deck of 32 cards that includes all four suits of cards with ranks A, 2, 3, 4, 5, 6, 7 and 8. The magician first gives the deck to the audience and allows them to cut which preserves the cyclic order. Five volunteers are selected to pick a card on top of the deck hiding it from the magician. The volunteers are then asked to raise their hands if they have a red card. The magician subsequently accurately predicts all the volunteers’ card, as well as the remaining cards in the deck one by one.

How does the magician pull off this feat? The secret of the de Bruijn card trick refers to the ordering of the cards at the beginning. To begin with, each card is encoded into a length 5 binary string using its suit and rank. The suit of a card is encoded into a length 2 binary string as follows:

- ♣: 00
- ♠: 01
- ♦: 10
- ♥: 11

The black suits have the first bit equal to 0 and the red suits have the first bit equal to 1. Meanwhile, the rank of a card is encoded into a length 3 binary string as follows:

- A: 000 5: 100
- 2: 001 6: 101
- 3: 010 7: 110
- 4: 011 8: 111

Thus, each card corresponds to a unique length 5 binary string by concatenating the binary strings corresponding to its suit and its rank. For example, the card ♣2 has
suit ♠: 00 and rank 2: 001, thus the card corresponds to the string 00001. Then, we arrange the deck by following the ordering of the 32 length 5 binary strings below:

- 00001, 00010, 00101, 10101, 01011, 10111, 01110,
- 11101, 11011, 10110, 01100, 11000, 10001, 00011, 00111,
- 01111, 11111, 11110, 11100, 11001, 00110, 01101,
- 11010, 10100, 01001, 10010, 01000, 01000, 10000.

The cards are thus arranged as follows (the card trick works for any rotation of this ordering):

- ♣2, ♣3, ♣6, ♣3, ◆6, ♣4, ◆8, ♣7, ◆4, ◆7, ♣5, ◆A, ◆2, ♣4, ♣8,
- ♣8, ◆8, ◆7, ◆5, ◆2, ◆4, ♣7, ◆6, ◆3, ◆5, ♣2, ◆3, ♣5, ♣A, ◆A, ♣A.

This ordering of length 5 binary strings has some unique properties. First when we concatenate the first bit of each length 5 binary string together, we obtain a sequence of length 32 which contains each length 5 binary string as a substring exactly once when considered circularly. The i-th length 5 binary string in the listing above is the length 5 substring that starts at index i of the sequence when considered circularly, where 0 < i ≤ 32. Such a sequence is an example of a de Bruijn sequence, or more generally a universal cycle. The de Bruijn sequence that corresponds to the 32 length 5 binary strings above is

- 0000101011011000111110011010010.

This de Bruijn sequence has a successor rule to find the next bit b in the sequence. The next bit in the de Bruijn sequence is determined by the following function f, where \( b_1 b_2 b_3 b_4 b_5 \) denotes the length 5 binary string before the bit b:

- \( f(b_1 b_2 b_3 b_4 b_5) = (b_1 + b_4) \mod 2 \) for all strings except 00000 and 00001, which map to 00001 and 00000 respectively.

As an example of the application of f, the next bit after the substring 01110 in the de Bruijn sequence is \( f(01110) = (0 + 1) \mod 2 = 1 \). This function is an example of feedback functions which are commonly used in feedback shift registers to generate de Bruijn sequences for length n binary strings.
Recall that the colors of cards refer to the first bit of their corresponding length 5 binary strings. Thus, the colors of the first 5 cards correspond to the first 5 bits of the de Bruijn sequence. Therefore, when the colors of the first 5 cards are revealed, the magician translates the colors into a length 5 binary string, which is the length 5 binary string that corresponds to the first card. The magician then repeatedly applies the function \( f \) to compute the rest of the cards. As an example, assume the first 5 cards are ♠7, ♥6, ♥4, ♦7 and ♠5 respectively. When the volunteers are asked to raise their hands if they have cards with red suits, the second, the third and the forth volunteers will raise their hands. The magician then knows the length 5 binary string corresponds to the first card is 01110, thus the first card is ♠7. The next bit in the sequence is \( f(01110) = (1 + 0) \mod 2 = 1 \). Thus, the length 5 binary string corresponds to the next card is 11101. Therefore, the next card is ♥6. Magic!

### 1.1 Universal cycles and de Bruijn sequences

A universal cycle for a set \( S \) is a cyclic sequence of length \( |S| \) whose length \( n \) substrings encode \( |S| \) distinct instances in \( S \). Thus, the maximum length of a universal cycle is \( k^n \) where \( k \) is the number of distinct symbols in the universal cycle. As an example, the following sequence is a universal cycle for the subsets of size \( n = 2 \) over the character set \( \{1, 2, 3, 4, 5\} \):

1234524135.

The sequence has a total length of 10. Each length 2 substring of the sequence encodes a unique size 2 subset when considered circularly as follows:

\[
\{1, 2\}, \{2, 3\}, \{3, 4\}, \{4, 5\}, \{2, 5\}, \{2, 4\}, \{1, 4\}, \{1, 3\}, \{3, 5\}, \{1, 5\}.
\]

The term universal cycle was first introduced by Chung, Diaconis and Graham in 1992 [17]. Universal cycles have been extensively studied for \( k \)-ary strings of length \( n \) under the name de Bruijn sequence. As an example, the sequence

112132233
is a de Bruijn sequence for 3-ary strings of length 2 since it contains all nine 3-ary strings of length 2 as a substring exactly once when considered circularly:

\[ 11, 12, 21, 13, 32, 22, 23, 33, 31. \]

Universal cycles and de Bruijn sequences can also be represented circularly. A circular representation of a de Bruijn sequence for length 4 binary strings is as follows.

In this thesis, we use the term universal cycle when \( S \) refers to an arbitrary set. We use the term de Bruijn sequence when \( S \) refers to the set of \( k \)-ary strings of length \( n \).

The study of universal cycles has a long history which was first motivated by applications of memory wheels. The first known memory wheel originated in India at around 1000 A.D. by medieval Indian poets and musicians [82, 97]. The memory wheel \( yamātārājabhānasalagām \) contains all possible triplets of short and long syllables as a substring. Martin first proved the existence of de Bruijn sequences in 1934 [69]. Later in 1951, van Aardenne-Ehrenfest and de Bruijn showed that the number of de Bruijn sequences is equal to \( k!^{k-1}/k^n \) [99]. The formula for \( k = 2 \) was derived by de Bruijn in 1946 [19] and thirty years later he learned that the same formula had also been derived by Flye Sainte-Marie in 1894 [87] thanks to historical research by Richard Stanley [20]. The idea was then generalized to universal cycles which considered sets other than \( k \)-ary strings of length \( n \). Universal cycles have since become classic objects of study in Combinatorics and theoretical computer science [19, 20, 49]. The studies of these sequences are covered in influential textbooks such as *Concrete Mathematics: A Foundation for Computer Science* by Graham, Knuth, and Patashnik [43], and *The Art of Computer Programming, Volume 4A, Combinatorial Algorithms* by Knuth [57].
In 2004, the workshop *Generalizations of de Bruijn Cycles and Gray Codes* was held in Banff, Canada to advance the interest and knowledge in universal cycles. The unsolved problems in the meeting were published as a special issue in the journal *Discrete Mathematics* [52]. The *Joint Mathematics Meetings* in 2014 held in Baltimore also dedicated a special session *AMS Special Session on De Bruijn Sequences and Their Generalizations* to discuss the latest results related to universal cycles. In these contexts, the term “generalizations” refers to universal cycles.

1.2 Thesis statement

Recall the de Bruijn card trick that illustrates one of the many applications of universal cycles (more applications are discussed in Chapter 2.5). There are two difficulties that restrict the card trick to be extended to include more cards. Firstly, the feedback function $f$ is not applicable to all values of $n$. Linear feedback functions have not been found for some large value of $n$. Thus the magician may need to memorize the whole de Bruijn sequence to perform the card trick when no successor rule is available. Secondly, the length of a de Bruijn sequence is restricted to be equal to $2^n$ for length $n$ binary strings, and $k^n$ for $k$-ary strings of length $n$. However, the total number of cards may not be equal to these values. In fact, the total number of cards in a deck of playing card is 52. These restrictions not just limit the card trick, but also other applications of universal cycles.

This thesis provides some answers to the following research questions:

1. Is there a simple successor rule to generate the next symbol in a de Bruijn sequence for all values of $n$ and $k$?

2. Do universal cycles exist for some previously unstudied, yet interesting subsets of length $n$ binary strings and $k$-ary strings?

3. If so, are there simple and efficient constructions to generate the universal cycles?

Our research leads to the following new results in this area:
1. Novel shift rules to construct de Bruijn sequences for length $n$ binary strings and $k$-ary strings. The shift rules provide simple and efficient successor rules to find the next symbol in the de Bruijn sequences.

2. Generalizations of the Fredricksen-Kessler-Maiorana (FKM) construction and greedy construction to generate universal cycles for a broad class of $k$-ary strings.

3. The first known efficient construction to generate universal cycles for length $n$ binary strings where the number of 1s (weight) range from $c$ to $d$ given $0 \leq c < d \leq n$ (also known as universal cycles for weight-range binary strings).

Some of these results have been published and can be found in [91, 92].

1.3 Overview of thesis

This thesis is organized as follows. Chapter 2 provides a literature review of universal cycles and discusses some applications. Chapter 3 introduces novel shift rules that construct de Bruijn sequences for length $n$ binary strings and $k$-ary strings. Chapter 4 discusses generalizations of the FKM and greedy constructions to generate universal cycles for a broad class of $k$-ary strings. Chapter 5 introduces an efficient construction to generate universal cycles for weight-range binary strings. We conclude this thesis and introduce avenues for future research in Chapter 6.
Chapter 2

Literature review and relevant results

This chapter provides a literature review on universal cycles. We discuss existence results and also review some common constructions of de Bruijn sequences. We also introduce a construction that generates universal cycle for a subset of length \( n \) binary strings. We then introduce some applications of universal cycles.

2.1 Notations and definitions

We first introduce some basic notations and definitions used in this thesis. A universal cycle has an efficient algorithm if each successive symbol of the sequence can be generated in \( O(1) \)-amortized time (also known as constant amortized time (CAT)) while using a polynomial amount of space with respect to \( n \). A necklace is the lexicographically smallest string in an equivalence class of strings under rotation. The aperiodic prefix of a string \( \alpha \), denoted as \( \text{ap}(\alpha) \), is its shortest prefix whose repeated concatenation yields \( \alpha \). That is, the aperiodic prefix of \( \alpha = a_1a_2 \cdots a_n \) is the shortest prefix \( \text{ap}(\alpha) = a_1a_2 \cdots a_p \) such that \((\text{ap}(\alpha))^\frac{n}{p} = \alpha\), where exponentiation denotes repeated concatenation and \( \frac{n}{p} \) is an integer. For example, when \( \alpha = 001001001 \), \( \text{ap}(\alpha) = 001 \) with \( \frac{n}{p} = 3 \) an integer. A string is a prenecklace if it is the prefix of some necklace. A string \( \alpha \) is aperiodic if \( \text{ap}(\alpha) = \alpha \), otherwise it is periodic. Aperiodic necklaces are also known as Lyndon words. Denote the set of all necklaces that belong to \( S \) by \( \mathbf{N}(S) \). Let \( \mathbf{B}(n) \), \( \mathbf{P}(n) \), \( \mathbf{N}(n) \) and \( \mathbf{L}(n) \) denote the set of length \( n \) binary strings, length \( n \) binary prenecklaces, length \( n \) binary necklaces and length \( n \) binary Lyndon words respectively. We also denote \( \text{Neck}(\beta) \) as the set of strings rotationally equivalent to \( \beta \). For example:
• \( B(4) = \{0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, 1000, 1001, 1010, 1011, 1100, 1101, 1110, 1111\} \),

• \( P(4) = \{0000, 0001, 0010, 0011, 0101, 0110, 0111, 1111\} \),

• \( N(4) = \{0000, 0001, 0011, 0101, 0111, 1111\} \),

• \( L(4) = \{0001, 0011, 0111\} \),

• \( \text{Neck}(0001) = \{0001, 0010, 0100, 1000\} \).

Observe that the prenecklaces 0010 and 0110 are prefixes of the necklaces 00101 and 011011 respectively so they are in \( P(4) \). The cardinality of \( N(n) \) and \( L(n) \) are given by the following formulas [40], where \( \phi \) and \( \mu \) are the Euler’s totient function and the Möbius function respectively:

\[
|N(n)| = \frac{1}{n} \sum_{d|n} \phi(d) 2^{n/d} \quad \text{and} \quad |L(n)| = \frac{1}{n} \sum_{d|n} \mu(d) 2^{n/d}.
\]

We also extend the notations to general alphabets. Let \( T(n, k) \), \( N(n, k) \) and \( L(n, k) \) denote the sets of \( k \)-ary strings of length \( n \), \( k \)-ary necklaces of length \( n \) and \( k \)-ary Lyndon words of length \( n \) respectively. For example:

• \( T(2, 3) = \{11, 12, 13, 21, 22, 23, 31, 32, 33\} \),

• \( N(2, 3) = \{11, 12, 13, 22, 23, 33\} \),

• \( L(2, 3) = \{12, 13, 23\} \).

The cardinality of \( N(n, k) \) and \( L(n, k) \) are given by the following formulas [40]:

\[
|N(n, k)| = \frac{1}{n} \sum_{d|n} \phi(d) k^{n/d} \quad \text{and} \quad |L(n, k)| = \frac{1}{n} \sum_{d|n} \mu(d) k^{n/d}.
\]

**Membership tester for necklaces**

The efficiency of many universal cycle constructions depends on the efficiency of a membership tester of necklaces. Most of these constructions embed an \( O(n) \) algorithm
Algorithm 1 An $O(n)$-time membership tester of necklaces.

1: function IsNecklace($a_1a_2\cdots a_n$)
   2:   $p \leftarrow 1$
   3:   for $i$ from 2 to $n$ do
   4:      if $a_{i-p} < a_i$ then $p \leftarrow i$
   5:      else if $a_{i-p} > a_i$ then return 0
   6:      if $n \mod p = 0$ then return $p$
   7:      else return 0

To test whether a string is a necklace. The algorithm can be easily derived from the works of Booth [9] and Duval [26].

Given an arbitrary $k$-ary string $\alpha = a_1a_2\cdots a_n$, the algorithm returns the length of the longest aperiodic prefix of $\alpha$ if $\alpha$ is a necklace, otherwise the algorithm returns 0. The algorithm scans each character of $\alpha$ and maintains a variable $p$ which stores the length of the longest aperiodic prefix of $\alpha$ that is a prenecklace. The algorithm initializes $p = 1$, and updates $p = i$ when $a_i > a_{i-p}$. If $a_i < a_{i-p}$, then $a_1a_2\cdots a_{p\lfloor \frac{i}{p} \rfloor}$ is lexicographically larger than $a_{p\lfloor \frac{i}{p} \rfloor+1}a_{p\lfloor \frac{i}{p} \rfloor+2}\cdots a_i$ and hence $\alpha$ is not a necklace. Thus, in this case, the algorithm returns 0. Otherwise, the algorithm continues to scan $\alpha$ and maintains the variable $p$ until reaching $a_n$. The string $\alpha$ is a necklace if $n \mod p = 0$. The algorithm thus returns the value of $p$ if $n \mod p = 0$, otherwise returns 0. Pseudocode of this algorithm is given in Algorithm 1.

2.2 De Bruijn graphs and existence results of universal cycles

Martin proved that de Bruijn sequences always exist. However, when we consider an arbitrary set $S$, universal cycles do not always exist for the set $S$. As an example, universal cycle does not exist for permutations over the character set $\{1, 2, \ldots, 8\}$ when represented as strings in one-line notation. To demonstrate the non-existence result, observe that the universal cycle must contain the substring 87654321. Since 7654321_ is also a permutation of the character set $\{1, 2, \ldots, 8\}$, the missing symbol at the po-
sition labeled “..” must be 8. By following the same argument, the next length 8 sub-
strings in the sequence after 87654321 are 76543218, 65432187, 54321876, 43218765,
32187654, 21876543, 18765432 and 87654321, a repetition. Thus the sequence cannot
be extended to a universal cycle. In general, universal cycle for permutations over
the character set \{1, 2, ..., n\} does not exist when \(n > 2\).

The existence of universal cycles of a combinatorial object is related to the Eule-
rian property of its underlying graph known as de Bruijn graph. The de Bruijn graph
\(G(S)\) for a set of length \(n\) strings \(S\) is a directed graph whose vertex set consists of
the length \(n-1\) prefixes and suffixes of the strings in \(S\). For each string \(b_1b_2 \cdots b_n \in S\)
there is an edge labeled \(b_n\) that is directed from the prefix \(b_1b_2 \cdots b_{n-1}\) to the suf-
fix \(b_2b_3 \cdots b_n\). Thus, the graph has \(|S|\) edges. As an example, the de Bruijn graph
\(G(B(4))\) is illustrated in Figure 2.1.

A (directed) cycle in a directed graph \(G = (V, E)\) is a sequence \(v_1, \ldots, v_j, v_1\)
where \(v_i \in V\) and \((v_i, v_{i+1}) \in E\). A directed graph is said to be Eulerian if it contains
a directed cycle that includes each edge exactly once. It is well known that \(S\) admits
a universal cycle if and only if \(G(S)\) is Eulerian. If \(G(S)\) contains an Euler cycle, then
a universal cycle is produced by traversing an Euler cycle and outputting the edge
labels. A directed graph is said to be balanced if the in-degree of each vertex is the
same as its out-degree. It is weakly connected if the underlying undirected graph is
connected. The following result is well-known, and appears in many references such
as [81]:

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure21.jpg}
\caption{The de Bruijn graph \(G(B(4))\).}
\end{figure}
Theorem 2.2.1  A directed graph is Eulerian if and only if it is balanced and weakly connected.

Universal cycles were shown to exist for the following combinatorial objects by studying the Eulerian property of their underlying de Bruijn graphs:

- $k$-ary strings of length $n$ [19, 20, 49],
- $k$-ary strings of length $n$ with forbidden substrings [73, 75],
- Length $n$ binary strings with weight in the range $c, c + 1, \ldots, d$ where $0 \leq c < d \leq n$ [7, 12, 86],
- $k$-permutations over the character set $\{1, 2, \ldots, n\}$ when $k < n$ [51],
- Permutations over the character set $\{1, 2, \ldots, n\}$ when represented under order-isomorphism or shorthand representation [17, 57],
- Labeled graphs [11], and
- Passwords [60].

In theory, a specific universal cycle can be constructed by using the Euler cycle characterization and a classic Euler cycle algorithm. In practice, there are two issues with this approach. Firstly, the underlying de Bruijn graph is often exponentially large with respect to $n$. For example, the memory required to store a de Bruijn graph for $B(n)$ is $\Omega(2^n)$. Secondly, the Euler cycle algorithms can be computationally intensive and inconvenient to work with. In particular, Fleury’s algorithm [29] relies on repeated connectivity tests, and Hierholzer’s algorithm [46] creates subsequences that must be cut and pasted together to create the final universal cycle. The goal in this thesis is therefore to discover efficient constructions for universal cycles for a variety of combinatorial objects.
2.3 De Bruijn sequence constructions

This section provides a literature review on important constructions of de Bruijn sequences. Fredricksen provided an excellent survey in [34] that summarizes most commonly known methods to construct these sequences. As mentioned earlier, the number of unique de Bruijn sequences is equal to $k!^{kn-1}/k^n$ [19, 99]; however, only a very small number are known to have simple and efficient constructions. This fact is perhaps best captured by a quote in Fredricksen’s survey:

When a mathematician on the street is presented with the problem of generating a full cycle (de Bruijn sequence), one of the three things happens: he gives up, or produces a sequence based on a primitive polynomial, or produces the prefer-one sequence. Only rarely a new algorithm is proposed.

In the rest of this section, we introduce some current approaches to construct de Bruijn sequences, including

- Greedy algorithms including “prefer-0” by Martin [69] in 1934, and “prefer opposite” by Alhakim [2] in 2010,

- Feedback shift register algorithms using a primitive polynomial over the binary finite field by Golomb [42] (original draft from 1967),

- A recursive approach by Lempel [61] in 1970,

- Necklace and Lyndon word concatenation using lexicographic ordering by Fredricksen and Maiorana [36] in 1978 commonly known as the Fredricksen-Kessler-Maiorana construction, or simply the FKM construction,


The readers should take note that some of these constructions only generate de Bruijn sequences over a binary alphabet but not a general alphabet. Also, many of them are not efficient.
2.3.1 Greedy construction

Martin [69] proposed a simple algorithm to construct a de Bruijn sequence over a general alphabet. The algorithm starts with the sequence $k^{n-1}$ and applies a simple rule which can be summarized as follows:

Repeatedly append the smallest symbol in $\{1, 2, \ldots, k\}$ so that the substrings of length $n$ in the resulting sequence are distinct and in $T(n, k)$.

The greedy algorithm always terminates with a length $k^n + n - 1$ sequence that has suffix $k^n$. A de Bruijn sequence is thus obtained by removing the initial prefix $k^{n-1}$. For example, Martin’s sequence for 3-ary strings of length 2 is shown below:

\[
112132233
\]

has substrings 11, 12, 21, 13, 32, 22, 23, 33, 31.

The first step of Martin’s algorithm is crucial to its correctness. For example, if we start the algorithm with $1^{n-1}$, then the algorithm terminates with the sequence 112131 for $n = 2$ and $k = 3$, which is not a de Bruijn sequence.

Martin’s sequence is the lexicographically smallest de Bruijn sequence. Knuth refers to Martin’s sequence as the grand-daddy of all de Bruijn sequences [57]. Similar algorithms have been proposed with different symbol insertion criteria. Some of these well known algorithms are the prefer-one, prefer-same [34] and prefer-opposite algorithms [2] for binary alphabets, and prefer-higher algorithm for general alphabets [3, 30]. These algorithms are categorized as greedy constructions of de Bruijn sequences.

The greedy constructions take $O(kn)$ time to generate each symbol for the worst case but require $\Omega(k^n)$ space to memorize the appearance of each $k$-ary strings of length $n$ in the sequence. In Chapter 4, we will extend Martin’s construction to generate universal cycles for a broad class of $k$-ary strings.

2.3.2 Feedback shift register sequence

Feedback shift registers have been heavily studied [6, 50, 63, 67, 71, 78] and are used extensively to generate de Bruijn sequences over a binary alphabet. Let $\mathbb{F}_2$ denote the
binary finite field and $\mathbb{F}_2^n$ denote the $n$-dimensional vector space over $\mathbb{F}_2$ consisting of the $n$-tuples of elements of $\mathbb{F}_2$. A boolean function is a mapping from $\mathbb{F}_2^n$ to $\mathbb{F}_2$. Then a feedback shift register is defined as a mapping from $\mathbb{F}_2^n$ to $\mathbb{F}_2^n$ such that

$$(b_1, b_2, \ldots, b_n) \rightarrow (b_2, b_3, \ldots, b_n, q(b_1, b_2, \ldots, b_n)),$$

where $q$ is a boolean function which is known as feedback function.

A feedback shift register shifts its contents into adjacent positions and fills the vacant position with a new value generated by a feedback function $q$. The initially loaded content in the register is called an initial state. At every clock pulse, the state in the register is updated from $b_1, b_2, \ldots, b_n$ to $b_2, b_3, \ldots, b_n, q(b_1, b_2, \ldots, b_n)$. The register terminates when it returns to the initial state. As an example, the sequence 0010111

is generated by a feedback shift register with the feedback function $b_{3+i} = (1 + b_i + b_{i+3-1} \cdot b_{i+3-2} + b_{i+3-1}) \mod 2$ and initial state 001. The sequence contains all binary strings of length 3 as a substring exactly once except 000.

A feedback shift register is a linear feedback shift register (LFSR) if its feedback function is linear, otherwise it is a nonlinear feedback shift register (NLFSR). An $m$-sequence is a sequence of length $2^n - 1$ which contains each length $n$ binary string as a substring except $0^n$. An $m$-sequence is the maximum-length sequence a linear feedback shift register can generate. A linear feedback shift register generates an $m$-sequence if and only if its feedback function is primitive [42].

LFSRs have been well-studied and have a vast number of applications; however, their simplicity and linearity leads to a relatively easy cryptanalysis. NLFSRs have thus attracted interests from applications which require higher security and complexity. For example, NLFSRs have been used in stream ciphers of RFID and smart card applications. Stream ciphers that involve NLFSRs include Achterbahn [38], Grain [45, 70], KeeLoq [8], Trivium [13], VEST [41], and the stream cipher in [39]. An open cryptographic competition eSTREAM was held yearly from 2004 to 2008 to
call for submissions of new stream ciphers suitable for widespread adoption. Their results have been compiled in [80] and many of them involve heavy use of NLFSRs.

There are $2^{2n-1}/2^n$ feedback functions to generate de Bruijn sequences over a binary alphabet. Among these functions, $\phi(2^n - 1)/n$ of them are linear where $\phi$ denotes the Euler’s totient function. Algorithms to generate LFSRs include an exhaustive search of primitive feedback functions [101] and constructions based on known primitive feedback functions [93, 103]. These methods are, however, complex and computational infeasible when $n$ is large. A partial list of known primitive feedback functions for degree up to 4423 can be found in [100, 102, 103]. For NLFSRs, construction of maximal period NLFSRs when $n$ is large remains an open problem. Most current available solutions construct an NLFSR using existing LFSRs [24] which are only applicable when $n$ is small [25, 64]. A list of known feedback functions for NLFSRs with $n$ up to 25 and 27 can be found in [23, 79].

A LFSR generates a de Bruijn sequence in $O(n)$ time per bit using $O(n)$ space, while the complexity for an NLFSR to generate a de Bruijn sequence depends on the complexity of the feedback function. The feedback shift register approach is the only classic construction that provides a successor rule to generate the next bit of an arbitrary length $n$ binary string in a de Bruijn sequence. However, it uses different feedback functions for different values of $n$. Numerous algorithms have been developed based on feedback shift registers to construct universal cycles for sets other than length $n$ binary strings [37, 53, 56, 62, 68, 76, 77, 95]. In Chapter 3, we will introduce a novel shift rule to construct de Bruijn sequence over a binary alphabet which also has a nice successor rule. The shift rule is applicable to all values of $n$.

2.3.3 Lempel’s recursive construction

Lempel [61] introduced a recursive construction to generate de Bruijn sequences over a binary alphabet by studying the homomorphism of the de Bruijn graphs $G(B(n))$ and $G(B(n-1))$. A binary sequence is complement-free if it does not contain two length $n$ substrings which are complement to each other. A complement-free de Bruijn sequence for $B(n)$ is a complement-free sequence of length $2^{n-1}$ where each length $n$
substring is distinct. As an example, the sequence

0000011000101001

is a complement-free de Bruijn sequence for $B(5)$ since its 16 substrings of length 5 are distinct, and no pair of length 5 binary string are complement to each other:

00000, 00001, 00011, 00110, 01100, 11000, 10001, 00010,
00101, 01010, 10100, 01001, 10010, 00100, 01000, 10000.

Since a complement-free de Bruijn sequence $C$ contains no two length $n$ substrings that are complement to each other, we can produce another complement-free de Bruijn sequence $\overline{C}$ by complementing each bit of the sequence. As an example, by complementing each bit of the above complement-free de Bruijn sequence, we obtain another complement-free de Bruijn sequence for $B(5)$:

1111100111010110.

A pair of complement-free de Bruijn sequences $C$ and $\overline{C}$ for $B(n - 1)$ can be joined together to form a de Bruijn sequence for $B(n)$. As an example, the sequences

0000011000101001 and 111100111010110

are a pair of complement-free de Bruijn sequences for $B(4)$ which they are complement to each other. By rotating both sequences to start with the prefix 1100 and joining them together, we obtain a de Bruijn sequence for $B(5)$:

1100111010110111000100100000.

In Chapter 5, we will discuss more about this cycle joining method.

A pair of complement-free de Bruijn sequences for $B(n)$ can be obtained by applying $D$-morphism on a de Bruijn sequence for $B(n - 1)$ [61].

**Lemma 2.3.1 (Lempel’s $D$-morphism)** Let $C = b_1b_2\cdots b_m$ be a complement-free binary sequence of length $m$ corresponds to successive edge labels of a simple cycle of length $m$ in the de Bruijn graph for $B(n)$. Define a mapping $D$: $\sum^m \to \sum^m$ for $i \geq 2$ with $\sum = \{0, 1\}$:
\[ D(C) = (b_1 \oplus b_2, b_2 \oplus b_3, \ldots, b_{m-2} \oplus b_{m-1}, b_{m-1} \oplus b_m). \]

The binary sequence \( D(C) \) corresponds to successive edge labels of a length \( m \) cycle in the de Bruijn graph \( G(B(n-1)) \).

Let \( b_1 b_2 \cdots b_{2n-1} \) be a de Bruijn sequence for \( B(n-1) \), and \( C = c_1 c_2 \cdots c_{2n-1} \) be a sequence obtained by the operations below:

1. Set \( c_1 = 0 \),
2. For any subsequent bits \( c_i \) in \( C \) with \( i > 1 \), if \( b_i = 0 \), update \( c_i = c_{i-1} \). Otherwise update \( c_i = \overline{c_{i-1}} \).

**Lemma 2.3.2** \( C \) is a complement-free de Bruijn sequence for \( B(n) \) when \( n > 2 \).

As an example of the application of Lemma 2.3.2, consider a de Bruijn sequence for \( B(5) \):

\[ 0000100011001010011101011011111. \]

By applying Lemma 2.3.2, we obtain a complement-free de Bruijn sequence for \( B(6) \) as follows:

\[ C = 0000011110110011100101101010. \]

We can thus obtain another complement-free de Bruijn sequence \( \overline{C} \) by complementing each bit of \( C \), thus

\[ \overline{C} = 1111100001000110010110010101. \]

By joining \( C \) and \( \overline{C} \) together, we create a de Bruijn sequence for \( B(6) \):

\[ 000011110110011101010000010001100101100100101011111. \]

Several algorithms have been developed based on Lempel’s \( D \)-morphism to construct de Bruijn sequences over a binary alphabet \([4, 15, 16, 27]\). The fastest known algorithm in this category generates a de Bruijn sequence in \( O(1) \)-amortized time per bit while using \( \Omega(2^n) \) space \([4, 16]\).
2.3.4 The FKM construction

Fredricksen and Kessler [32, 33, 35] (for \( k = 2 \)) and later Fredricksen and Maiorana [36] (for \( k \geq 2 \)) discovered a very simple and efficient construction to generate de Bruijn sequence over a general alphabet. Recall that \( T(n, k) \) is the set of \( k \)-ary strings of length \( n \). Their construction can be summarized as follows:

*Concatenate the aperiodic prefixes of the necklaces in \( T(n, k) \) in lexicographic order.*

For example, the necklaces in \( T(2, 3) \) are 11, 12, 13, 23, 33 and thus:

\[
1 \cdot 12 \cdot 13 \cdot 2 \cdot 23 \cdot 3 \quad \text{is a de Bruijn sequence for} \quad T(2, 3).
\]

This construction is often known as the *Fredricksen-Kessler-Maiorana construction*, or simply the *FKM construction*. The FKM construction produces the lexicographically smallest de Bruijn sequence. Furthermore, the de Bruijn sequence is identical to the one created greedily by Martin’s sequence when we consider the set \( T(n, k) \).

Ruskey, Savage and Wang [83] provided an algorithm to generate necklaces in \( T(n, k) \) in lexicographic ordering using \( O(1) \)-amortized time and \( O(n) \) space. Such an algorithm can be used to generate the necklaces in the FKM construction, and is now commonly known as the *FKM algorithm*. The FKM algorithm generates the de Bruijn sequence in \( O(1) \)-amortized time per symbol using \( O(n) \) space. Recently, Kociumaka, Radoszewski and Rytter introduced a ranking method that computes the position of an arbitrary binary necklace or binary Lyndon word in lexicographic ordering in polynomial time [58]. This ranking method has application on decoding the de Bruijn sequence constructed by the FKM construction over a binary alphabet.

The FKM construction is one of the most commonly used de Bruijn sequence constructions because it is accompanied with an efficient algorithm. In Chapter 4, we will generalize the FKM construction to generate universal cycles for a broad class of \( k \)-ary strings.
2.3.5 Doubly recursive approach

Mitchell, Etzion and Paterson [72] introduced a doubly recursive approach to construct decodable de Bruijn sequences over a binary alphabet. These sequences have an advantage that the position of any arbitrary length $n$ binary string in the de Bruijn sequence can be determined very efficiently.

A punctured de Bruijn sequence is a cyclic sequence of length $2^n - 1$ which contains every length $n$ binary strings as a substring except $0^n$ when considered circularly (same as an $m$-sequence as defined in Chapter 2.3.2). Similarly, a doubly punctured de Bruijn sequence is a cyclic sequence of length $2^n - 2$ which contains every length $n$ binary strings as a substring except $0^n$ and $1^n$ when considered circularly. The algorithm considers the interleaving of punctured de Bruijn sequences and doubly punctured de Bruijn sequences to produce a de Bruijn sequence in a higher dimension. Consider a doubly punctured de Bruijn sequence $S_A$ for $n = 3$, and $S_B$ which is a sequence formed by replacing the substring $0^{n-1}$ in $S_A$ with $0^{n+1}$, and the substring $1^{n-1}$ with $1^{n+1}$. For example:

$$S_A = 001011, S_B = 0000101111.$$ 

Mitchell, Etzion and Paterson proved that the interleaving of $\frac{|S_A|}{2}$ copies of $S_B$ and $\frac{|S_A|}{2} + 2$ copies of $S_A$, denoted by $I(S_B^{\frac{|S_A|}{2} + 2}, S_A^{\frac{|S_A|}{2}})$, creates a sequence with no duplicate length $2n$ substring when considered circularly. For example,

$$I(S_B^3, S_A^5) = 000001001101101101011000001100111101001000101100111111110111.$$ 

The missing length 6 binary strings in the sequence are 000000, 111111, 010101 and 101010. By inserting the underlined extra bits in the below sequence, we obtain a de Bruijn sequence for $B(6)$:

$$00000010011011010101100100001100111110100100010110001111111.$$ 

Several similar approaches have been discussed in [72]. Each successive bit of these sequences can be generated in $O(1)$-amortized time per bit when a doubly punctured de Bruijn sequence or a punctured de Bruijn sequence is known in advance.
2.4 Other universal cycle construction

Most of the constructions discussed in previous section consider constructions of de Bruijn sequences, however, practical applications may demand constructions of universal cycles with shorter length, or with constraints over the elements of the set. In this section, we consider universal cycles for fixed-weight binary strings, which are universal cycles for subsets of $\mathbf{B}(n)$ that contain binary strings with the same number of 1s.

Let $\mathbf{B}_w(n)$ denote the set of binary strings with number of 1s (weight) equal to $w$. For example, $\mathbf{B}_2(4) = \{0011, 0101, 0110, 1010, 1100, 1001\}$. Universal cycle for $\mathbf{B}_w(n)$ does not exist under standard one-line notation (the non-existence proof is similar to the proof in Chapter 2.2 for universal cycle for permutation over $n$ symbols). However, it exists using other notations. Ruskey, Holroyd and Williams [47] introduced the shorthand encoding to construct a universal cycle for $\mathbf{B}_w(n)$. Observe that the last bit of each string in $\mathbf{B}_w(n)$ is redundant. That is, each string in $\mathbf{B}_w(n)$ can be determined by its first $n - 1$ bits. Thus a universal cycle for $\mathbf{B}_w(n)$ is equivalent to a universal cycle for the set of length $n$ binary strings with weight $w$ and $w + 1$ (also known as duel-weight universal cycle). For example, the sequence

\[
0011101011
\]

is a shorthand universal cycle for $\mathbf{B}_3(5)$. Each length 5 binary string with weight 3 is obtained by appending the missing bit (the underlined bit) to each length 4 substring in the shorthand universal cycle:

\[
0011\underline{1}, \ 0110, \ 1100, \ 1101, \ 1010, \ 0101, \ 1011, \ 0110, \ 1100, \ 1001.
\]

2.4.1 The cool-daddy construction

Ruskey, Sawada and Williams [85] proposed using cool-lex ordering to construct a universal cycle for fixed-weight binary strings. Cool-lex order is a shift Gray code where successive strings differ by a single shift. A shift of the $j^{th}$ position and the $i^{th}$
Figure 2.2: Concatenation of aperiodic prefixes of length 8 binary necklaces with weight 4 in reverse cool-lex ordering. The resulting sequence is a shorthand universal cycle for length 8 binary strings with weight 4.

We denote such an operation as \( \text{shift}_\alpha(i,j) \).

Let \( \alpha = 0^s1^t\gamma \) be a necklace of weight \( w \) with \( s, t > 0 \) and \( \gamma \) is the empty string or a string begins with 0. Let \( \text{next}(\alpha) \) denote the next necklace in cool-lex ordering after \( \alpha \), and \( N_w(n) \) be the set of binary necklaces of length \( n \) with weight \( w \). The next necklace in cool-lex ordering is given by the formula:

\[
\text{next}(\alpha) = \begin{cases} 
\text{shift}_\alpha(s + t, i + 1) & \text{if } \gamma = \epsilon; \\
\text{shift}_\alpha(s + t + 1, i) & \text{if } a_{s+t+2} = 0 \text{ or } \beta \notin N_w(n); \\
\text{shift}_\alpha(s + t + 2, i + 1) & \text{otherwise},
\end{cases}
\]

where \( \beta = \text{shift}_\alpha(s+t+2, s+t+1) \), and \( i \) is the minimum value such that \( 0^t1^{t-i}0^{s-i}\gamma \) is a necklace of weight \( w \).

The \textit{cool-daddy universal cycle} is constructed in a similar way as the FKM construction but orders the aperiodic prefixes of necklaces with weight \( w \) in reverse cool-lex ordering instead of lexicographic ordering. The construction can be summarized by the following formula:

\[
\text{Cool}_w(n) = \text{ap}(\ell_m) \cdot \text{ap}(\ell_{m-1}) \cdots \text{ap}(\ell_1),
\]
where $\ell_1, \ell_2, \ldots, \ell_m$ are necklaces with weight $w$ ordered in cool-lex ordering. As an example, Figure 2.2 demonstrates a shorthand universal cycle for length 8 binary strings with weight 4 constructed by concatenating the aperiodic prefixes of length 8 necklaces with weight 4 in reverse cool-lex ordering. The shorthand universal cycle is also a universal cycle for length 7 binary strings with weight 3 and 4.

Ruskey, Williams and Sawada [86] proved that shorthand universal cycle for fixed-weight binary strings can be constructed in $O(1)$-amortized time per bit using $O(n)$ space.

2.5 Applications

Universal cycles have many practical applications including dynamic connections in overlay networks [31], genomics [1], software calculation of the ruler function in computer words [57], and indexing a 1 in a computer word [59]. In Chapter 1, we outlined the details of a card trick application for de Bruijn sequences. This section outlines two more applications of universal cycles to give readers more understanding on how these sequences can be useful.

**Breaking key-lock system**

Consider a keyless-entry keypad which requires a 5-digit access code to login. Suppose we forget the password, then one approach to open the lock is to try every possibilities. There are ten possible values for each digit (0 - 9), therefore the number of possible passwords is $10^5$, that is 100000.

To input all 100000 possible passwords, we could arrange all 100000 possible 5-digit codes into one long sequence in the format 00000 · 00001 · · · 99999 according to lexicographic ordering. This sequence must contain all length 5 passwords as a substring. The length of such sequence, however, is 500000 digits long. To optimize the number of keys input, we can take advantage of the fact that there is no Enter key in the keypad. Observe that a continuous sequence would open the lock once the correct 5-digit code appears as a substring of the sequence, and de Bruijn sequences are the
shortest possible sequences that contain all possible 5-digit codes as a substring. We can thus make use of a de Bruijn sequence open the lock.

To illustrate how this process works, we consider a simpler example with a keypad accepting 2-digit codes with keys 1, 2 and 3. In this example, there are nine possible passwords, thus the continuous sequence is 18 digits long if we concatenate all possible 2-digit codes in lexicographic ordering. To refine our strategy, we rearrange the 2-digit codes and concatenate them in a different way. Consider the following sequence:

$$1121322331.$$ 

This sequence is created by appending an additional 1 in a de Bruijn sequence for $T(2, 3)$. It contains all possible 2-digit codes as a substring. By typing this sequence instead of the 18-digit sequence, we input only 10 digits. The improvement is more remarkable when we consider our original example with 5 digits and 10 values, which optimizes the number of entries from $10^5 \times 5 = 500000$ digits to $10^5 + (5-1) = 100004$ digits.

**DNA sequencing**

Universal cycles also have applications in the field of bioinformatics. Each strand of DNA is composed of four basic nucleotides adenine (A), cytosine (C), guanine (G), and thymine (T) that are ordered in some specific order. Modern DNA sequencing involves breaking up large DNA molecules and reassembling the fragments to remove overlappings between each fragment to obtain a shorter sequence. This method is commonly known as **shotgun sequencing**. To illustrate how universal cycles can be useful in this problem, consider a DNA strand with the following DNA fragments:

**GGA, ATT, GAT, TGC, TTG.**

There are 120 (5!) possible ways to rearrange the DNA fragments to reconstruct a sequence. However, not all sequences have the same length because of the overlapping between each DNA fragment. As an example, the ordering GGA, ATT, GAT, TGC,
TTG gives us a sequence of 13 nucleotides after removing the overlapping of each consecutive DNA fragments:

\[
\text{GGATTGATGCTTG.}
\]

We can order the DNA fragments in a different way. For example, the ordering GGA, GAT, TGC, ATT, TTG gives us another sequence which contains only 10 nucleotides after removing the overlapping of each consecutive DNA fragments:

\[
\text{GGATGCATTG.}
\]

The goal is to minimize the length of the resulting sequence while containing all DNA fragments as a substring. To obtain such an optimal sequence, biologists study the de Bruijn graph of DNA fragments and create an Euler path to find a linear universal string (a sequence that contains each element of a set as a linear substring exactly once) for DNA fragments. As an example, the sequence

\[
\text{GGATTGC}
\]

is a linear universal string which contains the DNA fragments GGA, GAT, TGC, ATT and TTG as a length 3 substring. The use of universal cycles in shotgun sequencing is considered more efficient than most other methods available [88].

2.6 Summary

We provide an overview on the existence results and applications of universal cycles, and summarize the common approaches to construct these sequences. Most of these approaches consider construction of de Bruijn sequences, but are not applicable to produce shorter universal cycles for subsets of length $n$ binary strings and $k$-ary strings. Furthermore, many of these constructions are not efficient, while some of them need prior knowledge on primitive feedback functions or de Bruijn sequences of a smaller dimension. These restrictions hinder the applications of universal cycles, and lead us to explore new efficient constructions of universal cycles for various interesting combinatorial objects.
Chapter 3

A simple de Bruijn sequence construction

Pick any length $n$ binary string $b_1 b_2 \cdots b_n$ and remove the first bit $b_1$. If $b_2 b_3 \cdots b_n 1$ is a necklace then append $b_1$ to the end of the remaining string; otherwise append $b_1$. Amazingly, by repeating this process, eventually all $2^n$ binary strings will be visited cyclically. In this chapter, we will prove that this shift rule yields a new construction for de Bruijn sequence over a binary alphabet: simply concatenate together the first bit of each string visited. Since this shift rule can easily be implemented in $O(n)$ time, the de Bruijn sequence can be generated in $O(n)$ time per bit. However, by studying the properties of this new sequence, we develop a more sophisticated approach to generate this new de Bruijn sequence in $O(1)$-amortized time per bit. We then extend this shift rule to construct de Bruijn sequence over a general alphabet.

3.1 Shift rule to generate de Bruijn sequence over a binary alphabet

To begin with, we consider an application of de Bruijn networks. A de Bruijn network is a network built based on a de Bruijn graph (mostly consider the set of length $n$ binary strings). These networks have applications on constructing interconnection networks and fault tolerance networks [96]. In Figure 3.1, a robot with limited memory and computation power is placed on an arbitrary vertex of a de Bruijn network. How can the robot travels each edge in the network exactly once and returns to the starting vertex? Observe that this task is equivalent to traversing the edges of an Euler cycle in the underlying de Bruijn graph.

One answer to this problem is to use a feedback shift register to compute the
next vertex in the de Bruijn network as described in Chapter 2.3.2. This can be done in $O(n)$ time and space. However, there is no known efficient method to find a feedback function for an arbitrary $n$, and this approach is only applicable for de Bruijn networks built based on a de Bruijn graph over a binary alphabet. Another approach is to construct a de Bruijn sequence and locate the position of the current length $n$ string in the de Bruijn sequence, and then traverses the edges of the de Bruijn network following the subsequent symbols of the de Bruijn sequence. This approach, however, requires $\Omega(k^n)$ space to memorize the whole de Bruijn sequence, which may far exceed the memory and computation power of the robot.

This application led us to take a completely different approach to generate de Bruijn sequences. We claim that the following function $f : \mathbf{B}(n) \rightarrow \mathbf{B}(n)$ induces a cyclic ordering for strings in $\mathbf{B}(n)$:

$$f(b_1b_2\cdots b_n) = \begin{cases} b_2b_3\cdots b_nb_1 & \text{if } b_2b_3\cdots b_n1 \in \mathbf{N}(n); \\ b_2b_3\cdots b_nb_1 & \text{otherwise.} \end{cases} \quad (3.1)$$

As an illustration, successive applications of this rule for $n = 5$ starting with 00000 produce the following listing (the underlined strings will be discussed later):

00000, \underline{00001}, 00011, 00111, 01111, 11111, 11110, 11101, 11011, 10111, 01110, 11100, 11001, 10011, 00110, 01101, 11010, 10110, 01100, 11000, 10001, \underline{00010}, 00101, 01011, 10110, 01101, 11010, 10101, 01010, 10100, 01001, 10010, \underline{00100}, \underline{01000}, 10000.
Observe that every string in $B(5)$ gets visited exactly once and that by applying one more application of the rule, we return to the first string 00000. Thus by the definition of $f$, a de Bruijn sequence is obtained by concatenating the first bit of each string. As an example, by concatenating the first bit of each string in the listing of $B(5)$, we obtain a de Bruijn sequence for $B(5)$ as follows:

$$000001111101110101100100.$$

This de Bruijn sequence differs from all other de Bruijn sequences generated by known constructions mentioned in Chapter 2. In particular, a reversed rotation of the sequence generated by the FKM construction described in Chapter 2.3.4 differs at the 14th bit:

$$000001111101110101100100110001.$$

Our shift rule creates a de Bruijn sequence for $B(n)$ for all $n$. We prove this fact in Chapter 3.2. Furthermore, we provide an efficient algorithm to generate the de Bruijn sequence in Chapter 3.3. We also introduce an application to find non-linear feedback functions for NLFSRs in Chapters 3.4. We then generalize the shift rule to general alphabets in Chapter 3.5 and prove its correctness in Chapter 3.6.

### 3.2 Proving the correctness of the shift rule over a binary alphabet

In this section we prove that the shift rule $f$ from (3.1) induces a cyclic ordering on the set of length $n$ binary strings.

**Theorem 3.2.1** The shift rule $f$ induces a cyclic ordering on $B(n)$.

This theorem is proved in two steps.

1. Show that the function $f$ is one-to-one (Lemma 3.2.2).

2. Show that every strings are reachable from $0^n$ by repeatedly applying $f$ (Lemma 3.2.4).
The first step alone is not sufficient since such a function could decompose $B(n)$ into more than one circuits. The left rotation of $\alpha = b_1 b_2 \cdots b_n$ is $b_2 b_3 \cdots b_n b_1$ and is denoted by $\text{LeftRotate}(\alpha)$. Let $\text{LeftRotate}^r(\alpha)$ denote the string that results from applying a left rotation $r$ times to $\alpha$. Thus $\text{LeftRotate}^r(\alpha) = b_{r+1} b_{r+2} \cdots b_n b_1 b_2 \cdots b_r$ when $0 \leq r < n$.

**Lemma 3.2.2** The function $f$ is one-to-one.

**Proof.** By contradiction. Consider two binary strings $\alpha = a_1 a_2 \cdots a_n$ and $\beta = b_1 b_2 \cdots b_n$ where $\alpha \neq \beta$ and $f(\alpha) = f(\beta)$. Clearly from the definition of $f$, we have $a_2 a_3 \cdots a_n = b_2 b_3 \cdots b_n$. Thus, since $\alpha \neq \beta$, $a_1 \neq b_1$. Without loss of generality, assume $a_1 = 1$ and $b_1 = 0$. If $a_2 a_3 \cdots a_n 1 = b_2 b_3 \cdots b_n 1$ is a necklace, then $f(\alpha) = a_2 a_3 \cdots a_n 0$ while $f(\beta) = b_2 b_3 \cdots b_n 1$, a contradiction. Otherwise if $a_2 a_3 \cdots a_n 1 = b_2 b_3 \cdots b_n 1$ is not a necklace, then $f(\alpha) = a_2 a_3 \cdots a_n 1$ while $f(\beta) = b_2 b_3 \cdots b_n 0$ which also contradicts with the assumption that $f(\alpha) = f(\beta)$. \qed

Now, revisit the example listing of $B(5)$ given in Chapter 3.1. Observe that given any $\alpha \in \mathcal{N}(5)$, the sequence of strings $\alpha$, $\text{LeftRotate}(\alpha)$, $\text{LeftRotate}^2(\alpha)$, $\text{LeftRotate}^3(\alpha)$ and $\text{LeftRotate}^4(\alpha)$ appears as a subsequence in the listing. In particular, the underlined strings are precisely the strings in $\text{Neck}(00001)$ and they appear in the order described. This property is the key to prove that all strings are reachable from $0^n$.

Formally, a string $\beta$ is reachable from $\alpha$ if $\beta$ can be obtained from $\alpha$ by repeatedly applying the shift rule $f$. Since $f$ is one-to one, the following inverse function is well defined:

$$f^{-1}(b_1 b_2 \cdots b_n) = \begin{cases} \overline{b_n} b_1 b_2 \cdots b_{n-1} & \text{if } b_1 b_2 \cdots b_{n-1}1 \in \mathcal{N}(n); \\ b_n b_1 b_2 \cdots b_{n-1} & \text{otherwise}. \end{cases}$$

This inverse function will be useful when proving the next lemma.

**Lemma 3.2.3** Let $\alpha \in \mathcal{N}(n)$ and $\beta = b_1 b_2 \cdots b_n \in \text{Neck}(\alpha)$. Then $\beta$ is reachable from $\alpha$.  

28
Proof. Apply induction on the number of 0s of \( \alpha \).

**Base case:** When the number of 0s in \( \alpha \) is zero, \( \alpha = 1^n \) and the only string in \( \text{Neck}(\alpha) \) is \( 1^n \). Each string in \( \text{Neck}(\alpha) \) is reachable from \( \alpha \).

**Inductive hypothesis:** Assume that given any \( \alpha \in \mathbb{N}(n) \) with \( k \) 0s where \( 0 \leq k < n \), each string \( \beta \in \text{Neck}(\alpha) \) is reachable from \( \alpha \).

**Inductive step:** We consider \( \alpha \in \mathbb{N}(n) \) with \( k + 1 \) 0s. Observe that \( \text{Neck}(\alpha) \) contains all the left rotations of \( \alpha \), that is \( \text{Neck}(\alpha) = \{\text{LeftRotate}^r(\alpha) \mid r \in \mathbb{Z} \text{ and } 0 \leq r \leq |\text{ap}(\alpha)| - 1\} \). A nested induction on \( r \) proves that \( \text{LeftRotate}^r(\alpha) \) is reachable from \( \alpha \).

**Base case:** When \( r = 0 \), \( \text{LeftRotate}^0(\alpha) = \alpha \) and is reachable from \( \alpha \).

**Inductive hypothesis:** Assume \( \text{LeftRotate}^t(\alpha) \) is reachable from \( \alpha \) where \( 0 \leq t < |\text{ap}(\alpha)| - 1 \).

**Inductive step:** Let \( \text{LeftRotate}^{t+1}(\alpha) = b_1b_2 \cdots b_n \). If \( b_1b_2 \cdots b_{n-1} \not\in \mathbb{N}(n) \), then observe that \( f^{-1}(b_1b_2 \cdots b_n) = b_n b_1b_2 \cdots b_{n-1} = \text{LeftRotate}^t(\alpha) \). Otherwise, \( b_n = 0 \) and \( f^{-1}(b_1b_2 \cdots b_n) = 1b_1b_2 \cdots b_{n-1} \) because \( b_1b_2 \cdots b_n \not\in \mathbb{N}(n) \) but \( b_1b_2 \cdots b_{n-1}1 \in \mathbb{N}(n) \). Now observe that \( b_1b_2 \cdots b_{n-1}1 \in \mathbb{N}(n) \) has \( k \) 0s and it is the necklace representative of \( 1b_1b_2 \cdots b_{n-1} \). Thus by the inductive hypothesis, the string \( 1b_1b_2 \cdots b_{n-1}1 \) is reachable from \( b_1b_2 \cdots b_{n-1}1 \). Finally, since we have \( f^{-1}(b_1b_2 \cdots b_{n-1}1) = 0b_1b_2 \cdots b_{n-1} = \text{LeftRotate}^t(\alpha) \), \( \text{LeftRotate}^{t+1}(\alpha) \) is reachable from \( \text{LeftRotate}^t(\alpha) \).

Thus by induction, \( \text{LeftRotate}^{r+1}(\alpha) \) is reachable from \( \text{LeftRotate}^r(\alpha) \). Since reachability is transitive, each string in \( \text{Neck}(\alpha) \) can be obtained from \( \alpha \) by repeatedly applying the shift rule \( f \).

By applying this lemma we now prove that every length \( n \) binary strings are reachable from \( 0^n \).

**Lemma 3.2.4** Each string \( \beta \in B(n) \) is reachable from \( 0^n \).
Proof. Apply induction on the weight (number of 1s) of $\beta$.

**Base case:** When the weight of $\beta$ is zero, the only string with weight equal to zero is $0^n$ which is reachable from $0^n$.

**Inductive hypothesis:** Assume any weight $k$ string is reachable from $0^n$.

**Inductive step:** Let $\beta$ be a string with weight $k+1$, and $\alpha = b_1b_2\cdots b_{n-1}1 \in \mathbb{N}(n)$ be the necklace representative of $\beta$. The string $\alpha$ has weight $k + 1$ since $\beta \in \text{Neck}(\alpha)$. By Lemma 3.2.3, $\beta$ is reachable from $\alpha$. Observe that $f^{-1}(\alpha) = 0b_1b_2\cdots b_{n-1}$, which is a string with weight $k$ that is reachable from $0^n$ by the assumption.

Thus by induction, each length $n$ binary string is reachable from $0^n$ by transitivity.

Together, Lemma 3.2.2 and Lemma 3.2.4 prove Theorem 3.2.1.

### 3.3 Efficient implementation for the shift rule over a binary alphabet

In this section, we introduce algorithms to generate the de Bruijn sequence based on the shift rule $f$. First we give a simple intuitive $O(n)$ per bit algorithm to generate the de Bruijn sequence, then we discuss ways to optimize the algorithm to generate the sequence in $O(1)$-amortized time per bit.

A simple approach to generate a de Bruijn sequence by the shift rule $f$ is to start with the string $0^n$ and repeatedly apply the function $f$ and outputting the first bit each time. Recall that in Chapter 2.1 we mentioned a membership tester for necklaces which can be implemented in $O(n)$ time using $O(n)$ space, thus the de Bruijn sequence can be generated in $O(n)$ time per bit using $O(n)$ space. Pseudocode of this simple algorithm is given in Algorithm 2.

To improve the algorithm to run in $O(1)$-amortized time per bit, we study the strings $b_1b_2\cdots b_n$ such that $b_2b_3\cdots b_n1$ is a necklace. In Table 3.1 we list the length 6
Algorithm 2 Simple shift-based algorithm to generate de Bruijn sequence for $B(n)$ in $O(n)$ time per bit.

1: procedure SimpleDeBruijn
2:   $b_1 b_2 \cdots b_n \leftarrow 0^n$
3:   do
4:     Print($b_1$)
5:     $b_1 b_2 \cdots b_n \leftarrow f(b_1 b_2 \cdots b_n)$
6:   while $b_1 b_2 \cdots b_n \neq 0^n$

binary strings obtained by starting from 000000 and successively applying the function $f$ a total of $2^6 - 1$ times. Each row ends with a string $b_1 b_2 \cdots b_n$ such that $b_2 b_3 \cdots b_n 1$ is a necklace, and hence when the function $f$ is applied to this final string, it will complement the final bit after rotation. This means that the first string $\alpha = b_1 b_2 \cdots b_n$ in each row has the property that $b_2 b_3 \cdots b_{n-1} 1$ is a necklace. Observe there are $2|N(6)| - 2 = 2(14) - 2 = 26$ rows in this table. In the third column of this table, the value $g(\alpha)$ corresponds to the number of strings in each row. More formally, $g(\alpha)$ is the number of successive applications of the function $f$ starting from $\alpha$ until a bit gets complemented by the function $f$.

Let $f^j(\alpha)$ denote successively applying the shift rule $f$ on $\alpha = b_1 b_2 \cdots b_n$ for $j$ times. Let $\alpha = b_1 b_2 \cdots b_n$ and $b_1 b_2 \cdots b_{n-1} 1 \in N(n)$. Then we define the following function $g : B(n) \rightarrow N$:

$$g(\alpha) = \begin{cases} 
1 & \text{if } b_2 b_3 \cdots b_n 1 \in N(n); \\
|\text{ap}(\alpha)| & \text{if } b_2 b_3 \cdots b_n 1 \notin N(n) \text{ and } \alpha \in N(n); \\
n - q & \text{if } b_2 b_3 \cdots b_n 1 \notin N(n) \text{ and } \alpha \notin N(n),
\end{cases}$$

where $q$ is the largest possible value such that $\alpha$ has suffix $0^q$.

Lemma 3.3.1 The function $g$ computes the smallest value $j$ such that $f^j(\alpha) \neq \text{LeftRotate}^j(\alpha)$ when $\alpha = b_1 b_2 \cdots b_n$ and $b_1 b_2 \cdots b_{n-1} 1 \in N(n)$.

Proof. By the definition of $f$, clearly $g(\alpha) = 1$ when $b_2 b_3 \cdots b_n 1 \in N(n)$. Otherwise consider two cases depending on whether or not $\alpha$ is a necklace.

Case 1: $\alpha$ is a necklace. If $\alpha$ is periodic with $p = |\text{ap}(\alpha)|$, then the substring $b_{p+1} b_{p+2} \cdots b_{2p}$ is lexicographically smaller than $b_1 b_{i+1} \cdots b_p$ for all $i > 1$. Thus
<table>
<thead>
<tr>
<th>$i$</th>
<th>$\alpha, f(\alpha), f(f(\alpha))$, …</th>
<th>$g(\alpha)$</th>
<th>Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>000000</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>000001</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>000011</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>000111</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>001111</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>011111</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>111111</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>111110, 111011, 110111, 101111</td>
<td>5</td>
<td>11111</td>
</tr>
<tr>
<td>9</td>
<td>011110, 111100, 111001, 100111, 100011</td>
<td>5</td>
<td>01111</td>
</tr>
<tr>
<td>10</td>
<td>001110, 111100, 111001, 110011, 101111</td>
<td>5</td>
<td>00111</td>
</tr>
<tr>
<td>11</td>
<td>000110</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>001101</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>13</td>
<td>011011, 110110, 101101</td>
<td>3</td>
<td>011</td>
</tr>
<tr>
<td>14</td>
<td>011010, 110100, 101001, 010011, 100111</td>
<td>5</td>
<td>01101</td>
</tr>
<tr>
<td>15</td>
<td>001100, 011000, 110001, 100001</td>
<td>4</td>
<td>0011</td>
</tr>
<tr>
<td>16</td>
<td>000010</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>17</td>
<td>000101</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>18</td>
<td>010111</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>19</td>
<td>010110, 101100, 011001, 110010, 100011</td>
<td>5</td>
<td>01011</td>
</tr>
<tr>
<td>20</td>
<td>010101</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>21</td>
<td>010100, 101000, 010001, 100101</td>
<td>4</td>
<td>0101</td>
</tr>
<tr>
<td>22</td>
<td>001010</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>23</td>
<td>000100</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>24</td>
<td>000100, 010000, 010000</td>
<td>3</td>
<td>001</td>
</tr>
</tbody>
</table>

Table 3.1: The cyclic order of $B(6)$ starting from 000000 induced by the function $f$. The rows break down the order based on when $f$ applies a complemented reversal.

$f^j(\alpha) = \text{LeftRotate}^j(\alpha)$ when $j < p$. Now consider $f^{p-1}(\alpha)$, observe that $f^{p-1}(\alpha) = 1^p a^p b_1 b_2 \cdots b_{p-1}$ and $a^p b_1 b_2 \cdots b_{p-1} 1 = \alpha \in N(n)$. Therefore $f^p(\alpha) \neq \text{LeftRotate}^p(\alpha)$ and $g(\alpha) = p = |a^p(\alpha)|$. When $\alpha$ is aperiodic, let $\gamma$ be the lexicographically least maximal substring of the form $0^*1^*$ within $\alpha$. Clearly $b_1 b_2 \cdots b_{|\gamma|} = \gamma$ because $\alpha \in N(n)$. Also since $b_2 b_3 \cdots b_{n+1} \notin N(n)$, there exist some substring $\beta$ within $b_1 b_2 \cdots b_{|\gamma|} 1 \cdots b_{n+1}$ which is lexicographically smaller than $b_i b_{i+1} \cdots b_{|\beta|+i-1}$ for all $1 < i < |\gamma|$. Thus $f^j(\alpha) = \text{LeftRotate}^j(\alpha)$ when $j < |\gamma|$. Then notice that $f^{|\gamma|}(\alpha)$ contains the suffix $\gamma$ which is strictly the lexicographically least maximal substring in $\alpha$. Thus $f^j(\alpha) = \text{LeftRotate}^j(\alpha)$
when \( j < n \). Now observe that \( f^{n-1}(\alpha) = b_n b_1 b_2 \cdots b_{n-1} = \alpha \in N(n) \), thus \( f^n(\alpha) \neq \text{LeftRotate}^n(\alpha) \) and \( g(\alpha) = n = |\text{ap}(\alpha)| \).

**Case 2:** \( \alpha \) is not a necklace. Since \( b_1 b_2 \cdots b_{n-1} 1 \in N(n) \) but \( \alpha \notin N(n) \), \( \alpha \) ends with the suffix \( 0^q \) with \( q \geq 1 \). Similarly, let \( \gamma \) be the lexicographically least maximal substring of the form \( 0^*1^* \) which ends with a 1 within \( \alpha \). Clearly \( b_1 b_2 \cdots b_{|\gamma|} = \gamma \) because \( b_1 b_2 \cdots b_{n-1} 1 \in N(n) \) and \( b_n = 0 \). The string \( f^j(\alpha) = \text{LeftRotate}^j(\alpha) \) when \( j < |\gamma| \) since \( b_{i+1} b_{i+2} \cdots b_{|\gamma|} \) is lexicographically larger than the suffix \( 0^q b_i \) for all \( 1 \leq i \leq |\gamma| \). Then consider \( f^{|\gamma|}(\alpha) \), it contains the suffix \( 0^q \gamma \) which is strictly the lexicographically least maximal substring in \( \alpha \). Thus \( f^j(\alpha) = \text{LeftRotate}^j(\alpha) \) when \( j < n - q \). Now observe that \( f^{n-q-1}(\alpha) = b_{n-q} 0^q \gamma b_{|\gamma|+1} b_{|\gamma|+2} \cdots b_{n-q-1} \) where \( 0^q \gamma b_{|\gamma|+1} b_{|\gamma|+2} \cdots b_{n-q-1} 1 \in N(n) \) because \( 0^q \gamma \) is strictly the lexicographically least maximal substring within \( \alpha \). Thus \( f^{n-q}(\alpha) \neq \text{LeftRotate}^{n-q}(\alpha) \) and \( g(\alpha) = n - q \).

Thus the function \( g \) computes the minimum value \( j \) such that \( f^j(\alpha) \neq \text{LeftRotate}^j(\alpha) \) as claimed. \( \square \)

To understand how Lemma 3.3.1 optimizes the runtime, we revisit the example in Table 3.1. Note that the concatenation of the first bit of each string in the second column of each row is highlighted in the final column labeled “Bits”. We thus obtain a de Bruijn sequence for \( B(6) \) by concatenating all the strings together in this final column. Also observe that the strings in each row of Table 3.1 are obtained by repeatedly applying a left rotation starting from the initial string \( \alpha \). Thus, given the value \( g(\alpha) \), we can output the string in the last column in constant time per bit. This leads to an optimized algorithm to generate the de Bruijn sequence given in Algorithm 3.

**Theorem 3.3.2** The algorithm \textsc{FastDeBruijn} generates a de Bruijn sequence for \( B(n) \) in \( O(1) \)-amortized time per bit.

**Proof.** The functions \( f \) and \( g \) can easily be computed in \( O(n) \) time using a standard membership tester for necklaces. Thus, it is easy to see that each iteration of the
Algorithm 3 Optimized shift-based algorithm to generate a de Bruijn sequence for \( B(n) \) in \( O(1) \)-amortized time per bit.

1: procedure FastDeBruijn
2: \( b_1b_2\cdots b_n \leftarrow 0^n \)
3: do
4: \( j \leftarrow g(b_1b_2\cdots b_n) \)
5: Print(\( b_1b_2\cdots b_j \))
6: \( b_1b_2\cdots b_n \leftarrow f(b_jb_{j+1}\cdots b_nb_1b_2\cdots b_{j-1}) \)
7: while \( b_1b_2\cdots b_n \neq 0^n \)

\textbf{do/while} loop requires \( O(n) \) time. The total number of strings \( b_1b_2\cdots b_n \in B(n) \) with \( b_2b_3\cdots b_n,1 \in N(n) \) is \( 2|N(n)| - 2 \), thus there are \( 2|N(n)| - 2 \) iterations of the \textbf{do/while} loop. Thus, the overall running time will be proportional to \( O(n|N(n)|) = \theta(2^n) \).

A complete C implementation is given in Appendix A.

### 3.4 Translate the shift rule over a binary alphabet into NLFSR

This section provides an application of the shift rule \( f \) relating to NLFSR. Recall that a NLFSR is a feedback shift register where its feedback function is non-linear. These functions have a vast number of applications (see Chapter 2.3.2 for details), however, there is no efficient way to find a NLFSR for an arbitrary \( n \). We provide a method to find a non-linear feedback function which generates a de Bruijn sequence for length \( n \) binary strings for an arbitrary \( n \). The method, however, is not efficient.

A product term of a boolean function of \( n \) variables \( \{b_1, b_2, \ldots, b_n\} \) is the expression \( \hat{b}_1\hat{b}_2\cdots\hat{b}_n \) such that \( \hat{b}_i \) is either the variable \( b_i \) itself or its complement \( \overline{b}_i \). Let \( p(\alpha) = \hat{b}_1\hat{b}_2\cdots\hat{b}_n \) denote a product term corresponds to a length \( n \) binary string \( \alpha = a_1a_2\cdots a_n \) such that \( \hat{b}_i = \overline{b}_i \) when \( a_i = 0 \), and \( \hat{b}_i = b_i \) when \( a_i = 1 \). Our shift rule \( f \) can be easily translated into the following feedback function \( f_q \):

\[
 f_q(b_1b_2\cdots b_n) = b_1 \oplus \sum_{b_2b_3\cdots b_n,1 \in N(n)} p(b_2b_3\cdots b_n).
\]

As an example, consider the case when \( n = 5 \), the length 4 binary strings \( \alpha \) with \( \alpha \cdot 1 \in N(5) \) are 0000, 0001, 0010, 0011, 0111 and 1111. The feedback function \( f_q \) for
\[ f_q(b_1b_2 \cdots b_5) = b_1 \oplus (b_2b_3b_4b_5 + \overline{b_2b_3b_4b_5} + b_2b_3b_4b_5 + \overline{b_2b_3b_4b_5} + b_2b_3b_4b_5 + b_2b_3b_4b_5). \]

This feedback function generates the same listing as the one given in Chapter 3.1.

**Lemma 3.4.1** The non-linear feedback function \( f_q \) has period \( 2^n \) for all \( n \).

**Proof.** The term \[ \sum_{b_2b_3 \cdots b_n \in \mathbb{N}(n)} p(b_2b_3 \cdots b_n) \] returns 1 when \( b_2b_3 \cdots b_n, 1 \) is a necklace, or otherwise returns 0. Thus, from the truth table of XOR, \( f_q \) outputs \( b_1 \) when \( b_2b_3 \cdots b_n, 1 \) is a necklace, and otherwise outputs \( b_1 \). Therefore \( f_q \) operates in the same way as the function \( f \) and has period \( 2^n \) by Theorem 3.2.1.

Recall that the necklaces in \( \mathbb{N}(n) \) can be generated in \( O(1) \)-amortized time per bit using the algorithm in [14], and \( |\mathbb{N}(n)| = \frac{1}{n} \sum_{d|n} \phi(d) 2^{n/d} \).

**Lemma 3.4.2** The non-linear feedback function \( f_q \) can be generated in \( O(2^n) \) time for all \( n \).

### 3.5 Generalizing the shift rule to general alphabets

In this section, we generalize the function \( f \) to general alphabets. We claim that the following function \( f_k : \mathbb{T}(n,k) \rightarrow \mathbb{T}(n,k) \) induces a cyclic ordering for the strings in \( \mathbb{T}(n,k) \):

\[
 f_k(a_1a_2 \cdots a_n) = \begin{cases} 
 a_2a_3 \cdots a_nb & \text{if } a_1 = k; \\
 a_2a_3 \cdots a_n(a_1 + 1) & \text{if } a_1 \neq k \text{ and } a_2a_3 \cdots a_n(a_1 + 1) \in \mathbb{N}(n,k); \\
 a_2a_3 \cdots a_na_1 & \text{otherwise}, 
\end{cases}
\]  

(3.2)

where \( b > 0 \) is the largest possible value such that \( a_2a_3 \cdots a_nb \) is not a necklace, or \( b = 1 \) if no such value exists. Notice that \( b = 1 \) if and only if \( a_2a_3 \cdots a_n = 1^{n-1} \), also \( f_2 = f \) when we use the binary notation for strings in \( \mathbb{T}(n,2) \).

As an example, successive applications of this rule for the set \( \mathbb{T}(3,3) \) starting with the string 111 produces the following listing:
Observe that every strings in $T(3, 3)$ get visited exactly once and that by applying one more application of the rule, we return to the first string 111. In next section, we prove that $f_k$ produces a de Bruijn sequence for $k$-ary strings of length $n$ by concatenating the first symbol of each string. As an example, by concatenating the first symbol of each string in the listing of $T(3, 3)$, we obtain a de Bruijn sequence for $T(3, 3)$ as follows:

$$111222333232212312113213313.$$ 

This generalized shift rule creates a de Bruijn sequence for $k$-ary strings of length $n$ for all $n$ and $k$.

### 3.6 Proving the correctness of the shift rule over a general alphabet

In this section we prove that the shift rule $f_k$ from (3.2) induces a cyclic ordering on the set of $k$-ary strings of length $n$.

**Theorem 3.6.1** The shift rule $f_k$ induces a cyclic ordering on $T(n, k)$.

This theorem is proved in a similar way as Theorem 3.2.1 which involves two steps.

1. Show that the function $f_k$ is one-to-one (Lemma 3.6.2).

2. Show that every strings are reachable from $1^n$ by applying $f_k$ (Lemma 3.6.5).

**Lemma 3.6.2** The function $f_k$ is one-to-one.

**Proof.** By contradiction. Consider two $k$-ary strings $\alpha = a_1a_2\cdots a_n$ and $\beta = b_1b_2\cdots b_n$ where $\alpha \neq \beta$ and $f_k(\alpha) = f_k(\beta)$. Clearly from the definition of $f_k$, we have $a_2a_3\cdots a_n = b_2b_3\cdots b_n$. Thus, since $\alpha \neq \beta$, $a_1 \neq b_1$. We then consider two cases.
Case 1: $a_1 \neq k$ and $b_1 \neq k$. There are 3 subcases. If $a_2a_3\cdots a_n(a_1 + 1)$ and $b_2b_3\cdots b_n(b_1 + 1)$ are both necklaces, then $f_k(\alpha) = a_2a_3\cdots a_n(a_1 + 1)$ while $f_k(\beta) = b_2b_3\cdots b_n(b_1 + 1)$, a contradiction to the assumption that $f_k(\alpha) = f_k(\beta)$ since $a_1 \neq b_1$. Then without loss of generality, assume $a_2a_3\cdots a_n(a_1 + 1)$ is a necklace but $b_2b_3\cdots b_n(b_1 + 1)$ is not a necklace. Then $a_1 > b_1$ by the definition of necklace. Thus $f_k(\alpha) = a_2a_3\cdots a_n(a_1 + 1)$ while $f_k(\beta) = b_2b_3\cdots b_nb_1$, also a contradiction to the assumption that $f_k(\alpha) = f_k(\beta)$ since $a_1 > b_1$. Otherwise if both $a_2a_3\cdots a_n(a_1 + 1)$ and $b_2b_3\cdots b_n(b_1 + 1)$ are not necklaces, then $f_k(\alpha) = a_2a_3\cdots a_nb_1$ while $f_k(\beta) = b_2b_3\cdots b_nb_1$, which also contradicts with the assumption that $f_k(\alpha) = f_k(\beta)$ since $a_1 \neq b_1$.

Case 2: Without loss of generality, $a_1 = k$ and $b_1 \neq k$. If $a_2a_3\cdots a_n = b_2b_3\cdots b_n = 1^{n-1}$, then $f_k(\alpha) = 1^n$ and $f_k(\beta) = 1^{n-1}(b_1 + 1)$, which contradicts with the assumption that $f_k(\alpha) = f_k(\beta)$ since $b_1 + 1 > 1$. Then if $b_2b_3\cdots b_n \neq 1^{n-1}$ and $b_2b_3\cdots b_n(b_1 + 1)$ is a necklace, then $f_k(\alpha) = a_2a_3\cdots a_nb$ for some maximum value of $b$ such that $a_2a_3\cdots a_nb$ is not a necklace, and $f_k(\beta) = b_2b_3\cdots b_n(b_1 + 1)$. Since $f_k(\alpha) = a_2a_3\cdots a_nb$ is not a necklace while $f_k(\beta) = b_2b_3\cdots b_n(b_1 + 1)$ is a necklace, clearly $f_k(\alpha) \neq f_k(\beta)$ which contradicts with the assumption that $f_k(\alpha) = f_k(\beta)$. Otherwise if $b_2b_3\cdots b_n(b_1 + 1)$ is not a necklace, then $f_k(\alpha) = a_2a_3\cdots a_nb$ for some maximum value of $b$ such that $a_2a_3\cdots a_nb$ is not a necklace, and $f_k(\beta) = b_2b_3\cdots b_nb_1$. Since $b_2b_3\cdots b_n(b_1 + 1)$ is not a necklace, $b \geq b_1 + 1 > b_1$, which also contradicts with the assumption that $f_k(\alpha) = f_k(\beta)$.

Therefore, the function $f_k$ is one-to-one. \hfill $\Box$

Figure 3.2 illustrates the one-to-one relationship of incoming edges and outgoing edges of two vertices in the de Bruijn graph for $T(6,6)$ with respect to the application of $f_k$. The vertex on the left can potentially be extended to a necklace while the vertex on the right cannot.
Figure 3.2: The incoming edges and outgoing edges of two vertices in the de Bruijn graph for $T(6, 6)$ with respect to the application of $f_k$.

Since $f_k$ is one-to one, the following inverse function is well defined:

$$f_k^{-1}(\alpha) = \begin{cases} 
ka_1a_2 \cdots a_{n-1} & \text{if } a_1a_2 \cdots a_{n-1}(a_n + 1) \text{ is a necklace,} \\
(a_n - 1)a_1a_2 \cdots a_{n-1} & \text{if } \alpha \text{ is a necklace and } \alpha \neq 1^n; \\
a_na_1a_2 \cdots a_{n-1} & \text{otherwise.}
\end{cases}$$

Lemma 3.6.3 The function $f_k^{-1}$ is an inverse function of $f_k$.

Proof. Let $\alpha = a_1a_2 \cdots a_n \in T(n, k)$. We prove that $f_k^{-1}$ is an inverse function of $f_k$ by showing that $f_k(f_k^{-1}(\alpha)) = \alpha$. We consider the following three cases.

Case 1: $a_1a_2 \cdots a_{n-1}(a_n + 1)$ is a necklace and $\alpha = 1^n$ or $\alpha$ is not a necklace: By the definition of $f_k^{-1}$, $f_k^{-1}(\alpha) = ka_1a_2 \cdots a_{n-1}$. Now consider $f_k(f_k^{-1}(\alpha))$. By the definition of $f_k$, $f_k(f_k^{-1}(\alpha)) = a_1a_2 \cdots a_{n-1}b$ where $b$ is the largest possible value such that $a_1a_2 \cdots a_{n-1}b$ is not a necklace, or $b = 1$ if no such value exists. Hence if $\alpha = 1^n$, then $b = a_n = 1$ since $a_1a_2 \cdots a_{n-1}b$ is a necklace for all $b \geq 1$. Otherwise, $b = a_n$ since $a_1a_2 \cdots a_{n-1}(a_n + 1)$ is a necklace but $\alpha = a_1a_2 \cdots a_n$ is not a necklace.

Case 2: $\alpha$ is a necklace and $\alpha \neq 1^n$: By the definition of $f_k^{-1}$, $f_k^{-1}(\alpha) = (a_n -
1)a_1a_2\cdots a_{n-1}. Then by the definition of \( f_k \), 
\[ f_k(f_k^{-1}(\alpha)) = a_1a_2\cdots a_{n-1}(a_n - 1 + 1) = \alpha. \]

**Case 3:** Otherwise: By the definition of \( f_k^{-1} \) and \( f_k \), clearly 
\[ f_k(f_k^{-1}(\alpha)) = a_1a_2\cdots a_n = \alpha \] 
since \( \alpha \notin \mathbb{N}(n,k) \) and \( a_1a_2\cdots a_{n-1}(a_n + 1) \notin \mathbb{N}(n,k) \).

Therefore, \( f_k(f_k^{-1}(\alpha)) = \alpha \) and \( f_k^{-1} \) is an inverse function of \( f_k \).

We use the inverse function \( f_k^{-1} \) to prove the next lemma.

**Lemma 3.6.4** Let \( \alpha = a_1a_2\cdots a_n \in \mathbb{N}(n,k) \). Then every strings in \( \text{Neck}(\alpha) \) are reachable from \( \alpha \).

**Proof.** Let \( w(\alpha) = kn - \sum_{i=1}^{n} a_i \). Apply strong induction on \( w(\alpha) \) of \( \alpha \).

**Base case:** We consider the cases when \( w(\alpha) = 0 \) and \( w(\alpha) = 1 \). When \( w(\alpha) = 0 \), the only necklace is \( k^n \). The only string in \( \text{Neck}(k^n) \) is \( k^n \). When \( w(\alpha) = 1 \), the only necklace is \( (k-1)k^{n-1} \). Observe that by applying the function \( f_k \) by \( n+1 \) times, we get all strings in \( \text{Neck}((k-1)k^{n-1}) \) and \( k^n \). Thus, each string in \( \text{Neck}(\alpha) \) is reachable from \( \alpha \) when \( w(\alpha) \in \{0,1\} \).

**Inductive hypothesis:** Assume that for every \( \alpha \) with \( w(\alpha) \leq j \) where \( 0 \leq j < kn - k \), each string in \( \text{Neck}(\alpha) \) is reachable from \( \alpha \).

**Inductive step:** We consider the case when \( w(\alpha) = j + 1 \). Observe that \( \text{Neck}(\alpha) \) contains all the left rotations of \( \alpha \), that is \( \text{Neck}(\alpha) = \{\text{LeftRotate}^r(\alpha) \mid r \in \mathbb{Z} \text{ and } 0 \leq r \leq |\text{ap}(\alpha)| - 1 \} \). A nested induction on \( r \) proves that \( \text{LeftRotate}^r(\alpha) \) is reachable from \( \alpha \).

**Base case:** When \( r = 0 \), \( \text{LeftRotate}^0(\alpha) = \alpha \) and is reachable from \( \alpha \).

**Inductive hypothesis:** Assume \( \text{LeftRotate}^t(\alpha) \) is reachable from \( \alpha \) where \( 0 \leq t < |\text{ap}(\alpha)| - 1 \).

**Inductive step:** Let \( \text{LeftRotate}^{t+1}(\alpha) = \beta = b_1b_2\cdots b_n \). Clearly \( \alpha, \beta \neq 1^n \) since \( j + 1 > 0 \). The string \( b_1b_2\cdots b_n \notin \mathbb{N}(n,k) \) since \( t + 1 > 0 \). If
\[ b_1b_2\cdots b_{n-1}(b_n + 1) \notin \mathbb{N}(n, k), \] then clearly \( f_k^{-1} (\beta) = b_nb_1b_2\cdots b_{n-1} = \text{LeftRotate}^t (\alpha). \) Otherwise if \( b_1b_2\cdots b_{n-1}(b_n + 1) \in \mathbb{N}(n, k), \) then \( f_k^{-1} (\beta) = kb_1b_2\cdots b_{n-1}. \) Observe that \( w(kb_1b_2\cdots b_{n-1}) = j + 1 + b_n - k \leq j \) since \( k \geq b_n + 1 \) because \( b_1b_2\cdots b_{n-1}(b_n + 1) \in \mathbb{N}(n, k). \) Furthermore, \( b_1b_2\cdots b_{n-1}k \) is a necklace representative of \( kb_1b_2\cdots b_{n-1} \) since \( b_1b_2\cdots b_{n-1}(b_n + 1) \in \mathbb{N}(n, k) \) and \( k \geq b_n + 1. \) Therefore, the string \( kb_1b_2\cdots b_{n-1}k \) is reachable from its necklace representative \( b_1b_2\cdots b_{n-1}k \) by the inductive hypothesis. Now observe that \( f_k^{-1} (b_1b_2\cdots b_{n-1}k) = (k - 1)b_1b_2\cdots b_{n-1} \) with \( w((k - 1)b_1b_2\cdots b_{n-1}) = j + 2 + b_n - k. \) If \( w((k - 1)b_1b_2\cdots b_{n-1}) = j + 2 + b_n - k = j + 1, \) then \( b_n = k - 1 \) and thus \( (k - 1)b_1b_2\cdots b_{n-1} = b_nb_1b_2\cdots b_{n-1} = \text{LeftRotate}^t (\alpha). \) Otherwise, let \( w((k - 1)b_1b_2\cdots b_{n-1}) = j + 1 + h - k \) for some \( h \) such that \( b_n < h < k - 1, \) observe that \( b_1b_2\cdots b_{n-1}h \in \mathbb{N}(n, k) \) since \( b_1b_2\cdots b_{n-1}(b_n + 1) \in \mathbb{N}(n, k) \) and \( h \geq b_n + 1. \) Thus, \( f_k^{-1} (b_1b_2\cdots b_{n-1}h) = (h - 1)b_1b_2\cdots b_{n-1}. \) By repeatedly applying the same argument, the strings \( (k - 1)b_1b_2\cdots b_{n-1} \) and \( (h - 1)b_1b_2\cdots b_{n-1} \) are reachable from \( b_1b_2\cdots b_{n-1}(b_n + 1), \) where \( f_k^{-1} (b_1b_2\cdots b_{n-1}(b_n + 1)) = b_nb_1b_2\cdots b_{n-1} = \text{LeftRotate}^t (\alpha) \) since \( b_1b_2\cdots b_{n-1}(b_n + 1) \in \mathbb{N}(n, k). \)

Thus by induction, \( \text{LeftRotate}^{t+1} (\alpha) \) is reachable from \( \text{LeftRotate}^t (\alpha). \) Since reachability is transitive, each string in \( \text{Neck} (\alpha) \) can be obtained from \( \alpha \) by repeatedly applying the shift rule \( f_k. \)

By applying this lemma we now prove that every \( k \)-ary strings of length \( n \) are reachable from \( 1^n. \)

**Lemma 3.6.5** Each string \( \beta \in \mathbb{T}(n, k) \) is reachable from \( 1^n. \)

**Proof.** Let \( t \) be the number of characters in \( \beta \) that are not equal to 1. Apply induction on \( t \) of \( \beta. \)

**Base case:** When \( t = 0, \) the only string with \( t = 0 \) is \( 1^n \) which is reachable from \( 1^n. \)

**Inductive hypothesis:** Assume any string in \( \mathbb{T}(n, k) \) that has \( t \) characters not equal to 1 is reachable from \( 1^n, \) where \( 0 \leq t < n. \)

40
**Inductive step:** Let $\beta$ be a string with $t + 1$ characters not equal to 1, and $\alpha = a_1a_2\cdots a_n \in \text{N}(n,k)$ be the necklace representative of $\beta$. The necklace $\alpha$ also has $t + 1$ characters not equal to 1 because $\beta \in \text{Neck}(\alpha)$. Also, $\alpha \neq 1^n$ since $t + 1 > 0$. The character $a_n > 1$ since $\alpha \in \text{N}(n,k)$ and $\alpha \neq 1^n$. By Lemma 3.6.4, $\beta$ is reachable from $\alpha$. The string $f_k^{-1}(\alpha) = (a_n - 1)a_1a_2\cdots a_{n-1}$ since $\alpha \in \text{N}(n,k)$. If $a_n - 1 = 1$, then $(a_n - 1)a_1a_2\cdots a_{n-1}$ has $t$ characters not equal to 1 and by the inductive hypothesis, $f_k^{-1}(\alpha)$ is reachable from $1^n$. Otherwise if $a_n - 1 > 1$, then $f_k^{-1}(\alpha) = (a_n - 1)a_1a_2\cdots a_{n-1}$ which still has $t + 1$ characters not equal to 1. The necklace representative $\gamma = b_1b_2\cdots b_n$ of $\text{Neck}(f_k^{-1}(\alpha))$ also has $t + 1$ characters not equal to 1 with $b_n > 1$ since $\gamma \in \text{N}(n,k)$. Thus by Lemma 3.6.4, $f_k^{-1}(\alpha)$ is reachable from $\gamma$. The string $f_k^{-1}(\gamma) = (b_n - 1)b_1b_2\cdots b_{n-1}$ since $\gamma \in \text{N}(n,k)$ and $\gamma \neq 1^n$ because $t + 1 > 0$. By repeatedly applying Lemma 3.6.4 and $f_k^{-1}$, one character $a_i > 1$ of $\alpha$ would decrement to 1 and thus the updated string has $t$ characters not equal to 1, which is reachable from $1^n$ by the assumption.

Thus by induction, each $k$-ary string of length $n$ is reachable from $1^n$ by transitivity.  

Together, Lemma 3.6.2 and Lemma 3.6.5 prove Theorem 3.6.1.

### 3.7 Efficient implementation for the shift rule over a general alphabet

The de Bruijn sequence constructed by $f_k$ can be generated in a similar way as $f$. Pseudocode of this simple algorithm `SIMPLESHIFT` is given in Algorithm 4. The function $f_k$ is the application of the necklace membership tester mentioned in Chapter 2.1. The algorithm performs $k$ necklace membership testers at the worst case. Thus the algorithm generates a de Bruijn sequence in $O(kn)$ time per symbol.

The algorithm can be optimized to run in $O(1)$-amortized time per symbol by focusing on the strings $\alpha = a_1a_2\cdots a_n$ such that $f_k(\alpha) \neq \text{LeftRotate}(\alpha)$. By the defi-
Algorithm 4 Simple shift-based algorithm to generate a de Bruijn sequence for $T(n,k)$ in $O(kn)$ time per symbol.

1: procedure SIMPLESHIFT
2:   $a_1a_2\cdots a_n \leftarrow 1^n$
3:   do
4:      Print($a_1$)
5:      $a_1a_2\cdots a_n \leftarrow f_k(a_1a_2\cdots a_n)$
6:   while $a_1a_2\cdots a_n \neq 1^n$

If, the strings that have such property either have $a_1 = k$ and $a_2a_3\cdots a_na_1$ is a necklace, or $a_1 \neq k$ and $a_2a_3\cdots a_n(a_1+1)$ is a necklace.

In Table 3.2 we list the 3-ary strings of length 4 obtained by starting from 1111 and successively applying the function $f_k$ a total of $3^4 = 81$ times. Each row ends with a string $\beta$ such that $f_k(\beta) \neq \text{LeftRotate}(\beta)$. Hence when the function $f_k$ is applied to this final string, it changes the value of the final character after rotation.

This means that the first string $\alpha = a_1a_2\cdots a_n$ in each row is either a necklace, or has the property that $a_1a_2\cdots a_{n-1}(a_n+1) \in N(n,k)$. Observe there are 37 rows in this table, which is bounded by $2|N(4,3)| = 2(24) = 48$. In the third column of this table, the value $g_k(\alpha)$ corresponds to the number of strings in each row. More formally, $g_k(\alpha)$ is the number of successive applications of the function $f_k$ starting from $\alpha$ until the value of a character is changed by the function $f_k$.

Let $f_j^k(\alpha)$ denote successively applying the shift rule $f_k$ on $\alpha = a_1a_2\cdots a_n$ for $j$ times. We also define two additional functions $q$ and $s$. Let $q : T(n,k) \rightarrow \mathbb{Z}$ denote a function that returns the smallest possible index $i$ on $\alpha = a_1a_2\cdots a_n$ such that $a_{i+1}a_{i+2}\cdots a_na_1a_2\cdots a_{i-1}(a_i+1) \in N(n,k)$ with $a_1a_2\cdots a_{i-1} = a_{i+1}a_{i+2}\cdots a_{2i-1}$ and $a_{2i} \in \{a_i, a_i+1\}$, or returns $n$ if no such index exists. Let $s : T(n,k) \rightarrow \mathbb{N}$ denote a function that returns the smallest possible index $i$ on $\alpha = a_1a_2\cdots a_n$ such that $a_{i+1}a_{i+2}\cdots a_na_1a_2\cdots a_{i-1}(a_i+1) \in N(n,k)$ and $a_1a_2\cdots a_i$ is aperiodic, or otherwise return $r$ which is the smallest value such that $\text{LeftRotate}^r(\alpha) \in N(n,k)$ if no such $i < r$ exists. As an example of the applications of $q$ and $s$, when $\alpha = 1213 \in T(4,3)$, then $q(1213) = 2$ since when $i = 2$, we have $a_3a_4a_1(a_2+1) = 1313 \in N(4,3)$, $a_1 = a_3 = 1$, 42
<table>
<thead>
<tr>
<th>$i$</th>
<th>$\alpha, f_k(\alpha), f_k(f_k(\alpha)), \ldots$</th>
<th>$g_k(\alpha)$</th>
<th>firstchar</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1111</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1112</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1122</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1222</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>2222</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>2223</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>2233</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>2333</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>3333</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>10</td>
<td><strong>3332, 3323, 3233</strong></td>
<td>3</td>
<td>333</td>
</tr>
<tr>
<td>11</td>
<td><strong>2332, 3322, 3223</strong></td>
<td>3</td>
<td>233</td>
</tr>
<tr>
<td>12</td>
<td><strong>2232</strong></td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>13</td>
<td>2323, 3232</td>
<td>2</td>
<td>23</td>
</tr>
<tr>
<td>14</td>
<td><strong>2322, 3222</strong></td>
<td>2</td>
<td>23</td>
</tr>
<tr>
<td>15</td>
<td><strong>2221, 2212, 2122</strong></td>
<td>3</td>
<td>222</td>
</tr>
<tr>
<td>16</td>
<td>1223, 2231, 2312, 3122</td>
<td>4</td>
<td>1223</td>
</tr>
<tr>
<td>17</td>
<td><strong>1221, 2211, 2112</strong></td>
<td>3</td>
<td>122</td>
</tr>
<tr>
<td>18</td>
<td>1123</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>19</td>
<td>1232, 2321, 3212, 2123</td>
<td>4</td>
<td>1232</td>
</tr>
<tr>
<td>20</td>
<td>1233, 2331, 3312, 3123</td>
<td>4</td>
<td>1233</td>
</tr>
<tr>
<td>21</td>
<td><strong>1231, 2311, 3112</strong></td>
<td>3</td>
<td>123</td>
</tr>
<tr>
<td>22</td>
<td><strong>1211</strong></td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>23</td>
<td>1212, 2121</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>24</td>
<td>1213, 2131</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>25</td>
<td>1313, 3131</td>
<td>2</td>
<td>13</td>
</tr>
<tr>
<td>26</td>
<td><strong>1312, 3121</strong></td>
<td>2</td>
<td>13</td>
</tr>
<tr>
<td>27</td>
<td><strong>1211, 2111</strong></td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>28</td>
<td>1113</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>29</td>
<td>1132</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>30</td>
<td>1322, 3221, 2213, 2132</td>
<td>4</td>
<td>1322</td>
</tr>
<tr>
<td>31</td>
<td>1323, 3231, 2313, 3132</td>
<td>4</td>
<td>1323</td>
</tr>
<tr>
<td>32</td>
<td><strong>1321, 3211, 2113</strong></td>
<td>3</td>
<td>132</td>
</tr>
<tr>
<td>33</td>
<td>1133</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>34</td>
<td>1332, 3321, 3213, 2133</td>
<td>4</td>
<td>1332</td>
</tr>
<tr>
<td>35</td>
<td>1333, 3331, 3313, 3133</td>
<td>4</td>
<td>1333</td>
</tr>
<tr>
<td>36</td>
<td><strong>1331, 3311, 3113</strong></td>
<td>3</td>
<td>133</td>
</tr>
<tr>
<td>37</td>
<td><strong>1131, 1311, 3111</strong></td>
<td>3</td>
<td>113</td>
</tr>
</tbody>
</table>

Table 3.2: The cyclic order of $T(4,3)$ starting from 1111 induced by the function $f_k$. The underlined strings are of the form $\alpha = a_1 a_2 \cdots a_n$ such that $\alpha$ is not a necklace but $a_1 a_2 \cdots a_{n-1} (a_n + 1)$ is a necklace.
and \( a_4 = a_2 + 1 = 3 \) while \( i = 1 \) does not satisfy these properties. On the other hand, when \( \alpha = 1212131212 \in T(12,3) \), then \( s(\alpha) = 2 \) since when \( i = 2 \), we have \( a_3 a_4 \cdots a_9 a_1 (a_2 + 1) = 1212131212 \in N(12,3) \) and \( a_1 a_2 = 12 \) is aperiodic while \( i = 1 \) does not satisfy these properties, and \( r = 8 \) where \( i < r \).

Let \( \alpha = a_1 a_2 \cdots a_n \in T(n,k) \) such that either \( \alpha \) is a necklace, or \( a_1 a_2 \cdots a_{n-1} (a_n + 1) \in N(n,k) \). We also introduce the notation \( z_i = \min(a_i + 1, k) \) to combine the cases when \( f_k(\alpha) \neq \text{LeftRotate}(\alpha) \). That is, \( f_k(\alpha) \neq \text{LeftRotate}(\alpha) \) if and only if \( a_2 a_3 \cdots a_n z_1 \in N(n,k) \). Then we define the following function \( g_k : T(n,k) \to \mathbb{N} \):

\[
g_k(\alpha) = \begin{cases} 
1 & \text{if } a_2 a_3 \cdots a_n z_1 \in N(n,k); \\
|\alpha p| & \text{if } a_2 a_3 \cdots a_n z_1 \notin N(n,k) \text{ and } \alpha \in N(n,k) \text{ and } |\alpha p| < n; \\
q(\alpha) & \text{if } a_2 a_3 \cdots a_n z_1 \notin N(n,k) \text{ and } \alpha \in N(n,k) \text{ and } |\alpha p| = n; \\
s(\alpha) & \text{otherwise.}
\end{cases}
\]

**Lemma 3.7.1** The function \( g_k \) computes the minimal value \( j \) such that \( f_k^j(\alpha) \neq \text{LeftRotate}^j(\alpha) \) when \( \alpha = a_1 a_2 \cdots a_n \) is either a necklace, or \( a_1 a_2 \cdots a_{n-1} (a_n + 1) \in N(n,k) \).

**Proof.** By the definition of \( f_k \), there are two cases when \( g_k(\alpha) = 1 \). When \( a_1 = k \) and \( a_2 a_3 \cdots a_n a_1 \in N(n,k) \), then \( f_k(\alpha) = a_2 a_3 \cdots a_n b \) for some \( b \neq a_1 \) since \( a_2 a_3 \cdots a_n a_1 \in N(n,k) \). Thus \( g_k(\alpha) = 1 \). Then when \( a_1 \neq k \) and \( a_2 a_3 \cdots a_n (a_1 + 1) \in N(n,k) \), then \( f_k(\alpha) = a_2 a_3 \cdots a_n (a_1 + 1) \) and clearly \( g_k(\alpha) = 1 \). Thus, \( g_k(\alpha) = 1 \) when \( a_2 a_3 \cdots a_n z_1 \in N(n,k) \). We then consider the remaining cases.

**Case 1:** \( \alpha \) is a periodic necklace: If \( \alpha \) is periodic with \( p = |\alpha p| \), then the substring \( a_{p+1} a_{p+2} \cdots a_{2p} \) is lexicographically smaller than \( a_i a_{i+1} \cdots a_p \) for all \( i > 1 \). Thus \( f_k^1(\alpha) = \text{LeftRotate}^j(\alpha) \) when \( j < p \). Now consider \( f_k^{p-1}(\alpha) \), observe that \( f_k^{p-1}(\alpha) = a_p \cdot \alpha p a_{p+1} \cdots a_{p-1} a_1 a_2 \cdots a_{p-1} a_p \in N(n,k) \) since \( a_p \cdot \alpha p a_{p+1} \cdots a_{p-1} a_p = \alpha \in N(n,k) \). Therefore \( f_k^p(\alpha) \neq \text{LeftRotate}^p(\alpha) \) and \( g_k(\alpha) = p = |\alpha p| \).

**Case 2:** \( \alpha \) is an aperiodic necklace: There are two subcases. Suppose there exist a smallest possible index \( i \) such that \( a_{i+1} a_{i+2} \cdots a_n a_1 a_2 \cdots a_i (a_i + 1) \in
$N(n, k)$ with $a_1a_2\cdots a_{i-1} = a_{i+1}a_{i+2}\cdots a_{2i-1}$ and $a_{2i} \in \{a_i, a_i + 1\}$. Since $a_1a_2\cdots a_{i-1} = a_{i+1}a_{i+2}\cdots a_{2i-1}$, the substring $a_{i+1}a_{i+2}\cdots a_{2i}$ is lexicographically smaller than $a_ja_{j+1}\cdots a_{i-1}$ for all $j > 1$. Thus $f_k^1(\alpha) = \text{LeftRotate}^i(\alpha)$ when $j < i - 1$. Now observe that $f_k^{j-1}(\alpha) = a_1a_{i+1}a_{i+2}\cdots a_n$ for all $i < k$ and $a_{i+1}a_{i+2}\cdots a_n \in N(n, k)$ since $a_{i+1}a_{i+2}\cdots a_n a_1a_2\cdots a_{i-1}(a_i + 1) \in N(n, k)$. Therefore $f_k^i(\alpha) \neq \text{LeftRotate}^i(\alpha)$ and $g_k(\alpha) = i = q(\alpha)$. Otherwise, $\alpha$ contains the prefix $a_1a_2\cdots a_i$ such that $a_{i+1}a_{i+2}\cdots a_n \notin N(n, k)$ for all values of $i \leq n/2$ since $a_{i+1}a_{i+2}\cdots a_{2i}$ must be lexicographically larger than or equal to $a_1a_2\cdots a_i$ because $\alpha$ is a necklace. Thus $f_k^j(\alpha) = \text{LeftRotate}^i(\alpha)$ when $i < n/2$. Since $\alpha$ is an aperiodic necklace, $a_1a_2\cdots a_{[n/2]}$ is strictly lexicographically smaller than $a_{j+1}a_{j+2}\cdots a_n$ for all $j > [n/2]$. Thus $f_k^j(\alpha) = \text{LeftRotate}^i(\alpha)$ when $j < i$. Now consider $f_k^{n-1}(\alpha)$, observe $f_k^{n-1}(\alpha) = a_n a_1a_2\cdots a_{n-1} a_1a_2\cdots a_{n-1} \in N(n, k)$ since $\alpha = a_1a_2\cdots a_n \in N(n, k)$. Therefore, $f_k^n(\alpha) \neq \text{LeftRotate}^n(\alpha)$ and $g_k(\alpha) = n = q(\alpha)$.

Case 3: $\alpha$ is not a necklace: There are two subcases. First we prove by contradiction that $a_1a_2\cdots a_i$ must be aperiodic if $i$ is the smallest index such that $f_k^i(\alpha) \neq \text{LeftRotate}^i(\alpha)$. Assume $a_1a_2\cdots a_i$ is periodic with $(a_1a_2\cdots a_{j})^t = a_1a_2\cdots a_i$ for some $t > 1$ and $f_k^i(\alpha) \neq \text{LeftRotate}^i(\alpha)$, then observe that $a_{j+1}a_{j+2}\cdots a_{n} a_1a_2\cdots a_{j-1}(a_j + 1) \in N(n, k)$, a contradiction to the assumption that $i$ is the smallest index such that $f_k^i(\alpha) \neq \text{LeftRotate}^i(\alpha)$ since $j < i$. Then if $i$ is the smallest possible index such that $a_{i+1}a_{i+2}\cdots a_n a_1a_2\cdots a_{i-1}(a_i + 1) \in N(n, k)$ and $a_1a_2\cdots a_i$ is aperidoic, then clearly $g_k(\alpha) = i = s(\alpha)$ by the definition of $g_k$. Otherwise if such an index $i < r$ does not exist, $\alpha$ contains the prefix $a_1a_2\cdots a_i$ such that $a_{i+1}a_{i+2}\cdots a_n a_1a_2\cdots a_{i-1} \in N(n, k)$ for all $i < r$. Thus $f_k^j(\alpha) = \text{LeftRotate}^i(\alpha)$ when $j < r$. Now observe that $f_k^{-1}(\alpha) = a_{r-1}a_r\cdots a_n a_1a_2\cdots a_{r-2}$ and $a_r a_{r+1}\cdots a_n a_1a_2\cdots a_{r-2} a_{r-1} \in N(n, k)$ since $\text{LeftRotate}^r(\alpha) \in N(n, k)$. Thus $f_k^j(\alpha) \neq \text{LeftRotate}^r(\alpha)$ and $g_k(\alpha) = r = s(\alpha)$.

Thus the function $g_k$ computes the minimal value $j$ such that $f_k^j(\alpha) \neq \text{LeftRotate}^i(\alpha)$.
as claimed. □

To understand how we optimize the runtime to $O(1)$-amortized time per bit, we need to consider

1. how often the optimized algorithm applies the function $f_k$, and

2. the runtime of the function $g_k$.

The following result gives us a bound on applying the function $f_k$.

**Lemma 3.7.2** The number of times the algorithm FastShift applies the function $f_k$ is bounded by $2|N(n,k)|$.

*Proof.* The number of times the algorithm FastShift applies the function $f_k$ is equal to the number of strings of the form $\alpha = a_1a_2 \cdots a_n$ such that $\alpha$ is either a necklace, or $a_1a_2 \cdots a_{n-1}(a_n + 1) \in N(n,k)$. We partition the set of strings of this form into two subsets. The first subset contains strings that are necklaces which clearly has the cardinality $|N(n,k)|$. The second subset contains strings of the form $\beta = b_1b_2 \cdots b_n$ such that $\beta \notin N(n,k)$ while $b_1b_2 \cdots b_{n-1}(b_n + 1) \in N(n,k)$. The cardinality of the second subset is clearly bounded by $|N(n,k)|$. Therefore, the number of times the algorithm FastShift applies the function $f_k$ is bounded by $2|N(n,k)|$. □

**Lemma 3.7.3** The function $g_k$ can be computed in $O(n)$ time.

*Proof.* A standard membership tester for necklaces and the value of $|ap(\alpha)|$ can be implemented and computed in $O(n)$ time. The function $q$ is based on a standard membership tester for necklaces which additionally maintains the length of the longest aperiodic prefix of $a_{j+1}a_{j+2} \cdots a_n$ such that $j/2$ is an integer and $a_1a_2 \cdots a_{\frac{j}{2} - 1} = a_{\frac{j}{2} + 1}a_{\frac{j}{2} + 2} \cdots a_{j-1}$ with $a_j \in \{a_{\frac{j}{2}}, a_{\frac{j}{2}} + 1\}$. If the string $a_{j+1}a_{j+2} \cdots a_n a_1a_2 \cdots a_{j-1}(a_j + 1)$ is a necklace, returns $j$; otherwise returns $n$. All operations of function $q$ can be computed in $O(n)$ time. Similarly function $s$ is also based on a standard membership tester for necklaces which maintains an additional variable $t$ that is the length of the longest aperiodic prefix for $a_1a_2 \cdots a_n$ such that $a_1a_2 \cdots a_t$ is aperiodic. This
value of $t$ can be easily maintained in $O(n)$ time. If $a_{t+1}a_{t+2} \cdots a_na_1a_2 \cdots a_{t-1}(a_t + 1)$ is a necklace, returns $t$, otherwise returns $r$ which is the smallest value such that \text{LeftRotate}'(\alpha) \in \mathcal{N}(n,k)$. Such a value of $r$ can be computed by Booth’s algorithm in $O(n)$ time [10]. Thus, all operations of function $s$ can also be computed in $O(n)$ time. Therefore, $g_k$ can be computed in $O(n)$ time.

We also note that $f_k$ can be optimized to run in $O(n)$ time.

**Lemma 3.7.4** The function $f_k$ can be computed in $O(n)$ time.

*Proof.* A standard membership tester for necklaces can be implemented in $O(n)$ time. Thus when $a_1 \neq k$, $f_k$ can be computed easily in $O(n)$ time. Now consider the case when $a_1 = k$. There are three subcases. If $a_2a_3 \cdots a_n = 1^{n-1}$, then clearly $b = 1$ by the definition of $f_k$. This case can be easily handled in $O(n)$ time. Now if $a_2a_3 \cdots a_n$ is not a prenecklace, then clearly $b = k$ as appending any symbol after $a_2a_3 \cdots a_n$ will not create a necklace. Otherwise if $a_2a_3 \cdots a_n$ is a prenecklace, then let $a_2a_2 \cdots a_p$ be the longest aperiodic prefix of $a_2a_3 \cdots a_n$ that is a necklace. If $n \mod p = 0$, then observe that $a_2a_3 \cdots a_na_{n-p}$ is a necklace while $a_2a_3 \cdots a_n(a_{n-p} - 1)$ is not a necklace, thus $b = a_{n-p} - 1$. Otherwise, $a_2a_3 \cdots a_n(a_{n-p} + 1)$ is a necklace while $a_2a_3 \cdots a_na_{n-p}$ is not a necklace. Thus $b = a_{n-p}$. Since the value of $p$ and the membership tester for prenecklaces can be computed and implemented in $O(n)$ time by the standard membership tester for necklace, $b$ can be computed in $O(n)$ time. Therefore, $f_k$ can be computed in $O(n)$ time.

We can thus optimize the runtime of the algorithm in the same way as the function $g$ for binary alphabets. This leads to an optimized algorithm to generate the de Bruijn sequence given in Algorithm 5.

**Theorem 3.7.5** The algorithm \text{FastShift} generates a de Bruijn sequence for $\mathbf{T}(n,k)$ in $O(1)$-amortized time per symbol.

*Proof.* By lemma 3.7.4 and lemma 3.7.3, the functions $f_k$ and $g_k$ can be computed in $O(n)$ time. Thus, it is easy to see that each iteration of the \textbf{do/while} loop requires
Algorithm 5 Optimized shift-based algorithm to generate a de Bruijn sequence for $T(n, k)$ in $O(1)$-amortized time per bit.

1: procedure FastShift
2: $a_1a_2\cdots a_n \leftarrow 1^n$
3: do
4: $j \leftarrow g_k(a_1a_2\cdots a_n)$
5: Print($a_1a_2\cdots a_j$)
6: $a_1a_2\cdots a_n \leftarrow f_k(a_ja_{j+1}\cdots a_na_1a_2\cdots a_{j-1})$
7: while $a_1a_2\cdots a_n \neq 1^n$

$O(n)$ time. By lemma 3.7.2, the number of times the function $f_k$ is applied is bounded by $2|N(n, k)|$, thus there are $O(|N(n, k)|)$ iterations of the do/while loop. Thus, the overall running time will be proportional to $O(n|N(n, k)|) = \theta(k^n)$. □

A complete C implementation is given in Appendix B.

3.8 Summary

Most of the classic constructions discussed in Chapter 2.3 take a global perspective that generate the entire de Bruijn sequences while not capable to generate the successor of an arbitrary length $n$ string in the de Bruijn sequences efficiently. The shift rule $f$ provides a whole new successor rule perspective to efficiently generate the next bit in a de Bruijn sequence over a binary alphabet (the other construction that has a successor rule is feedback shift register, but is not applicable for all values of $n$). The shift rule can also be generalized to general alphabets. The de Bruijn sequences can further be constructed in $O(1)$-amortized time per symbol, and they differ from all other de Bruijn sequences constructed by other known constructions.
Chapter 4

Generalizations of the FKM construction and greedy construction

In this chapter, we generalize the FKM and greedy constructions to construct universal cycles for a broad class of \(k\)-ary strings. Specific examples include subsets of \(k\)-ary strings of length \(n\) that contains strings (i) with sum at least \(s\); (ii) with at least \(\ell_k\) copies of \(k\); (iii) with at most \(u_i\) copies of \(i \in \{1, 2, \ldots, k - 1\}\); (iv) that circularly avoid substring \(\beta \in \{1, 2, \ldots, k - 1\}^*\); (v) that are not rotations of some periodic necklaces; (vi) that are rotations of the lexicographically largest \(i\) necklaces; and (vii) that are union or intersection of these sets. A \(k\)-suffix language \(K\) is a subset of \(k\)-ary strings of length \(n\) that satisfies a simple closure property: If \(a_1a_2\cdots a_n \in K\), then \(a_1a_2\cdots a_{n-i}k^i \in K\) for all \(0 \leq i \leq n\). We prove that the FKM and greedy constructions produce a universal cycle for a set \(S\) if \(S\) is closed under rotation, and the necklaces in \(S\) form a \(k\)-suffix language.

4.1 A broad class of \(k\)-ary strings \(C(n, k)\)

We revisit the FKM and greedy constructions discussed in Chapter 2. The FKM construction generates a de Bruijn sequence in \(O(1)\)-amortized time which can be summarized as follows:

\[\text{Concatenate the aperiodic prefixes of the necklaces in } T(n, k) \text{ in lexicographic order.}\]

We use the notation \(FKM(S)\) to denote the sequence obtained by concatenating the aperiodic prefixes of necklaces for a given set \(S\) in lexicographic order. For example,
the necklaces in $T(2, 3)$ are 11, 12, 13, 23, 33 and thus:

$$FKM(T(2, 3)) = 1 \cdot 12 \cdot 13 \cdot 2 \cdot 23 \cdot 3.$$ 

The greedy construction, on the other hand, starts with the sequence $k^{n-1}$ and extends according to the simple rule below:

Repeatedly append the smallest symbol in $\{1, 2, \ldots, k\}$ so that substrings of length $n$ in the resulting sequence are distinct and in $T(n, k)$.

We use the notation $\text{Greedy}(S)$ to denote the sequence generated by the greedy construction for a given set $S$ after removing the initial $k^{n-1}$ to seed the construction. As we have mentioned in Chapter 2, $\text{Greedy}(S) = FKM(S)$ when $S = T(n, k)$.

Nevertheless, practical applications may demand shorter sequences or universal cycles for some specific subsets of length $n$ binary strings or $k$-ary strings. For example, recall the de Bruijn card trick discussed in Chapter 1, one of the reasons the de Bruijn card trick uses 32 cards but not all the 52 cards is due to the length of de Bruijn sequences. The de Bruijn sequence for length 6 binary strings has length 64 which is greater than 52 and is not suitable for the card trick. We thus investigate generalizations of the FKM and greedy constructions to generate universal cycles for other sets.

A $k$-suffix language $K$ is a subset of $T(n, k)$ which satisfies a simple closure property:

If $a_1a_2\cdots a_n \in K$, then $a_1a_2\cdots a_{n-i}k^i \in K$ for all $0 \leq i \leq n$.

Let $C(n, k)$ be composed of all sets $S$ in $T(n, k)$ that are closed under rotation, and necklaces in $S$ are $k$-suffix languages. We generalize the FKM and greedy constructions for sets in $C(n, k)$, and thus provide universal cycles for these sets. As an example, consider the subset $X = \{1333, 2233, 2323, 2332, 3133, 3223, 3232, 3233, 3313, 3322, 3323, 3331, 3332, 3333\} \subseteq T(4, 3)$ which contains strings with sum at least 10. The set $X$ is closed under rotation, and the necklaces in $X$ are 1333, 2233, 2323, 2333, 3333 which form a $k$-suffix language. Thus:

$$FKM(X) = \text{Greedy}(X) = 1333 \cdot 2233 \cdot 23 \cdot 2333 \cdot 3$$

is a universal cycle for $X$. 

50
The universal cycle is the lexicographically smallest universal cycle for the set \( X \).

Our result implies that universal cycles for the following sets in \( C(n, k) \) can be generated by the generalized FKM and greedy constructions:

1. \( T(n, k) \);
2. \( S_1 \subseteq T(n, k) \) contains strings with sum at least \( s \);
3. \( S_2 \subseteq T(n, k) \) contains strings with at least \( \ell_k \) copies of \( k \);
4. \( S_3 \subseteq T(n, k) \) contains strings with at most \( u_i \) copies of \( i \in \{1, 2, \ldots, k - 1\} \);
5. \( S_4 \subseteq T(n, k) \) contains strings circularly avoid substring \( \beta \in \{1, 2, \ldots, k - 1\}^* \);
6. \( S_5 \subseteq T(n, k) \) contains strings that not equal to rotations of a periodic necklace \( \beta \neq k^n \); and
7. \( S_6 \subseteq T(n, k) \) contains strings that are rotations of the lexicographically largest \( i \) necklaces.

We will further prove that \( C(n, k) \) is closed under both union and intersection. The main theorem of this chapter generalizes Moreno’s results [74] on universal cycles for the set of rotations of the lexicographically largest \( i \) necklaces. By considering a special case on the last symbol of the sequence, we can generalize our result to Au’s results [5] on universal cycles for aperiodic strings.

We define the complement of a character \( x \) in the character set \( \{1, 2, \ldots, k\} \) to be equal to \( k - x + 1 \) and denote the complement of \( x \) by \( \overline{x} \). Observe that a universal cycle for the set of strings that are complement of those in a set \( S \) can be obtained by complementing every bit of a universal cycle for \( S \). As an example, consider the subset \( X = \{1111, 1112, 1113, 1121, 1122, 1131, 1211, 1212, 1221, 1311, 2111, 2112, 2121, 2211, 3111\} \subseteq T(4, 3) \) that contains strings that are the complement of the strings in \( X \). By complementing every characters in \( FKM(X) \), we obtain a universal cycle for \( \overline{X} \):

\[
FKM(X) = \overline{Greedy(X)} = 1333 \cdot 2233 \cdot 23 \cdot 2333 \cdot 3 = 3111 \cdot 2211 \cdot 21 \cdot 2111 \cdot 1.
\]

This transformation thus further allows us to construct universal cycles for the following sets:
1. $S_7 \subseteq T(n, k)$ contains strings with sum at most $s$;

2. $S_8 \subseteq T(n, k)$ contains strings with at least $\ell_1$ copies of 1;

3. $S_9 \subseteq T(n, k)$ contains strings with at most $u_i$ copies of $i \in \{2, 3, \ldots, k\}$;

4. $S_{10} \subseteq T(n, k)$ contains strings circularly avoid substring $\beta \in \{2, 3, \ldots, k\}^*$;

5. $S_{11} \subseteq T(n, k)$ contains strings that not equal to rotations of a periodic necklace $\beta \neq 1^n$; and

6. $S_{12} \subseteq T(n, k)$ contains strings that are rotations of the lexicographically smallest $i$ necklaces.

The universal cycle construction $FKM(S)$ for $S \in C(n, k)$ is remarkable for two reasons:

- the universal cycles are the lexicographically smallest universal cycles in $S$;
- they can also be constructed by a generalization of Martin’s greedy construction.

The presentation of these results in the remainder of this chapter is as follows. Chapter 4.2 introduces $k$-suffix posets, $k$-suffix necklace posets and their ideals, and discusses their correlation with $k$-suffix languages and $C(n, k)$. Chapter 4.3 proves our primary result Theorem 4.3.7. Chapter 4.4 provides a greedy interpretations for our universal cycles and proves that they are the lexicographically smallest universal cycles for the sets. Chapter 4.5 introduces some combinatorial objects in $C(n, k)$. Chapter 4.6 provides an algorithm to generate universal cycles of combinatorial objects in $C(n, k)$. Chapter 4.7 introduces an even broader language $C'(n, k)$ and prove that the generalized FKM construction also produces a universal cycle for sets in $C'(n, k)$. Chapter 4.8 concludes this chapter.

### 4.2 $k$-suffix languages and $k$-suffix posets

In this section, we provide another perspective on defining $k$-suffix languages and $C(n, k)$. A $k$-suffix language can be defined using a $k$-suffix poset and its ideals. We
then introduce a \( k \)-suffix necklace poset which is a subset of a \( k \)-suffix poset. A set in \( C(n, k) \) can be viewed as the set of rotation of an ideal of a \( k \)-suffix necklace poset.

**Definition 4.2.1** The \( k \)-suffix poset \( \text{Poset}(n, k) \) has elements in \( T(n, k) \) and satisfies the following cover relation:

\[
\alpha k^j < \alpha x k^j \text{ for all } x < k.
\]

The unique minimum element in \( \text{Poset}(n, k) \) is \( k^n \). As an example, the strings \( 2122 < 2112 \) with \( \alpha = 21, x = 1, k = 2 \) and \( j = 1 \). Figure 4.1 illustrates the Hasse diagram of \( \text{Poset}(4, 2) \).

An *ideal* \( I \) of \( \text{Poset}(n, k) \) is a subset of \( \text{Poset}(n, k) \) such that for every \( x \in I \), \( x < y \) implies \( y \in I \).

**Theorem 4.2.2** A set \( S \subseteq T(n, k) \) is a \( k \)-suffix language if and only if \( S \) is an ideal of \( \text{Poset}(n, k) \).

*Proof.* Suppose \( S \) is a \( k \)-suffix language. If \( s \in S \) and \( s \neq k^n \), then \( s = \alpha x k^j \) for some \( x < k \). By the definition of \( k \)-suffix language, \( \alpha k^j \in S \). Therefore, the unique child of \( s \) is also in \( S \) with respect to \( \text{Poset}(n, k) \), and thus \( S \) is an ideal. The other direction is similar. \( \square \)
Figure 4.2: The Hasse diagram of $N\text{Poset}(3,3)$ with an ideal in bold. The ideal is the necklaces with sum at least 6. The set of rotations of the ideal corresponds to the strings in $T(3,3)$ with sum at least 6.

**Corollary 4.2.3** If $S_A, S_B \subseteq T(n,k)$ are $k$-suffix languages, then $S_A \cup S_B$ and $S_A \cap S_B$ are also $k$-suffix languages.

The ideal in Figure 4.1 for $\text{Poset}(4,2)$ is $N(4,2)$. In general, $N(n,k)$ is an ideal of $\text{Poset}(n,k)$.

**Lemma 4.2.4** The set $N(n,k)$ is an ideal of $\text{Poset}(n,k)$.

**Proof.** Suppose $\alpha x k^j \in N(n,k)$ where $\alpha x k^j \neq k^n$ and $x < k$. By the definition of necklace, $\alpha k k^j \in N(n,k)$. The unique minimum element is $k^n$ which is also in $N(n,k)$. Thus, $N(n,k)$ is an ideal of $\text{Poset}(n,k)$. \qed

**Definition 4.2.5** The $k$-suffix necklace poset $N\text{Poset}(n,k)$ has elements in $N(n,k)$ and satisfies the following cover relation:

$$\alpha k k^j \prec \alpha x k^j \text{ for all } x < k.$$  

Thus $N\text{Poset}(n,k)$ is a subset of $\text{Poset}(n,k)$.

Let $\text{Rot}(S)$ denote the closure of $S$ under the operation rotate. For example, when $S = \{112, 113, 133\}$, $\text{Rot}(S) = \{112, 113, 121, 131, 133, 211, 311, 313, 331\}$.
Lemma 4.2.6 Let $I$ be an ideal of $\text{NPoset}(n,k)$. The set $\text{Rot}(I)$ is in $\mathcal{C}(n,k)$.

Proof. The set $\text{Rot}(I)$ is clearly closed under rotation. Furthermore, the necklaces in $\text{Rot}(I)$ is $I$ which is a $k$-suffix language by definition. Therefore $\text{Rot}(I) \in \mathcal{C}(n,k)$ by the definition of $\mathcal{C}(n,k)$. \qed

In next section, we will prove that the FKM algorithm constructs a universal cycle for $S$ if $S \in \mathcal{C}(n,k)$. Thus by Lemma 4.2.6, concatenating the elements in an ideal $I$ of $\text{NPoset}(n,k)$ in lexicographical ordering corresponds to a universal cycle of $\text{Rot}(I)$. As an example, Figure 4.2 illustrates the Hasse diagram of $\text{NPoset}(3,3)$ and $X = \{123, 132, 133, 222, 223, 233, 333\}$ is an ideal of $\text{NPoset}(3,3)$. Thus the sequence

$$123 \cdot 132 \cdot 133 \cdot 2 \cdot 223 \cdot 233 \cdot 3$$

is a universal cycle for $\text{Rot}(X)$. The set $\text{Rot}(X) \in \mathcal{C}(3,3)$ is the set of all strings in $T(3,3)$ with sum at least 6.

### 4.3 The generalized FKM construction

In this section, we prove our main theorem (Theorem 4.3.7) in this chapter which generalizes the FKM algorithm for sets in $\mathcal{C}(n,k)$. We prove that $\text{FKM}(S)$ is a universal cycle for $S$ when $S \in \mathcal{C}(n,k)$. Recall that $N(S)$ denote the necklaces of the set $S$. Let $S \in \mathcal{C}(n,k)$ where $|S| > 1$ and let $\alpha_1, \alpha_2, \ldots, \alpha_m$ be the lexicographic ordering of necklaces in $N(S)$. We first prove the following results before proving the main result of this chapter:

1. $m > 1$ and $k^n \in N(S)$,
2. there are no consecutive periodic necklaces in the lexicographic ordering of $N(S)$,
3. if $\alpha_i = a_1a_2\cdots a_{n-j-1} xk^j$ for some $x < k$ and $1 \leq i < m$, then $\alpha_{i+1}$ has prefix $a_1a_2\cdots a_{n-j-1}$,
4. $\alpha_1$ is a prefix of $\text{FKM}(S)$,

5. $xk^n$ is a suffix of $\text{FKM}(S)$ where $x$ is the maximum value less than $k$ such that $xk^{n-1} \in N(S)$, and

6. if $\alpha_i = a_1a_2 \cdots a_{n-j-1}xk^j$ for some $x < k$ and $1 \leq i < m$, then $\text{FKM}(S)$ contains the substring $a_1a_2 \cdots a_n \cdot a_1a_2 \cdots a_{|\alpha|-j-1}$.

**Lemma 4.3.1** If $S \in C(n, k)$ and $|S| > 1$, then $N(S) > 1$ and $k^n \in N(S)$.

**Proof.** Since $|S| > 1$, there exist a string $\alpha$ in $S$ such that $\alpha \neq k^n$. Since $S$ is closed under rotation, it also contains a necklace $\beta$ such that $\beta \neq k^n$ and $\beta \in \text{Rot}(\alpha)$. Also, since $N(S)$ is a $k$-suffix language, $k^n$ must be in $N(S)$. Thus $N(S) > 1$ and $k^n \in N(S)$. $\square$

**Lemma 4.3.2** Suppose $S \in C(n, k)$ and $|S| > 1$. Let $\alpha$ and $\beta$ denote consecutive necklaces in the lexicographical ordering of $N(S)$ such that $\alpha < \beta$. If $\alpha = a_1a_2 \cdots a_{n-j-1}xk^j$ for some $x < k$, then $\beta$ has prefix $a_1a_2 \cdots a_{n-j-1}$.

**Proof.** Since $\alpha$ is a necklace, clearly $a_1a_2 \cdots a_{n-j-1}yk^j$ is a necklace for all $x < y \leq k$. There exist some smallest value of $y$ such that $a_1a_2 \cdots a_{n-j-1}yk^j \in N(S)$ since $N(S)$ is a $k$-suffix language. Such a necklace clearly follows $\alpha$ in the lexicographic ordering of $N(S)$ and hence $\beta$ has prefix $a_1a_2 \cdots a_{n-j-1}$. $\square$

**Lemma 4.3.3** If $S \in C(n, k)$ and $|S| > 1$, then there are no consecutive periodic necklaces in the lexicographic ordering of $N(S)$.

**Proof.** Let $\alpha$ and $\beta$ denote consecutive necklaces in the lexicographical ordering of $N(S)$ such that $\alpha < \beta$. We know they exist by Lemma 4.3.1. Suppose $\alpha = a_1a_2 \cdots a_{n-j-1}xk^j$ is periodic for some $x < k$. Such a value of $x$ exists since $\alpha < \beta$. Since $\alpha$ is periodic, $\alpha = (\gamma xk^j)^t$ for some $\gamma$ and $t > 1$. Since $N(S)$ is a $k$-suffix language, this implies that $\beta = (\gamma xk^j)^{t-1}\gamma yk^j$ for some $y > x$. Clearly $\beta$ is aperiodic because $\gamma yk^j > \gamma xk^j$. $\square$
Lemma 4.3.4 If $S \in \mathcal{C}(n, k)$ and $|S| > 1$, then the lexicographically smallest necklace in $N(S)$ is a prefix of $FKM(S)$.

Proof. Let $\alpha = a_1a_2\cdots a_{n-j-1}xk^j$ be the lexicographically smallest necklace in $N(S)$ for some $x < k$. Such a value of $x$ exists since $\alpha \neq k^n$ by Lemma 4.3.1. If $\alpha$ is aperiodic, then clearly $FKM(S)$ has prefix $\alpha$. Otherwise if $\alpha$ is periodic, then $\alpha = (ap(\alpha))^t$ for some $t > 1$. Let $\beta$ be the necklace that is after $\alpha$ in the lexicographic ordering of $N(S)$. Such a necklace exists since $|N(S)| > 1$ by Lemma 4.3.1. By Lemma 4.3.3, $\beta$ is aperiodic. Also by Lemma 4.3.2, $\beta$ has prefix $a_1a_2\cdots a_{n-j-1} = (ap(\alpha))^{t-1}a_{(t-1)|ap(\alpha)|+1}a_{(t-1)|ap(\alpha)|+2}\cdots a_{n-j-1}$. Therefore, $ap(\alpha) \cdot ap(\beta)$ has prefix $ap(\alpha) \cdot (ap(\alpha))^{t-1} = \alpha$. □

Lemma 4.3.5 If $S \in \mathcal{C}(n, k)$ and $|S| > 1$, then $FKM(S)$ has suffix $xk^n$, where $x$ is the maximum value such that $x < k$ and $xk^{n-1} \in N(S)$.

Proof. Let $\alpha$ and $\beta$ denote the two lexicographically largest necklace in $N(S)$ such that $\alpha < \beta$. By Lemma 4.3.1, $\beta = k^n$ and $\alpha$ exists since $|N(S)| > 1$. There exist a maximum value $x < k$ such that $xk^{n-1} \in N(S)$ since $N(S)$ is a $k$-suffix language. Observe that $\alpha = xk^{n-1}$ since there can be no necklace between it and $\beta$ in the lexicographical ordering of $N(S)$. Therefore, $FKM(S)$ has suffix $ap(\alpha) \cdot ap(\beta) = xk^{n-1} \cdot k = xk^n$. □

Lemma 4.3.6 Suppose $S \in \mathcal{C}(n, k)$ and $|S| > 1$. If $\alpha = a_1a_2\cdots a_{n-j-1}xk^j \in N(S)$ for some $x < k$, then $FKM(S)$ contains the substring $a_1a_2\cdots a_n \cdot a_1a_2\cdots a_{|ap(\alpha)|-j}$. 

Proof. Let $p = |ap(\alpha)|$ and let $\beta$ denote the necklace that is after $\alpha$ in the lexicographic ordering $N(S)$. Clearly $\alpha \neq k^n$ and $\beta$ exists by Lemma 4.3.1. By Lemma 4.3.2 $\beta$ has prefix $a_1a_2\cdots a_{n-j-1}$. By Lemma 4.3.3 at most one of these necklaces is periodic and we proceed in three cases.

1. If both $\alpha$ and $\beta$ are aperiodic, then $ap(\alpha) \cdot ap(\beta) = \alpha \cdot \beta$ is a substring of $FKM(S)$ which has prefix $a_1a_2\cdots a_n \cdot a_1a_2\cdots a_{n-j-1} = a_1a_2\cdots a_n \cdot a_1a_2\cdots a_{p-j-1}$.  

57
2. If $\alpha$ is periodic and $\beta$ is aperiodic, then $ap(\alpha) \cdot ap(\beta) = ap(\alpha) \cdot \beta$ is a substring of $FKM(S)$ which has prefix $ap(\alpha) \cdot a_{1}a_{2} \cdots a_{n-j-1}$. Let $\alpha = (ap(\alpha))^t$ for some $t > 1$. Observe that

$$ap(\alpha) \cdot a_{1}a_{2} \cdots a_{n-j-1} = ap(\alpha) \cdot (ap(\alpha))^{t-1}a_{(t-1)p+1}a_{(t-1)p+2} \cdots a_{n-j-1}$$

$$= \alpha \cdot a_{(t-1)p+1}a_{(t-1)p+2} \cdots a_{n-j-1}$$

$$= a_{1}a_{2} \cdots a_{n} \cdot a_{1}a_{2} \cdots a_{p-j-1}.$$  

3. If $\alpha$ is aperiodic and $\beta$ is periodic, then there are two subcases. If $\beta = k^n$, then the substring $a_{1}a_{2} \cdots a_{n} \cdot a_{1}a_{2} \cdots a_{p-j-1}$ is simply equal to $a_{1}a_{2} \cdots a_{n}$ due to the fact that $\alpha$ has suffix $k^{n-1}$. The desired substring is found in the length $n + 1$ suffix of $FKM(S)$ by Lemma 4.3.5. Otherwise if $\beta \neq k^n$, then let $\gamma$ be the necklace that is after $\beta$ in the lexicographic ordering of $N(S)$. Such a necklace exists since $\beta \neq k^n$. Notice that $\gamma$ is aperiodic by Lemma 4.3.3. Therefore, by the arguments in the previous case, $ap(\beta) \cdot ap(\gamma)$ has prefix $\beta$. Therefore, $ap(\alpha) \cdot ap(\beta) \cdot ap(\gamma)$ is a substring of $FKM(S)$ which has prefix $\alpha \cdot \beta$. The prefix $\alpha \cdot \beta$ contains the prefix $a_{1}a_{2} \cdots a_{n} \cdot a_{1}a_{2} \cdots a_{p-j-1}$.

Thus $FKM(S)$ contains the substring $a_{1}a_{2} \cdots a_{n} \cdot a_{1}a_{2} \cdots a_{|ap(\alpha)|-j-1}$.

We now proceed to the proof of the main theorem in this chapter.

**Theorem 4.3.7** If $S \in C(n, k)$, then $FKM(S)$ is a universal cycle for $S$.

*Proof.* Since $S$ is closed under rotation by the definition of $C(n, k)$ and its strings all have length $n$, the definition of $FKM(S)$ implies $|FKM(S)| = |S|$. Therefore, to prove $FKM(S)$ is a universal cycle for $S$, we only need to show that $FKM(S)$ contains each string in $S$ as a substring when the sequence is considered circularly.

When $|N(S)| = 1$, $S = N(S) = \{k^n\}$. In this case, $FKM(S)$ is the single character $k$ which is a universal cycle for $S$. For the remainder of the proof we assume $|N(S)| > 1$.

Now consider a rotation $\beta = a_{i}a_{i+1} \cdots a_{n}a_{1}a_{2} \cdots a_{i-1}$ of an arbitrary necklace $\alpha = a_{1}a_{2} \cdots a_{n}$ in $N(S)$. By Lemma 4.3.5 we can assume that $\alpha \neq k^n$ since the only
rotation of $k^n$ is found at the end of $FKM(S)$. Therefore, without loss of generality, we suppose $\alpha$ has suffix $xk^j$ for some $x < k$. Let $p = |ap(\alpha)|$. We show that all $p$ distinct rotations of $\alpha$ exist in $FKM(S)$. There are two cases depending on the value of $i$.

**Case 1:** $0 < i \leq p - j - 1$: By Lemma 4.3.6, $a_1a_2 \cdots a_n \cdot a_1a_2 \cdots a_{p-j-1}$ is a substring of $FKM(S)$. Observe that $\beta$ is a substring of $a_1a_2 \cdots a_n \cdot a_1a_2 \cdots a_{p-j-1}$ when $0 < i \leq p - j - 1$.

**Case 2:** $p - j - 1 < i \leq p$: Observe that $\beta = k^{n-i}a_1a_2 \cdots a_{i-1}$ when $p - j - 1 < i \leq p$.

Let $\gamma$ be the lexicographically smallest necklace in $N(S)$ such that it has prefix $a_1a_2 \cdots a_{i-1}$. If $\gamma$ is not the lexicographically smallest necklace in $N(S)$, then the previous necklace of $\gamma$ has the suffix $k^{n-i}$ due to the fact that $N(S)$ is a $k$-suffix language. On the other hand, if $\gamma$ is the lexicographically smallest necklace in $N(S)$, then the previous $n - i$ symbols in $FKM(S)$ are $k^{n-i}$ by Lemma 4.3.5 when the sequence is considered circularly. Thus, $\beta$ is a substring of $FKM(S)$.

Therefore, $FKM(S)$ contains each string in $S$ as a substring and is a universal cycle for $S$ since $|FKM(S)| = |S|$.

In addition to sets in $C(n,k)$, $FKM(S)$ can also construct universal cycles for sets $S$ that are “almost” in $C(n,k)$.

**Corollary 4.3.8** If $S \cup \{k^n\} \in C(n,k)$, then $FKM(S)$ is a universal cycle for $S$.

**Proof.** If $S \in C(n,k)$, then by Theorem 4.3.7 $FKM(S)$ is a universal cycle for $S$. Otherwise if $S \notin C(n,k)$ but $S \cup \{k^n\} \in C(n,k)$, then observe that $FKM(S) \cdot k = FKM(S \cup k^n)$. By Theorem 4.3.7 $FKM(S) \cdot k$ is a universal cycle for $S \cup k^n$. By Lemma 4.3.5, $FKM(S) \cdot k$ ends with the suffix $k^n$. Hence, removing the last $k$ in $FKM(S) \cdot k$ only removes the string $k^n$ while preserves all strings in $S \cup \{k^n\} \setminus \{k^n\} = S$ as a substring when considered circularly. Therefore, $FKM(S)$ is a universal cycle for $S$ since $FKM(S)$ has length $|S|$. \qed
The proof of Theorem 4.3.7 explicitly states where the rotations of each necklace are found. To specify the position of a substring in a universal cycle we introduce the following notation. If $FKM(S) = u_0u_1 \cdots u_{|S|-1}$ is a universal cycle for $S \subseteq T(n,k)$ and $\alpha \in S$, then let $last(\alpha)$ be the last position of the substring $\alpha$ in $FKM(S)$. In other words, if $last(\alpha) = i$, then $u_{i-n+1}u_{i-n+2} \cdots u_i = \alpha$ where $0 \leq i < |S|$ and the other index expressions are taken modulo $|S|$.

**Corollary 4.3.9** If $S \in C(n,k)$, $\alpha \in N(S)$ has suffix $xk^j$ for some $x < k$, and $\beta$ is a rotation of $\alpha$, then $last(\beta) \leq last(\alpha) + |ap(\alpha)| - j - 1$.

**Proof.** Let $\alpha = a_1a_2 \cdots a_n$. By Lemma 4.3.6, the rotations of $\alpha$ starting from $a_i$ where $1 \leq i \leq |ap(\alpha)| - j$ are all found in succession starting from $\alpha$ itself. In particular, the last of these rotations $\beta$ has $last(\beta) = last(\alpha) + |ap(\alpha)| - j - 1$. The remaining rotations of $\alpha$ end within the first necklace in the lexicographic ordering of $N(S)$ with prefix $a_1a_2 \cdots a_i$ for some $i$ satisfying $1 \leq i \leq |ap(\alpha)| - j - 1$. None of these necklaces appear after $\alpha$ in the lexicographic ordering of $N(S)$, thus $last(\beta) < last(\alpha)$ for any such rotation of $\beta$. \hfill \Box

This information will be crucial for the greedy algorithm in Chapter 4.4.

### 4.4 Greedy approach

In this section, we prove that $Greedy(S) = FKM(S)$ when $S \in C(n,k)$. We further prove that both $Greedy(S)$ and $FKM(S)$ produce the lexicographically smallest universal cycle for $S$ when $S \in C(n,k)$.

**Lemma 4.4.1** If $S \in C(n,k)$ and $|S| > 1$, then the lexicographically smallest necklace in $N(S)$ is a prefix of $Greedy(S)$.

**Proof.** By contradiction. Let $\alpha = a_1a_2 \cdots a_n$ be the lexicographically smallest necklace in $N(S)$. The string $\alpha$ is also the lexicographically smallest string in $S$ since $S$ is closed under rotation. Suppose $Greedy(S)$ starts with prefix $a_1a_2 \cdots a_i z$ for some $i < n$ and $z \neq a_{i+1}$. Observe that $z < a_{i+1}$ by the definition of the greedy algorithm. Furthermore, since the greedy algorithm starts with the initial seed $k^{n-1}$,
after \(i + 1\) iterations of the greedy algorithm the length \(n\) suffix of the sequence is \(\beta = k^{n-i}a_1a_2\cdots a_{i}z\). Notice that if \(\beta \in S\), then its rotation \(a_1a_2\cdots a_{i}zk^{n-i-1}\) must also be in \(S\) since \(S\) is closed under rotation. However, \(a_1a_2\cdots a_{i}zk^{n-i-1}\) is strictly less than \(\alpha\), a contradiction to \(\alpha\) being the lexicographically smallest string in \(S\). \(\square\)

**Theorem 4.4.2** If \(S \in C(n,k)\), then \(\text{Greedy}(S)\) is equivalent to \(\text{FKM}(S)\).

**Proof.** Let \(\alpha_1, \alpha_2, \ldots, \alpha_m\) denote the lexicographic ordering of necklaces in \(N(S)\). Let \(L_t = \text{ap}(\alpha_1)\text{ap}(\alpha_2)\cdots \text{ap}(\alpha_t)\) for \(1 \leq t \leq m\). When \(m = 1\), \(S = \{k^n\}\) and the greedy algorithm terminates with the correct sequence of length one, namely \(\text{Greedy}(S) = \text{FKM}(S) = k\).

For \(m > 1\), we prove that \(\text{Greedy}(S) = \text{FKM}(S) = L_m\) by contradiction. Suppose \(t\) is the smallest value such that \(L_{t+1}\) is not a prefix of \(\text{Greedy}(S)\), where \(\alpha_t = a_1a_2\cdots a_n = a_1a_2\cdots a_{n-j}k^j\) and \(a_{n-j} < k\). From Lemma 4.4.1 we know that \(1 \leq t < m\). Let \(\alpha_{t+1} = b_1b_2\cdots b_n\) and \(p = |\text{ap}(\alpha_{t+1})|\). Let \(i\) be the smallest value such that \(0 < i \leq p\) and \(L_t \cdot b_1b_2\cdots b_i\) is not a prefix of \(\text{Greedy}(S)\). Let \(\beta\) denote the length \(n - 1\) suffix of \(L_t \cdot b_1b_2\cdots b_{i-1}\). Such a suffix exists since both \(\text{Greedy}(S)\) and \(\text{FKM}(S)\) begin with \(\alpha_1\) by Lemma 4.3.4 and Lemma 4.4.1 when \(m > 1\). There are two cases depending on the value of \(i\).

**Case 1:** \(0 < i \leq p - j - 1\): By Lemma 4.3.2, \(b_1b_2\cdots b_{n-j-1} = a_1a_2\cdots a_{n-j-1}\). Therefore by Lemma 4.3.6, \(\beta = a_{i+1}a_{i+2}\cdots a_n \cdot a_1a_2\cdots a_{i-1}\) since \(\text{ap}(\alpha_t)\) is prior to \(b_1b_2\cdots b_{i-1}\). Since \(L_t \cdot b_1b_2\cdots b_i\) is not a prefix of \(\text{Greedy}(S)\), the greedy algorithm appends \(z\) to \(\beta\) where \(z < b_i\). By Corollary 4.3.9, \(\beta z\) is not a rotation of the necklaces \(\alpha_1, \alpha_2, \ldots, \alpha_{t-1}\). However, a rotation of \(\beta z\) is equal to \(a_1a_2\cdots a_{i-1}za_{i+1}a_{i+2}\cdots a_n\) and is strictly less than \(\alpha_t\). Therefore, \(\beta z\) is a rotation of some necklace that is between \(\alpha_{t-1}\) and \(\alpha_t\) in lexicographic order, a contradiction to \(\alpha_t\) being the necklace after \(\alpha_{t-1}\) in the lexicographic ordering of \(N(S)\). Thus \(z\) must be equal to \(b_i = a_i\).

**Case 2:** \(p - j - 1 < i \leq p\): By Lemma 4.3.6, \(\beta = a_{i+1}a_{i+2}\cdots a_nb_1b_2\cdots b_{i-1}\) which is equal to \(a_{i+1}a_{i+2}\cdots a_n a_1a_2\cdots a_{p-j-1} \cdot b_{p-j}b_{p-j+1}\cdots b_{i-1}\) since \(\text{ap}(\alpha_t)\) is prior.
to \( b_1 b_2 \cdots b_{i-1} \). Since \( L_t \cdot b_1 b_2 \cdots b_i \) is not a prefix of Greedy(S), the greedy algorithm appends \( z \) to \( \beta \) where \( z < b_i \). By Corollary 4.3.9, \( \beta z \) is not a rotation of the necklaces \( \alpha_1, \alpha_2, \ldots, \alpha_t \). However, a rotation of \( \beta z \) is equal to \( a_1 a_2 \cdots a_{p-j-1} \cdot b_p b_{p-j+1} \cdots b_i z a_{i+1} a_{i+2} \cdots a_n \) and is strictly less than \( \alpha_{t+1} \). Therefore, \( \beta z \) is a rotation of some necklace that is between \( \alpha_t \) and \( \alpha_{t+1} \) in lexicographic order, a contradiction to \( \alpha_{t+1} \) being the necklace after \( \alpha_t \) in the lexicographic ordering of \( N(S) \). Thus \( z \) must be equal to \( b_i \).

Thus by proof of contradiction, Greedy(S) = FKM(S) = \( \mathcal{L}_m \) as claimed. \( \square \)

**Corollary 4.4.3** \( \text{FKM}(S) \) and \( \text{Greedy}(S) \) produce the lexicographically smallest universal cycle among all universal cycles for \( S \in \mathcal{C}(n, k) \).

**Proof.** The greedy algorithm always starts with the lexicographically smallest necklace in \( N(S) \) by Lemma 4.4.1, which is also the lexicographically smallest string in \( S \). It then greedily appends the lexicographically smallest possible symbol such that the length \( n \) suffix is unique and in \( S \). By the definition of the greedy algorithm, appending any smaller symbol results in either a duplicate string, or a string not in \( S \). Thus Greedy(S) must be the lexicographically smallest universal cycle. Also by Theorem 4.4.2, FKM(S) and Greedy(S) produce the same universal cycle. Therefore, both FKM(S) and Greedy(S) produce the lexicographically smallest universal cycle for \( S \). \( \square \)

### 4.5 Combinatorial objects in \( \mathcal{C}(n, k) \)

This section introduces combinatorial objects that can be naturally represented in the language \( \mathcal{C}(n, k) \). In below we show that the sets \( T(n, k) \), and \( S_1, S_2, S_3, S_4, S_5 \) and \( S_6 \) we defined in Chapter 4.1 are in \( \mathcal{C}(n, k) \). Recall that \( S \in \mathcal{C}(n, k) \) if \( S \) is closed under rotation and \( N(S) \) is a \( k \)-suffix language. Observe that these sets are trivially closed under rotation. Thus we only need to show the necklace sets for \( T(n, k) \), and \( S_1, S_2, S_3, S_4, S_5 \) and \( S_6 \) are \( k \)-suffix languages.
The idea of many of our proofs below is to first show a set \( S \) is a \( k \)-suffix language. Then by Corollary 4.2.3 and Lemma 4.2.4, we can claim that \( N(S) = S \cap N(n, k) \) is a \( k \)-suffix language when \( S \) is a \( k \)-suffix language. Thus \( S \in \mathcal{C}(n, k) \) since we already know \( S \) is closed under rotation. This argument is used to prove the necklace sets of \( T(n, k), S_1, S_2, S_3 \) and \( S_4 \) are \( k \)-suffix languages.

\( T(n, k) \): Consider a string \( \alpha x^j k^j \in T(n, k) \) where \( x < k \) and \( 0 \leq j < n \), clearly \( \alpha k^j \in T(n, k) \). Thus \( T(n, k) \) and \( N(n, k) \) are \( k \)-suffix languages. Therefore \( T(n, k) \in \mathcal{C}(n, k) \).

\textbf{Sums}: Consider a string \( \alpha x^j k^j \in S_1 \) with sum \( w \geq s \) where \( x < k \) and \( 0 \leq j < n \), \( \alpha k^j \in S_1 \) since \( \alpha k^j \) has sum \( w + k - x \geq w \geq s \). Thus \( S_1 \) and \( N(S_1) = S_1 \cap N(n, k) \) are \( k \)-suffix languages. Therefore \( S_1 \in \mathcal{C}(n, k) \).

\textbf{Frequencies}: Consider a string \( \alpha x^j k^j \in S_2 \) with \( \ell \geq \ell_k \) copies of \( k \) where \( x < k \) and \( 0 \leq j < n \), \( \alpha k^j \in S_2 \) since \( \alpha k^j \) has \( \ell + 1 > \ell_k \) copies of \( k \). Thus \( S_2 \) and \( N(S_2) = S_2 \cap N(n, k) \) are \( k \)-suffix languages. Therefore \( S_2 \in \mathcal{C}(n, k) \).

Similarly, consider a string \( \alpha x^j k^j \in S_3 \) with \( u \leq u_i \) copies of \( i \) where \( x < k \) and \( 0 \leq j < n \), \( \alpha k^j \) has \( u - 1 < u_i \) copies of \( i \) when \( i = x \), and \( u \leq u_i \) copies of \( i \) when \( i \neq x \). Hence \( \alpha k^j \in S_3 \). Thus \( S_3 \) and \( N(S_3) = S_3 \cap N(n, k) \) are \( k \)-suffix languages. Therefore \( S_3 \in \mathcal{C}(n, k) \).

\textbf{Substring avoidance}: Consider a string \( \alpha x^j k^j \in S_4 \) that circularly avoids \( \beta \in \{1, 2, \ldots, k - 1\}^* \) where \( x < k \) and \( 0 \leq j < n \), \( \alpha k^j \in S_4 \) since \( \alpha k^j \) also circularly avoids \( \beta \) because \( \beta \) does not contain the symbol \( k \). Thus \( S_4 \) and \( N(S_4) = S_4 \cap N(n, k) \) are \( k \)-suffix languages. Therefore \( S_4 \in \mathcal{C}(n, k) \).

\textbf{Periodic string avoidance}: By contradiction, we show that \( N(S_5) \) is a \( k \)-suffix language. Assume \( \alpha x^j k^j \in N(S_5) \) is an aperiodic necklace where \( x < k \) and \( 0 \leq j < n \), and \( \alpha k^j = \beta \neq k^n \). Clearly \( \alpha k^j \) is a necklace because \( \alpha x^j k^j \) is a necklace. Since \( \beta \) is periodic and \( \beta \neq k^n \), \( \beta = (\gamma k^{j+1})^t \) where \( \gamma \neq 0 \) and \( t > 1 \). Observe that \( \alpha x^j = (\gamma k^{j+1})^{t-1} \gamma x^j k^j \) is not a necklace since \( x < k \), a
contradiction to $\alpha x k^j$ being a necklace. Thus $\alpha k k^j \neq \beta$ and $N(S_6)$ is a $k$-suffix language. Therefore $S_6 \in C(n, k)$.

**Rotations of the lexicographically largest $i$ necklaces:** Let $\alpha x k^j \in N(S_6)$ be one of the lexicographically largest $i$ necklace in $N(n, k)$ where $x < k$ and $0 \leq j < n$. Clearly $\alpha k k^j \in N(S_6)$ since it is also in $N(n, k)$ and is lexicographically larger than $\alpha x k^j$. Thus $N(S_6)$ is a $k$-suffix language and $S_6 \in C(n, k)$.

Notice that the intersection and union of these sets are closed under rotation. Furthermore, by Corollary 4.2.3 and Lemma 4.2.4, the necklace sets of the intersection and union of these sets are also $k$-suffix languages. Thus, the intersection and union of combinatorial objects in $C(n, k)$ are also in $C(n, k)$. For example, the following length 4 strings:

$$1233, 2331, 3313, 3132, 1323, 3231, 2313,$$
$$3133, 1332, 3321, 3213, 2133, 1333, 3332,$$
$$3323, 3233, 2333, 3333, 3331, 3312, 3123,$$

are precisely the subset of $T(4, 3)$ that contains either strings with sum $\geq 11$, or have at most one copy of 2 and at most one copy of 1. The set is in $C(4, 3)$.

Now revisit $S_5$ that contains all strings that are not equal to rotations of a periodic necklace $\beta \neq k^n$. Let $S_{13}$ denote the set that contains all strings in $T(n, k)$ not equal to any periodic necklace. Observe that $S_{13} \cup \{k^n\} \in C(n, k)$ since $S_5 \in C(n, k)$. Thus by considering the intersection and union of sets that avoid different periodic necklaces, we can generalize our results to Au’s results in [5].

**Lemma 4.5.1** If $N$ is a set that consists of periodic necklaces in $N(n, k)$ and $S = Rot(N(n, k) \setminus N)$, then FKM($S$) and Greedy($S$) produce a universal cycle for $S$.

### 4.6 The generalized FKM algorithm

In [14], Cattell et al. present a recursive necklace generation framework to generate prenecklaces, Lyndon words and necklaces of $T(n, k)$ in $O(1)$-amortized time per
Algorithm 6 The generalized FKM construction for universal cycles of $S \in \mathcal{C}(n, k)$.

1: procedure FKM($t, p$)
2:     if $t > n$ then
3:         if $n \mod p = 0$ then PRINT($a_1a_2\cdots a_p$)
4:     else
5:         for $a_t$ from $a_{t-p}$ to $k$ do
6:             if $a_{t-p} = a_t$ and $a_1a_2\cdots a_t \in P_A(S)$ then FKM($t+1, p$)
7:             else if $a_{t-p} < a_t$ and $a_1a_2\cdots a_t \in P_A(S)$ then FKM($t+1, t$)

symbol using $O(n)$ space. The basic idea is to recursively extend a prenecklace $\alpha = a_1a_2\cdots a_{t-1}$ to a length $t$ prenecklace in all possible ways. This is done efficiently by maintaining a variable $p$ which is the length of the longest prefix of $\alpha$ that is a Lyndon word. This algorithm can easily be adapted to construct $FKM(S)$ by concatenating the aperiodic prefixes of necklaces in $S \in \mathcal{C}(n, k)$. Let $P(S)$ denote the set of prenecklaces in $S$, and $P_A(S)$ denote the set of prenecklaces with length from 1 to $n$ that can be extended to a necklace in $S$. Pseudocode of the necklace concatenation algorithm is given in Algorithm 6.

Theorem 4.6.1 The algorithm FKM generates universal cycles for $S \in \mathcal{C}(n, k)$ in $O(1)$-amortized time per symbol using $O(n)$ space if

1. $n|P(S)|$ is directly proportional to $|S|$; and

2. there exist a membership tester for $P_A(S)$ that can be completed in $O(1)$-amortized time using $O(n)$ space.

Proof. The algorithm FKM generates each length $n$ prenecklace in $P(S)$. Since there is no dead end in the algorithm, the number of recursive calls is bounded by $n|P(S)|$. The runtime for each recursive call depends on the membership tester for $P_A(S)$. The algorithm thus generates a universal cycle for $S$ in $O(1)$-amortized time per symbol using $O(n)$ space if it satisfies the two conditions.

For example, the following sets $S$ have their membership testers for $P_A(S)$ run in $O(1)$-amortized time, and $n|P(S)|$ directly proportional to $|S|$ [83, 84, 92]:

1. $T(n, k)$;
2. subsets of $T(n, 2)$ with sum at least $s$;

3. subsets of $T(n, 2)$ with at least $\ell_k$ copies of $k$;

4. subsets of $T(n, 2)$ with at most $u_i$ copies of $i \in \{1, 2, \ldots, k - 1\}$;

5. subsets of $T(n, k)$ that circularly avoid substring $\beta \in \{1, 2, \ldots, k - 1\}^*$;

6. subsets of $T(n, k)$ that not equal to rotations of periodic necklace $\beta$; and

7. the union and intersection of any of the above sets.

Thus their universal cycles can be generated in $O(1)$-amortized time per symbol using $O(n)$ space. A complete C implementation of the generalized FKM construction for $T(n, 2)$ with sum at least $s$ is given in Appendix C. The efficient implementations of the other sets mentioned in this section are similar.

### 4.7 An even broader language $C'(n, k)$

The FKM construction can be further generalized to a broader language. Let $C'(n, k)$ be composed of all sets $S$ in $T(n, k)$ that are closed under rotation, and satisfies the following closure property:

If a string $a_1a_2\cdots a_{n-j}k^j \in S$ is a necklace but not the lexicographically largest necklace, then $a_1a_2\cdots a_{n-j-1}kk^j \in S$.

In this chapter, we prove that the FKM construction is also applicable for sets in $C'(n, k)$. As an example, consider the set $X = \{1212, 1213, 1223, 1312, 2112, 2131, 2231, 2312, 2331, 3121, 3122, 3123, 3312\} \subseteq T(4, 3)$. The set $X$ is closed under rotation, but $X \notin C(3, 3)$ because its necklace set $\{1212, 1213, 1223, 1233\}$ is not a $k$-suffix language. However, the set $X \in C'(n, k)$ since its necklace set $\{1212, 1213, 1223, 1233\}$ satisfies the closure property in $C'(n, k)$. Thus:

$$FKM(X) = 12 \cdot 1213 \cdot 1223 \cdot 1233 \cdot 133$$

is a universal cycle for $X$.

Observe that the language $C'(n, k)$ indeed includes sets in $C(n, k)$, thus we say $C'(n, k)$ is a broader language than $C(n, k)$.
Let $S \in C'(n,k)$ where $|N(S)| > 1$ and let $\alpha_1, \alpha_2, \ldots, \alpha_m$ be the lexicographic ordering of necklaces in $N(S)$. Before we prove that the FKM construction produces a universal cycle for sets in $C'(n,k)$ (Theorem 4.7.8), we first prove the following results:

1. if $\alpha_i = a_1a_2 \cdots a_{n-j-1} xk^j$ for some $x < k$ and $1 \leq i < m$, then $\alpha_{i+1}$ has prefix $a_1a_2 \cdots a_{n-j}$,

2. there are no consecutive periodic necklaces in the lexicographic ordering of $N(S)$,

3. $\alpha_1$ is a prefix of $\text{FKM}(S)$,

4. if $\alpha_m \neq k^n$, then $\alpha_m$ is aperiodic,

5. $\alpha_m$ is a suffix of $\text{FKM}(S)$,

6. if $\alpha_m = a_1a_2 \cdots a_{n-j-1} xk^j$, then $\alpha_i$ has the prefix $a_1a_2 \cdots a_{n-j-1}$ for $1 \leq i \leq m$.

7. if $\alpha_i = a_1a_2 \cdots a_{n-j-1} xk^j$ for some $x < k$ and $1 \leq i < m$, then $\text{FKM}(S)$ contains the substring $a_1a_2 \cdots a_n \cdot a_1a_2 \cdots a_{|\text{ap}(\alpha)|-j-1}$. 

**Lemma 4.7.1** Suppose $S \in C'(n,k)$ and $|N(S)| > 1$. Let $\alpha$ and $\beta$ denote consecutive necklaces in the lexicographical ordering of $N(S)$ such that $\alpha < \beta$. If $\alpha = a_1a_2 \cdots a_{n-j-1} xk^j$ for some $x < k$, then $\beta$ has prefix $a_1a_2 \cdots a_{n-j-1}$.

*Proof.* Since $\alpha$ is a necklace, clearly $a_1a_2 \cdots a_{n-j-1} yk^j$ is a necklace for all $x < y \leq k$. There exist some smallest value of $y$ such that $a_1a_2 \cdots a_{n-j-1} yk^j \in N(S)$ since $\alpha$ is not the lexicographically largest necklace in $N(S)$ and $S \in C'(n,k)$. Such a necklace clearly follows $\alpha$ in the lexicographic ordering of $N(S)$ and hence $\beta$ has prefix $a_1a_2 \cdots a_{n-j-1}$. 

**Lemma 4.7.2** If $S \in C'(n,k)$ and $|N(S)| > 1$, then there are no consecutive periodic necklaces in the lexicographic ordering of $N(S)$. 

67
Proof. Let $\alpha$ and $\beta$ denote consecutive necklaces in the lexicographical ordering of $\mathbb{N}(\mathcal{S})$ such that $\alpha < \beta$. Suppose $\alpha = a_1a_2 \cdots a_{n-j-1}xk^j$ is periodic for some $x < k$. Such a value of $x$ exists since $\alpha \neq k^n$ and $\alpha < \beta$. Since $\alpha$ is periodic, $\alpha = (\gamma xk^j)^t$ for some $\gamma$ and $t > 1$. Since $\alpha$ is not the lexicographically largest necklace in $\mathbb{N}(\mathcal{S})$ and $\mathcal{S} \in \mathcal{C}'(n, k)$, this implies that $\beta = (\gamma xk^j)^{t-1}\gamma yk^j$ for some $y > x$. Clearly $\beta$ is aperiodic because $\gamma yk^j > \gamma xk^j$.

Lemma 4.7.3 If $\mathcal{S} \in \mathcal{C}'(n, k)$ and $|\mathbb{N}(\mathcal{S})| > 1$, then the lexicographically smallest necklace in $\mathbb{N}(\mathcal{S})$ is a prefix of $\text{FKM}(\mathcal{S})$.

Proof. Let $\alpha = a_1a_2 \cdots a_{n-j-1}xk^j$ be the lexicographically smallest necklace in $\mathbb{N}(\mathcal{S})$ for some $x < k$. Such a value of $x$ exists since $\alpha \neq k^n$ as $\alpha$ is not the lexicographically largest necklace in $\mathbb{N}(\mathcal{S})$. If $\alpha$ is aperiodic, then clearly $\text{FKM}(\mathcal{S})$ has prefix $\alpha$. Otherwise if $\alpha$ is periodic, then $\alpha = (\text{ap}(\alpha))^t$ for some $t > 1$. Let $\beta$ be the necklace that is after $\alpha$ in the lexicographic ordering of $\mathbb{N}(\mathcal{S})$. Such a necklace exists since $|\mathbb{N}(\mathcal{S})| > 1$. By Lemma 4.7.2, $\beta$ is aperiodic. Also by Lemma 4.7.1, $\beta$ has prefix $a_1a_2 \cdots a_{n-j-1} = (\text{ap}(\alpha))^{t-1}a_{(t-1)}\text{ap}(\alpha)+\cdots a_{(t-1)}\text{ap}(\alpha)+2 \cdots a_{n-j-1}$. Therefore, $\text{ap}(\alpha) \cdot \text{ap}(\beta)$ has prefix $\text{ap}(\alpha) \cdot (\text{ap}(\alpha))^{t-1} = \alpha$. □

Lemma 4.7.4 Suppose $\mathcal{S} \in \mathcal{C}'(n, k)$ and $|\mathbb{N}(\mathcal{S})| > 1$. Let $\beta$ denote the lexicographically largest necklace in $\mathbb{N}(\mathcal{S})$. If $\beta \neq k^n$, then $\beta$ is aperiodic.

Proof. By contradiction, suppose $\beta$ is periodic and $\beta \neq k^n$. Since $\beta$ is periodic, $\beta = (\gamma xk^j)^t$ for some $\gamma$ and $t > 1$. By the definition of $\mathcal{C}'(n, k)$, there exist a necklace $\alpha = (\gamma xk^j)^{t-1}(\gamma yk^j)^{-1} \in \mathbb{N}(\mathcal{S})$ that is lexicographically smaller than $\beta$ for some $y < k$. However, observe that $\alpha$ is not a necklace since $(\gamma xk^j)^{t-1} < (\gamma xk^j)^{-1}$, a contradiction to $\alpha$ being a necklace. Thus $\beta$ must be aperiodic. □

Lemma 4.7.5 If $\mathcal{S} \in \mathcal{C}'(n, k)$ and $|\mathbb{N}(\mathcal{S})| > 1$, then the lexicographically largest necklace in $\mathbb{N}(\mathcal{S})$ is a suffix of $\text{FKM}(\mathcal{S})$. 68
Proof. Let $\beta$ denote the lexicographically largest necklace in $N(S)$. If $\beta = k^n$, then $S \in \mathcal{C}(n, k)$ and by Lemma 4.3.5 $FKM(S)$ has suffix $xk^n$ which ends with $\beta = k^n$. Otherwise if $\beta \neq k^n$, by Lemma 4.7.1 $\beta$ is aperiodic and thus clearly $FKM(S)$ has suffix $\beta$. □

Lemma 4.7.6 Suppose $S \in \mathcal{C}'(n, k)$ and $|N(S)| > 1$. If $\beta = b_1b_2 \cdots b_{n-j-1}xk^j$ is the lexicographically largest necklace in $N(S)$ for some $x < k$, then $\alpha \in N(S)$ has prefix $b_1b_2 \cdots b_{n-j-1}$.

Proof. If $\alpha = \beta$, then clearly $\alpha$ has prefix $b_1b_2 \cdots b_{n-j-1}$. Otherwise, we prove by contradiction that $\alpha$ has such a prefix. Suppose $\alpha$ has prefix $a_1a_2 \cdots a_i$, where $i < n-j-1$ and $a_1a_2 \cdots a_i = b_1b_2 \cdots b_{i-1}$ while $a_i \neq b_i$. Observe that $a_i < b_i$ since $\beta$ is the lexicographically largest necklace in $N(S)$ and $\alpha \neq \beta$. Let $\alpha = a_1a_2 \cdots a_i \cdots a_{n-h-1}yk^h$ for some $y < k$ and $h < n$. Since $S \in \mathcal{C}'(n, k)$ and $\alpha < \beta$, $a_1a_2 \cdots a_i \cdots a_{n-h-1}k^{h+1} \in N(S)$. Since $a_1a_2 \cdots a_i \cdots a_{n-h-1}k^{h+1}$ is lexicographically smaller than $\beta$, by the same argument $a_1a_2 \cdots a_i \cdots a_{n-h-2}k^{h+2} \in N(S)$. By repeatedly applying the same argument, $a_1a_2 \cdots a_i \cdots a_{n-h-i}k^{n-i} \in N(S)$ and is lexicographically smaller than $\beta$. However, $a_1a_2 \cdots a_{i-1}k^{i+1}$ should also be in $N(S)$, a contradiction to $\beta$ being the lexicographically largest necklace in $N(S)$. Thus $\alpha$ has prefix $a_1a_2 \cdots a_{n-j-1}$. □

Lemma 4.7.7 Suppose $S \in \mathcal{C}'(n, k)$ and $|N(S)| > 1$. If $\alpha = a_1a_2 \cdots a_{n-j-1}xk^j \in N(S)$, then $FKM(S)$ contains the substring $a_1a_2 \cdots a_n \cdot a_{|ap(\alpha)|-j-1}$.

Proof. Let $p = |ap(\alpha)|$ and let $\beta$ denote the necklace that is after $\alpha$ in the lexicographic ordering $N(S)$. If $\alpha$ is the lexicographically largest necklace in $N(S)$, then we consider the lexicographically smallest necklace in $N(S)$ as $\beta$ by taking account of the circular property of the sequence. By Lemma 4.7.6, $\beta$ has prefix $a_1a_2 \cdots a_{n-j-1}$ and thus $a_1a_2 \cdots a_n \cdot a_{p-j-1}$ is a substring of $FKM(S)$. Otherwise, by Lemma 4.7.1, $\beta$ has prefix $a_1a_2 \cdots a_{n-j-1}$. By Lemma 4.7.2, at most one of these necklaces is periodic and we proceed in three cases.
1. If both $\alpha$ and $\beta$ are aperiodic, then $\mathbf{ap}(\alpha) \cdot \mathbf{ap}(\beta) = \alpha \cdot \beta$ is a substring of FKM($S$) which has prefix $a_1a_2 \cdots a_n \cdot a_1a_2 \cdots a_{n-j-1} = a_1a_2 \cdots a_n \cdot a_1a_2 \cdots a_{p-j-1}$.

2. If $\alpha$ is periodic and $\beta$ is aperiodic, then $\mathbf{ap}(\alpha) \cdot \mathbf{ap}(\beta) = \mathbf{ap}(\alpha) \cdot \beta$ is a substring of FKM($S$) which has prefix $\mathbf{ap}(\alpha) \cdot a_1a_2 \cdots a_{n-j-1}$. Let $\alpha = (\mathbf{ap}(\alpha))^t$ for some $t > 1$. Observe that

$$\mathbf{ap}(\alpha) \cdot a_1a_2 \cdots a_{n-j-1} = \mathbf{ap}(\alpha) \cdot (\mathbf{ap}(\alpha))^{t-1}a_{(t-1)p+1}a_{(t-1)p+2} \cdots a_{n-j-1}$$

$$= \alpha \cdot a_{(t-1)p+1}a_{(t-1)p+2} \cdots a_{n-j-1}$$

$$= a_1a_2 \cdots a_n \cdot a_1a_2 \cdots a_{p-j-1}.$$

3. If $\alpha$ is aperiodic and $\beta$ is periodic, then there are two subcases. If $\beta = k^n$, then $S \in C(n, k)$ and the substring $a_1a_2 \cdots a_n \cdot a_1a_2 \cdots a_{p-j-1}$ is simply equal to $a_1a_2 \cdots a_n$ due to the fact that $\alpha$ has suffix $k^{n-1}$. The desired substring is found in the length $n + 1$ suffix of FKM($S$) by Lemma 4.3.5. Otherwise if $\beta \neq k^n$, then let $\gamma$ be the necklace that is after $\beta$ in the lexicographic ordering of $N(S)$. Such a $\gamma$ exists since $\beta \neq k^n$ and $\beta$ is not the lexicographically largest necklace in $C'(n, k)$ by Lemma 4.7.4 because $\beta$ is periodic. Notice that $\gamma$ is aperiodic by Lemma 4.7.2. Therefore, by the arguments in the previous case, $\mathbf{ap}(\beta) \cdot \mathbf{ap}(\gamma)$ has prefix $\beta$. Therefore, $\mathbf{ap}(\alpha) \cdot \mathbf{ap}(\beta) \cdot \mathbf{ap}(\gamma)$ is a substring of FKM($S$) which has prefix $\alpha \cdot \beta$. The prefix $\alpha \cdot \beta$ contains the prefix $a_1a_2 \cdots a_n \cdot a_1a_2 \cdots a_{p-j-1}$.

Thus FKM($S$) contains the substring $a_1a_2 \cdots a_n \cdot a_1a_2 \cdots a_{|\mathbf{ap}(\alpha)|-j-1}$.

We now prove that the FKM construction also generate universal cycles for sets in $C'(n, k)$, the proof is similar to the one in Theorem 4.3.7.

**Theorem 4.7.8** If $S \in C'(n, k)$, then FKM($S$) is a universal cycle for $S$.

*Proof.* Since $S$ is closed under rotation by the definition of $C'(n, k)$ and its strings all have length $n$, the definition of FKM($S$) implies $|\text{FKM}(S)| = |S|$. Therefore, to prove FKM($S$) is a universal cycle for $S$, we only need to show that FKM($S$) contains each string in $S$ as a substring when the sequence is considered circularly.
Now consider a rotation $\beta = a_i a_{i+1} \cdots a_n a_1 a_2 \cdots a_{i-1}$ of an arbitrary necklace $\alpha = a_1 a_2 \cdots a_n$ in $\mathbf{N}(S)$. If $\beta = k^n$, then $S \in \mathcal{C}(n, k)$ and by Theorem 4.3.7 $FKM(S)$ is a universal cycle for $S$. On the other hand, if $|\mathbf{N}(S)| = 1$, then $FKM(S)$ contains the aperiodic prefix of a single necklace which is a universal cycle for $S$. Thus, for the remainder of the proof we assume $\beta \neq k^n$ and $|\mathbf{N}(S)| > 1$. Without loss of generality, we suppose $\alpha$ has suffix $xk^j$ for some $x < k$. Let $p = |ap(\alpha)|$. We show that all $p$ distinct rotations of $\alpha$ exist in $FKM(S)$. There are two cases depending on the value of $i$.

**Case 1:** $0 < i \leq p - j - 1$: By Lemma 4.7.7, $a_1 a_2 \cdots a_n a_1 a_2 \cdots a_{p-j-1}$ is a substring of $FKM(S)$. Observe that $\beta$ is a substring of $a_1 a_2 \cdots a_n a_1 a_2 \cdots a_{p-j-1}$ when $0 < i \leq p - j - 1$.

**Case 2:** $p - j - 1 < i \leq p$: Observe that $\beta = k^{n-i} a_1 a_2 \cdots a_{i-1}$ when $p - j - 1 < i \leq p$. Let $\gamma$ be the lexicographically smallest necklace in $\mathbf{N}(S)$ such that it has prefix $a_1 a_2 \cdots a_{i-1}$. If $\gamma$ is not the lexicographically smallest necklace in $\mathbf{N}(S)$, then the previous necklace of $\gamma$ has the suffix $k^{n-i}$ due to the fact that $S \in \mathcal{C}'(n, k)$. On the other hand, if $\gamma$ is the lexicographically smallest necklace in $\mathbf{N}(S)$, then the previous $n - i$ symbols in $FKM(S)$ are $k^{n-i}$ by Lemma 4.7.5 when the sequence is considered circularly. Thus, $\beta$ is a substring of $FKM(S)$.

Therefore, $FKM(S)$ contains each string in $S$ as a substring and is a universal cycle for $S$ since $|FKM(S)| = |S|$.

**4.8 Summary**

We generalize the FKM and greedy constructions to construct universal cycles for sets in the language $\mathcal{C}(n, k)$. The language includes a lot of interesting combinatorial objects such as subsets of $k$-ary strings of length $n$ that contains strings (i) with sum at least $s$; (ii) with at least $\ell_k$ copies of $k$; (iii) with at most $u_i$ copies of $i \in \{1, 2, \ldots, k-1\}$; (iv) that circularly avoid substring $\beta \in \{1, 2, \ldots, k-1\}^*$; (v) that are not rotations

71
of some periodic necklaces; (vi) that are rotations of the lexicographically largest \( i \) necklaces; and (vii) that are union or intersection of these sets. By considering the complement of universal cycles for the sets in \( \mathcal{C}(n, k) \), we can further construct some new universal cycles for some interesting combinatorial objects such as subsets of \( k \)-ary strings of length \( n \) that contains strings: (i) with sum at most \( s \); (ii) with at least \( \ell_1 \) copies of 1; (iii) with at most \( u_i \) copies of \( i \in \{2, 3, \ldots, k\} \); (iv) that circularly avoid substring \( \beta \in \{2, 3, \ldots, k\}^* \); and (vi) that are rotations of the lexicographically smallest \( i \) necklaces. We also extend our results to construct universal cycles for sets in \( \mathcal{C}'(n, k) \). We further provide an algorithm which generates some of the universal cycles for sets in \( \mathcal{C}(n, k) \) in \( O(1) \)-amortized time per symbol.

A natural problem in this chapter is to further generalize the FKM and greedy constructions to include more interesting sets. The FKM and greedy constructions may not be applicable when we consider sets other than those in \( \mathcal{C}(n, k) \) and \( \mathcal{C}'(n, k) \). The constructions may also generate different universal cycles when we consider sets other than those in \( \mathcal{C}(n, k) \) and \( \mathcal{C}'(n, k) \). We will introduce some open problems related to this chapter in Chapter 6.
Chapter 5

An efficient universal cycle construction for weight-range binary strings

In this chapter, we introduce the first known efficient universal cycle construction for weight-range binary strings. The construction makes use of a simple lemma (the Gluing lemma) to glue universal cycles together. The lemma provides a new tool to obtain new universal cycles by concatenating two universal cycles together. We then apply the Gluing lemma to prove the existence of universal cycles for other combinatorial objects including subsets of passwords and labeled graphs.

5.1 Definitions

We first define a few notations and terminologies used in this chapter. We denote the set of length $n$ binary strings with number of 1s (weight) range from $c$ to $d$ as $B^c_d(n)$. As an example, $B^3_2(4) = \{0011, 0101, 0110, 1001, 1010, 1100, 0111, 1011, 1101, 1110\}$. Recall that in Chapter 2.4, we define $B_d(n)$ as the set of length $n$ binary strings with weight $d$. We also denote the set of necklaces and Lyndon words with weight $d$ similarly as $N_d(n)$ and $L_d(n)$ respectively. We refer to universal cycles for $B^c_d(n)$ as weight-range universal cycles. When $c = d - 1$, the weight-range universal cycles is known as dual-weight universal cycles. We say a weight-range is even when $|\{c, c + 1, \ldots, d\}|$ is even, and a weight-range is odd when $|\{c, c + 1, \ldots, d\}|$ is odd.
5.2 Gluing universal cycles

In this section we consider concatenating two universal cycles together to obtain a new universal cycle. As we have discussed in Chapter 2.2, if a directed graph is Eulerian, then an Euler cycle of the graph can be obtained by Hierholzer’s algorithm. Hierholzer’s algorithm constructs an Euler cycle by exhaustively concatenating edge-disjoint cycles that share a common vertex. The algorithm repeatedly applies the following lemma to produce an Euler cycle.

Lemma 5.2.1 Let $G = (V, E)$ and $H = (V', E')$ be two Eulerian graphs such that $V \cap V' \neq \emptyset$ and $E \cap E' = \emptyset$. Let $C_G = u_1, u_2, \ldots, u_j, u_1$ and $C_H = v_1, v_2, \ldots, v_k, v_1$ denote Euler cycles in $G$ and $H$ respectively such that $u_1 = v_1$. Then the concatenation of the two cycles $C_{GH} = u_1, \ldots, u_j, v_1, \ldots, v_k, v_1$ is an Euler cycle for $G \cup H$.

A universal cycle for a set $S$ corresponds to an Euler cycle of its de Bruijn graph $G(S)$. Thus by Lemma 5.2.1, universal cycles for two sets $S_1$ and $S_2$ can be joined together to form a new universal cycle for $S_1 \cup S_2$ if $G(S_1)$ and $G(S_2)$ are edge-disjoint and share a common vertex, or in other words, $S_1$ and $S_2$ are disjoint and have instances that share a length $n-1$ prefix or a length $n-1$ suffix. As an example, consider the following two universal cycles:

- universal cycle for $B_2^2(5)$: 000011000101001,
- universal cycle for $B_3^4(5)$: 001111011010111.

The de Bruijn graphs $G(B_2^2(5))$ and $G(B_3^4(5))$ are edge-disjoint and share a common vertex $\alpha = 0011$. Since the universal cycles are cyclic they can be re-written as 001100010100100 and 0011110110101111 respectively. By gluing these two strings together, observe that we obtain a universal cycle for $B_1^4(5) = B_2^2(5) \cup B_3^4(5)$:

$$001100010100100001110110101111.$$
**Lemma 5.2.2 (The Gluing lemma)** Let $U_1$ and $U_2$ be universal cycles for the sets of length $n$ strings $S_1$ and $S_2$, where $S_1 \cap S_2 = \emptyset$ and the length $n - 1$ prefixes of $U_1$ and $U_2$ are the same. Then the concatenated string $U_1 \cdot U_2$ is a universal cycle for $S_1 \cup S_2$.

The Gluing lemma can be applied to construct many new universal cycles based on existing ones. In the next few sections, we apply the Gluing lemma to construct universal cycles for $B^d_c(n)$, and prove the existence of universal cycles for other interesting combinatorial objects including subsets of passwords and labeled graphs.

### 5.3 Existence of weight-range universal cycle

This section starts our discussion on weight-range universal cycles. We first prove that universal cycles exist for $B^d_c(n)$ for any range $0 \leq c < d \leq n$. Some special cases on the existence of universal cycles for $B^d_c(n)$ and their algorithms have been previously studied:

- if $c = d - 1$, then an efficient algorithm is known [86] using cool-lex order,
- if $c = 0$ or $d = n$, then an efficient algorithm is known [90],
- if the weight-range is even, then a polynomial time algorithm is known [98].

We now prove that universal cycles for $B^d_c(n)$ exist for any range $0 \leq c < d \leq n$, slightly more complicated proofs are given in [7, 12]. We prove that by showing the de Bruijn graph $G(B^d_c(n))$ is Eulerian.

**Theorem 5.3.1** $G(B^d_c(n))$ is Eulerian for $0 \leq c < d \leq n$.

**Proof.** We prove that $G(B^d_c(n))$ is Eulerian by showing that it is balanced and weakly connected.

**Balanced:** The vertex set of $G(B^d_c(n))$ contains all strings of length $n - 1$ with weight in the range $c - 1, c, \ldots, d$. Each vertex with weight $c - 1$ has one incoming
edge and one outgoing edge, each labeled 1. Each vertex with weight $d$ has one incoming edge and one outgoing edge, each labeled 0. All other vertices have in-degree and out-degree equal to two.

**Weakly connected:** We apply induction on the size of the weight-range $c, c + 1, \ldots, d$. The base case when $c = d - 1$ is proved in Theorem 2.4 of [86]. For the inductive step assume that $G(\mathcal{B}_c^{d-1}(n))$ is weakly connected for $0 \leq c < d - 1$, and consider $G(\mathcal{B}_c^d(n))$. Observe:

- the vertex set of $G(\mathcal{B}_c^d(n))$ is equal to the union of the vertex sets of $G(\mathcal{B}_c^{d-1}(n))$ and $G(\mathcal{B}_c^{d-1}(n))$,
- the intersection of the vertex sets for $G(\mathcal{B}_c^{d-1}(n))$ and $G(\mathcal{B}_d^{d-1}(n))$ is non-empty,
- the edge sets of $G(\mathcal{B}_c^{d-1}(n))$ and $G(\mathcal{B}_d^{d-1}(n))$ are both subsets of the edge set for $G(\mathcal{B}_c^d(n))$.

Thus, since both $G(\mathcal{B}_c^{d-1}(n))$ and $G(\mathcal{B}_d^{d-1}(n))$ are weakly connected (inductive hypothesis and base case), there will be a path between any two vertices in $G(\mathcal{B}_c^d(n))$.

Therefore, $G(\mathcal{B}_c^d(n))$ is Eulerian by Theorem 2.2.1. □

5.4 **Construction of weight-range universal cycle**

As mentioned in Chapter 5.3, there exist an efficient algorithm to construct a dual-weight universal cycle. We use the Gluing lemma to create a weight-range universal cycle with even weight-ranges by gluing these dual-weight universal cycles together. To create weight-range universal cycles with odd weight-ranges, we will have to glue in individual necklaces which are defined in later sections.
Even weight-range

First we present an efficient construction of weight-range universal cycle with even weight-range. Suppose we want to construct a universal cycle for $B_{d-3}^d(n)$ from universal cycles for $B_{d-3}^{d-2}(n)$ and $B_{d-1}^d(n)$. Observe that $B_{d-3}^{d-2}(n)$ and $B_{d-1}^d(n)$ are disjoint, and their universal cycles share common length $n - 1$ substrings with weight $d - 2$. Thus, we can apply the Gluing lemma to construct a universal cycle for $B_{d-3}^d(n)$. We can then repeatedly apply the Gluing lemma on the resulting universal cycle and dual-weight universal cycles of lower weight-ranges to obtain a universal cycle of an even weight-range.

The difficult task remains on how to glue universal cycles efficiently, that is, gluing the universal cycles without scanning the whole sequence for a common length $n - 1$ substring. Let $UC_{d-1}^d(n)$ denote a dual-weight universal cycle constructed by the cool-lex construction mentioned in Chapter 2.4.1. We study the length $n + 1$ prefix of $UC_{d-1}^d(n)$ and use the prefix property to efficiently glue universal cycles together.

**Lemma 5.4.1** [86] The first necklace in reverse cool-lex order for $B_d(n+1)$ is $0^{n-d+1}1^d$.

Applying this lemma, the first $n+1$ bits in $UC_{d-3}^{d-2}(n)$ and $UC_{d-1}^d(n)$ are thus $0^{n-d+3}1^{d-2}$ and $0^{n-d+1}1^d$ respectively. If we rotate $UC_{d-3}^{d-2}(n)$ to the left by 2 bits, then the first $n - 1$ bits of both cycles are $0^{n-d+1}1^{d-2}$.

Let $VC_{d-1}^d(n)$ denote the sequence $UC_{d-1}^d(n)$ with the first 2 bits removed. The following recursive formula provides a construction of the universal cycle $UE_c^d(n)$ for $B_c^d(n)$ where the weight-range is even:

$$UE_c^d(n) = \begin{cases} 
UC_{d-1}^d(n) & \text{if } c = d - 1; \\
UC_{c+2}^d(n) \cdot VC_{c+1}^c(n) \cdot 00 & \text{if } c < d - 1. 
\end{cases}$$

We thus obtain the following formula by expanding the recursive function:

$$UE_c^d(n) = UC_{d-1}^d(n) \cdot VC_{d-3}^{d-2}(n) \cdot VC_{d-5}^{d-4}(n) \cdots VC_{c+1}^c(n) \cdot 0^{d-c-1}.$$  

**Theorem 5.4.2** $UE_c^d(n)$ is a universal cycle for $B_c^d(n)$ when $0 \leq c < d \leq n$ and the weight-range is even.
The example discussed in Chapter 5.2 shows how $\text{UE}_4^1(5)$ is constructed from $\text{UC}_3^4(5)$ and $\text{UC}_2^2(5)$. Note that the universal cycle $\text{UE}_c^d(n)$ is different from those created by the algorithms in [98].

**Theorem 5.4.3** A universal cycle $\text{UE}_c^d(n)$ for $\text{B}_d^c(n)$ can be constructed in $O(1)$-amortized time using $O(n)$ space when weight-range is even and $0 \leq c < d \leq n$.

*Proof.* Since there exist an efficient algorithm to construct the universal cycle $\text{UC}_{d-1}^d$ in $O(1)$-amortized time using $O(n)$ space, the sequence $\text{VC}_{d-1}^d$ can easily be obtained by constructing $\text{UC}_{d-1}^d$ and removing its first 2 bits. Therefore there exist an efficient algorithm to construct $\text{UE}_c^d(n)$ in $O(1)$-amortized time using $O(n)$ space when the weight-range is even. □

This is the first known efficient algorithm to construct even weight-range universal cycles. The constructions discussed in [98] are not accompanied by an efficient algorithm.

**Incrementing the weight-range (odd weight-range)**

In this section, we consider extending the weight-range of a universal cycle for $\text{B}_{d-1}^c(n)$ into a universal cycle for $\text{B}_d^c(n)$. We refer this process as *incrementing* the universal cycle’s weight range. The process of incrementing the weight-range of universal cycles allows us to extend an even weight-range universal cycle to an odd weight-range universal cycle.

Let $\alpha_1, \alpha_2, \ldots, \alpha_{|\text{N}_d(n)|}$ denote the necklaces in $\text{N}_d(n)$. We partition the strings in $\text{B}_d(n)$ into their necklace equivalence classes such that $\text{B}_d(n) = \text{Neck}(\alpha_1) \cup \text{Neck}(\alpha_2) \cup \cdots \cup \text{Neck}(\alpha_{|\text{N}_d(n)|})$. For example, $\text{B}_3(6)$ can be partitioned into four subsets $\text{B}_3(6) = \text{Neck}(000111) \cup \text{Neck}(001011) \cup \text{Neck}(001101) \cup \text{Neck}(010101)$ with elements of each set listed as follows:

- $\text{Neck}(000111) = \{000111, 001110, 011100, 111000, 110001, 100011\}$,
- $\text{Neck}(001011) = \{001011, 010110, 101100, 011001, 110010, 100101\}$,
The de Bruijn graph $G(\text{Neck}(\alpha_j))$ for each $j$ where $1 \leq j \leq |\mathcal{N}_d(n)|$ forms a simple cycle with concatenation of its edge labels correspond to $\mathcal{A}(\alpha_j)$, which is a universal cycle for $\text{Neck}(\alpha_j)$. As an example, Figure 5.1 illustrates the de Bruijn graphs for the four necklace equivalence classes that make up $B_3(6)$. The concatenation of edge labels of the cycles are 000111, 001011, 001101 and 01 which correspond to the universal cycles for $\text{Neck}(000111)$, $\text{Neck}(001011)$, $\text{Neck}(001101)$ and $\text{Neck}(010101)$ respectively. Notice that a universal cycle for $\text{Neck}(\alpha_j)$ has length less than $n$ when $\alpha_j \in \mathcal{N}_d(n)$ is periodic. The length $n - 1$ prefixes of these universal cycles are the length $n - 1$ prefixes of the corresponding necklace. For example, 01010 is the length 5 prefix of the universal cycle 01 when $n = 6$ since the universal cycle is traversed repeatedly.

Observe that $B_{c}^{d-1}(n)$ and $\text{Neck}(\alpha_j)$ are disjoint, and we can rotate a universal cycle for $B_{c}^{d-1}(n)$ such that its length $n - 1$ prefix is equal to that of $\text{Neck}(\alpha_j)$. We can then apply Lemma 5.2.2 (the Gluing lemma) to repeatedly concatenate universal cycles for each necklace equivalence class $\text{Neck}(\alpha_j)$ with a universal cycle for $B_{c}^{d-1}(n)$. For example, consider the universal cycles for $B_{2}^{d-1}(6)$, $\text{Neck}(000111)$, $\text{Neck}(001011)$, $\text{Neck}(001101)$ and $\text{Neck}(010101)$:
• universal cycle for Neck(000111): 000111,
• universal cycle for Neck(001101): 001101,
• universal cycle for Neck(001011): 001011,
• universal cycle for Neck(010101): 01,
• universal cycle for $B^2_1(6)$: 000110001010001000.

By repeatedly applying the Gluing lemma, we obtain a universal cycle for $B^3_1(6)$ as follows:

1. Glue the universal cycle for Neck(000111) and the universal cycle for $B^2_1(6)$:
   $\upDelta 00011\underline{00011}00000101000100100$, which is equivalent to the rotation,
   $\upDelta 001100001010001000001110$.

2. Glue the universal cycle for Neck(001101) and the universal cycle for $B^2_1(6) \cup Neck(000111)$:
   $\upDelta 00110100011000101000100100001110$, which is equivalent to the rotation,
   $\upDelta 001010000101000011100011001001100$.

3. Glue the universal cycles for Neck(001011) and the universal cycle for $B^2_1(6) \cup Neck(000111) \cup Neck(001101)$:
   $\upDelta 0010110101000100001000001110001101001100$, which is equivalent to the rotation,
   $\upDelta 0101001001000111100011010011000100110$.

4. Glue the universal cycles for Neck(010101) and the universal cycle for $B^2_1(6) \cup Neck(000111) \cup Neck(001101) \cup Neck(001011) \cup Neck(010101)$:
   $\upDelta 01010100010000111100011010001100010110$.

Since $B^2_1(6) \cup Neck(000111) \cup Neck(001101) \cup Neck(001011) \cup Neck(010101) = B^3_1(6)$, we obtain a universal cycle for $B^3_1(6)$. 80
To generate a weight-range universal cycle $UD^d_c$ with odd weight-range from $c$ to $d$, we insert the missing universal cycles for $\text{Neck}(\alpha_j)$ into a weight-range universal cycle with even weight-range from $c$ to $d-1$. Now given a length $n$ necklace $\alpha \cdot 1 \in N_d(n)$, the string $\alpha \cdot 0$ has weight $d-1$ and it exists in the even weight-range universal cycle with weight-range from $c$ to $d-1$, denoted by $UD^{d-1}_c$. We can thus increment the weight-range by scanning $UD^{d-1}_c$ and insert all strings $\beta \cdot 1$ when $\beta \cdot 1 \in N_d(n)$, where $\beta$ is a length $n - 1$ substring in $UD^{d-1}_c$.

**Theorem 5.4.4** $UD^d_c$ is a universal cycle for $B^d_c(n)$ when $0 \leq c < d \leq n$.

*Proof.* The string $ap(\beta \cdot 1)$ is a universal cycle for $\text{Neck}(\beta \cdot 1)$ where $\beta \cdot 1 \in N_d(n)$. Since $B^{d-1}_c(n)$ and $\text{Neck}(\beta \cdot 1)$ are disjoint and their universal cycles have the same length $n-1$ prefix $\beta$, by the Gluing lemma the construction exhaustively concatenates $UD^{d-1}_c$ and universal cycles for each necklace equivalence class $\text{Neck}(\alpha_j)$ where $\alpha_j \in N_d(n)$. The resulting string $UD^d_c$ is a universal cycle for the set $B^{d-1}_c(n) \cup \text{Neck}(\alpha_1) \cup \text{Neck}(\alpha_2) \cup \cdots \cup \text{Neck}(\alpha_{|N_d(n)|})$, that is $B^d_c(n)$. \qed

**Efficient implementation to increment weight range**

In this section, we consider efficiently increment the weight-range of a universal cycle for $B^{d-1}_c(n)$ into a universal cycle for $B^d_c(n)$. We assume that there is an efficient algorithm that outputs a universal cycle for $B^{d-1}_c(n)$ one bit at a time. We buffer this output into a sliding window, and examine it to determine if any additional bits need to be outputted. We first describe how this process works, and then we describe how to make the process efficient.

**A simple algorithm: SimpleIncrement**

A *linear universal string* of a universal cycle is obtained by appending the first $n - 1$ symbols to its end. For example, a linear universal string for the universal cycle 00010011010111110000 for $B(4)$ is 00001001101011111000. The construction SimpleIncrement follows the approach in Chapter 5.4. The algorithm reads each bit from a
Algorithm 7 Simple algorithm to increment weight-range of a given weight-range universal cycle in $O(n)$-amortized time per bit.

1: procedure SIMPLEINCREMENT
2:   $s \leftarrow 1$
3:   for $s$ from 1 to $|UD_c^{d-1}|$ do
4:     $t \leftarrow s + n - 1$
5:     $\alpha \leftarrow b_s b_{s+1} \cdots b_t$
6:     if $\alpha \cdot 1 \in N_d(n)$ then
7:       Print($\alpha \cdot 1$) // Insertion of $ap(\alpha \cdot 1)$
8:     Print($b_s$)
9:   $s \leftarrow s + 1$

linear universal string $b_1 b_2 \cdots b_{|UD_c^{d-1}|+n-1}$ for $B_c^{d-1}(n)$. It examines the sliding window $\alpha = b_s b_{s+1} \cdots b_t$ of size $n-1$ and inserts $ap(\alpha \cdot 1)$ if $\alpha \cdot 1 \in N_d(n)$. A membership tester for $N_d(n)$ can be completed in $O(n)$ time as mentioned in Chapter 2.1. Pseudocode that produces $UD_c^d(n)$ is shown in Algorithm 7.

Theorem 5.4.5 A universal cycle $UD_c^d(n)$ for $B_c^d(n)$ can be constructed in $O(n)$-amortized time per bit using $O(n)$ space for any weight-range where $0 \leq c < d \leq n$.

Proof. In addition to the time and space required to produce an input linear universal string for $B_c^{d-1}(n)$, the algorithm SIMPLEINCREMENT uses an additional $O(n)$ amount of work and $O(n)$ space per bit using a membership tester for $N_d(n)$. From Theorem 5.4.3, since we can construct an even weight-range universal cycle in $O(1)$-amortized time using $O(n)$ space, we can therefore construct $UD_c^d(n)$ in $O(n)$-amortized time per bit using $O(n)$ space. \qed
Algorithm 8 Fast algorithm to increment weight-range of a given weight-range universal cycle in $O(1)$ amortized time per bit.

1: procedure FastIncrement
2: $p \leftarrow 1; w \leftarrow 0; s \leftarrow 1$
3: for $s$ from 1 to $|UD_{c-1}|$ do
4: $w \leftarrow w + b_t$
5: // Glue universal cycles
6: if $t - s + 1 = n$ and $w = d - 1$ and $b_t = 0$ then
7: if $b_{t-p} < 1$ then Print($b_s b_{s+1} \cdots b_{t-1}$)
8: else Print($b_s b_{s+1} \cdots b_{s+p-1}$)
9: // Maintain prenecklace
10: if $b_t - p < b_t$ then $p \leftarrow t - s + 1$
11: else if $b_t - p > b_t$ then
12: $(s', p, w) \leftarrow$ Update($s + \lfloor \frac{t-s}{p} \rfloor \cdot p$, $w$)
13: Print($b_s b_{s+1} \cdots b_{s'-1}$)
14: $s \leftarrow s'$
15: // Update window-size
16: if $t - s + 1 = n$ then
17: if $p > n/2$ then $(s', p, w) \leftarrow$ Update($s + 1$, $w$)
18: else $(s', p, w) \leftarrow$ Update($s + p$, $w$)
19: Print($b_s b_{s+1} \cdots b_{s'-1}$)
20: $s \leftarrow s'$
21: function Update($k, w$)
22: $s' \leftarrow k; p \leftarrow 1; w' \leftarrow w$
23: for $i$ from $k + 1$ to $t$ do
24: if $b_{t-p} < b_t$ then $p \leftarrow i - s' + 1$
25: else if $b_{t-p} > b_t$ then
26: $s' \leftarrow s' + \lfloor \frac{i-s'}{p} \rfloor \cdot p$
27: $p \leftarrow 1$
28: for $j$ from $s' + 1$ to $i$ do
29: if $b_{j-p} < b_j$ then $p \leftarrow j - s' + 1$
30: for $i$ from $s$ to $s' - 1$ do
31: if $b_i = 1$ then $w' \leftarrow w - 1$
32: return($s', p, w'$)

Extending SimpleIncrement to CAT

The major overhead of SimpleIncrement in terms of efficiency comes from the membership tester for $N_d(n)$ which takes $O(n)$ amount of work per bit. To efficiently locate the position to insert the aperiodic prefixes, we maintain a sliding window $\beta = b_s b_{s+1} \cdots b_t$ of variable size instead of a length $n - 1$ window $\alpha$ in Algorithm 7.

Pseudocode of the efficient construction to increment a weight-range is given in Algorithm 8. The initial call is FastIncrement. The function Update($k, w$) scans
the string $b_kb_{k+1} \cdots b_t$ to update $s$ to $s'$ such that $b_{s'}b_{s'+1} \cdots b_t$ is a prenecklace and $p$ is the length of $\text{ap}(b_{s'}b_{s'+1} \cdots b_t)$. The algorithm can be summarized into the following three steps:

**Glue universal cycles:** We insert $\text{ap}(b_s b_{s+1} \cdots b_{t-1} \cdot 1)$ before the position $s$ if $b_s b_{s+1} \cdots b_{t-1} \cdot 1 \in \mathbb{N}_d(n)$. The string $b_s b_{s+1} \cdots b_{t-1} \cdot 1 \in \mathbb{N}_d(n)$ if $t - s + 1 = n$, $w = d - 1$ and $b_t = 0$.

**Maintain prenecklace:** We maintain the variables $s$ and $p$ such that $\beta$ is a prenecklace and $p$ is the length of $\text{ap}(\beta)$. There are three cases:

- if $b_{t-p} < b_t$, then $\beta$ is a prenecklace and $\text{ap}(\beta) = b_s b_{s+1} \cdots b_t$, thus we update $p = |\beta| = t - s + 1$;
- if $b_{t-p} = b_t$, then $\beta$ is a prenecklace and the aperiodic prefix remains unchanged, that is $\text{ap}(\beta) = \text{ap}(b_s b_{s+1} \cdots b_{t-1}) = b_s b_{s+1} \cdots b_{s+p-1}$, we keep $s$ and $p$ unchanged;
- if $b_{t-p} > b_t$, then $\beta$ is not a prenecklace; we update $s$ to $s + \lfloor \frac{t - s}{p} \rfloor \cdot p$; we update $p$ to be the length of $\text{ap}(b_{s+\lfloor \frac{t - s}{p} \rfloor \cdot p} \cdots b_t)$;

> for example, consider $n = 13$, $\beta = b_1 b_2 \cdots b_{12} = 001001001000$ and $p = 3$; $\beta$ is not a necklace because $b_{12-3} > b_{12}$; the variable $s$ is thus updated to $1 + \lfloor \frac{10}{3} \rfloor \cdot 3 = 10$ such that the sliding window $\beta$ starts with $b_{10} b_{11} b_{12} = 000$ which is lexicographically smaller than $b_1 b_2 b_3 = 001$; $p$ is updated to the length of $\text{ap}(b_{10} b_{11} b_{12}) = |\text{ap}(000)| = 1$.

**Update window-size:** We increment the variable $s$ to $s'$ when the size of $\beta$ reaches $n$ such that $b_{s'} b_{s'+1} \cdots b_t$ is a prenecklace; we update $p$ to be the length of $\text{ap}(b_{s'} b_{s'+1} \cdots b_t)$.

**Runtime analysis**

We demonstrate that the total amount of work of FastIncrement divided by the number of bits in a universal cycle for $B_d^c(n)$ is bounded by a constant. The works required for each of the above steps are as follows:
Glue universal cycles: The string $b_s b_{s+1} \cdots b_{t-1} \cdot 1 \in \mathbb{N}_d(n)$ if $w = d - 1$, $b_t = 0$ and $t - s + 1 = n$, this can be verified using only $O(1)$ time per bit.

Maintain prenecklace: If $b_{t-p} \leq b_t$, then $\beta$ is a prenecklace and we update $p$ which requires a constant amount of work; if $b_{t-p} > b_t$, we call the function $\text{Update}(s + [\frac{t-s}{p}] \cdot p, w)$ which requires $O(t - s - [\frac{t-s}{p}] \cdot p)$ amount of work; however, we print at least $[\frac{t-s}{p}] \cdot p$ bits. Observe that $t - s - [\frac{t-s}{p}] \cdot p < [\frac{t-s}{p}] \cdot p$ by the membership tester of necklace, the amount of work required is thus directly proportional to the number of bits, that is $O(1)$-amortized time per bit.

Update window-size: The algorithm calls the function $\text{Update}(s + 1, w)$ when the size of the sliding window $\beta$ reaches $n$ but prints only one bit. Thus, it requires $O(n)$ time per bit.

Let $P(n, w)$ denote the cardinality of the set of length $n$ binary prenecklaces with weight $w$, and $P_0(n, w)$ and $P_1(n, w)$ denote the cardinality of the sets of length $n$ binary prenecklaces with weight $w$ that end with the bit 0 and 1 respectively. We show that the number of length $n$ binary prenecklaces is bounded by the number of bits in the universal cycle for $\mathbb{B}^d_c(n)$ over $n$. The following lemma provides an upper bound of $P_1(n, w)$ in terms of $|N_w(n)|$ and $|L_w(n)|$.

Lemma 5.4.6 [89] $P_1(n, w) \leq |N_w(n)| + |L_w(n)|$.

Consider the upper bound of $P_0(n, w)$, replacing the last bit of a prenecklace of weight $w$ that ends with a 0 with a 1 will always yield a unique necklace of weight $w + 1$, $P_0(n, w)$ is therefore bounded by the number of necklaces of weight $w + 1$.

Lemma 5.4.7 $P_0(n, w) \leq |N_{w+1}(n)|$.

The upper bound of $|N_w(n)|$ and $|L_w(n)|$ in terms of $\binom{n}{w}$ was discussed in [89] and are given as follows:

$$|L_w(n)| \leq \frac{1}{n} \binom{n}{w} \quad \text{and} \quad |N_w(n)| \leq 2 |L_w(n)| \leq 2 \frac{n}{n} \binom{n}{w}.$$
The upper bound of \( P(n, w) \) in terms of \( \binom{n}{w} \) is therefore as follows:

\[
P(n, w) = P_0(n, w) + P_1(n, w)
\]

\[
\leq |N_{w+1}(n)| + |N_w(n)| + |L_w(n)|
\]

\[
\leq \frac{2}{n} \binom{n}{w + 1} + \frac{2}{n} \binom{n}{w} + \frac{1}{n} \binom{n}{w}
\]

\[
\leq \frac{2}{n} \binom{n}{w + 1} + \frac{3}{n} \binom{n}{w}.
\]

**Theorem 5.4.8** Algorithm `FastIncrement` is a CAT algorithm.

**Proof.** Let \( hn \) be the amount of work required in step 3 of `FastIncrement` to update a prenecklace, where \( h \) is a constant. The ratio between the total amount of work required in step 3 of `FastIncrement` to \( |B_d^c(n)| \) is as follows:

\[
\frac{\text{Total work in step 3}}{|B^d_c(n)|} = \frac{(P(n, d - 1) + P(n, d - 2) + \cdots + P(n, c)) \times hn}{\binom{n}{d} + \binom{n}{d-1} + \cdots + \binom{n}{c}}
\]

\[
\leq \frac{\left( \frac{2}{n} \binom{n}{d} + \frac{5}{n} \binom{n}{d-1} + \frac{5}{n} \binom{n}{d-2} + \cdots + \frac{5}{n} \binom{n}{c+1} + \frac{3}{n} \binom{n}{c} \right) \times hn}{\binom{n}{d} + \binom{n}{d-1} + \cdots + \binom{n}{c}}
\]

\[
= \frac{\left( 2 \binom{n}{d} + 5 \binom{n}{d-1} + 5 \binom{n}{d-2} + \cdots + 5 \binom{n}{c+1} + 3 \binom{n}{c} \right) \times h}{\binom{n}{d} + \binom{n}{d-1} + \cdots + \binom{n}{c}}
\]

\[
\leq 5h.
\]

Since stage 1 and 2 of `FastIncrement` requires only \( O(1) \)-amortized time per bit, the algorithm `FastIncrement` is a CAT algorithm. \( \square \)

**Theorem 5.4.9** A universal cycle \( UD_d^c(n) \) for \( B_d^c(n) \) can be constructed in \( O(1) \)-amortized time per bit using \( O(n) \) space for any weight-range where \( 0 \leq c < d \leq n \).

### 5.5 Other applications of the Gluing lemma

In this section we consider other sets of strings and their associated universal cycles. We apply the Gluing lemma to prove the existence of universal cycles for these sets.
5.5.1 Passwords

The set of passwords is defined to be the set of all strings of length $n$ over an alphabet of size $k$ partitioned into $q < k$ classes where each string contains at least one character from each class. For instance, 4 natural classes would be: lower case letters, upper case letters, digits, and special characters. A very secure password would contain one symbol from each class. In [60], Leitner and Godbole proved the following result:

**Theorem 5.5.1** A universal cycle exists for all $n$-letter passwords over an alphabet of size $k$ partitioned into $q < k$ classes, provided that $n \geq 2q$.

We relax the definition of a password to be a string that contains at least one symbol from $q' \leq q$ classes. In fact, this is a common requirement of passwords where they must either contain a number or a special character. As an example, consider all passwords of length $n$ containing characters in at least two classes. Such strings can be partitioned into $\binom{4}{2}$ sets of words containing exactly 2 classes, plus 4 sets of words containing exactly 3 classes, plus one set containing characters from all 4 classes. Observe that all sets are disjoint, and the sets containing strings from exactly 2 classes have many strings that have $n - 1$ characters in common. For instance ‘aAAAAAAAA’ and ‘1AAAAAAA’ and ‘#AAAAAAA’. Similarly, there exist common strings of length $n - 1$ between a set of exactly 2 classes and a set with one additional class. For instance ‘aAAAAAAAA’ and ‘aAAAAAA3’. Thus, the following theorem follows from Lemma 5.2.2.

**Theorem 5.5.2** Let an alphabet of size $k$ be partitioned into $q < k$ classes. there exist a universal cycle for all strings of length $n$ containing letters from at least $q' \leq q$ classes, provided that $n \geq 2q$.

Observe that if $q' = 1$, then the universal cycle is a traditional de Bruijn sequence over an alphabet of size $k$. 
5.5.2 Strings with content-range and sum-range over a general alphabet

We generalize the idea of weight-range binary strings to general alphabets. We say that a $k$-ary string of length $n$ has sum $s$ if the summation of the values of all $n$ symbols is equal to $s$. Let $\text{Sum}(c,d)$ denote the set that contains all $k$-ary strings with sum range from $c$ to $d$. Suppose there exist a universal cycle for $S_1 = \text{Sum}(c,d)$ and a universal cycle for $S_2 = \text{Sum}(d + h, j)$ where $h \leq k$ and $j > d + h$. Observe that there exist strings in $S_1$ and $S_2$ that have the same length $n - 1$ prefixes, and clearly $S_1 \cap S_2 = \emptyset$. Then by applying Lemma 5.2.2, there exist a universal cycle for $S_1 \cup S_2$.

The notion of weight-range binary strings can be further extended by considering the set of strings with fixed-content. We say that a set of strings has fixed-content if the number of occurrences of each symbol is fixed in each string. Let $\text{Fix}(n_0, n_1, \ldots, n_{k-1})$ denote a fixed-content set containing all $k$-ary strings with $n_i$ occurrences of the symbol $i$, where $0 \leq i < k$. Suppose there exist a universal cycle for $S_1 = \text{Fix}(n_0, n_1, \ldots, n_{k-1})$ and a universal cycle for $S_2 = \text{Fix}(n'_0, n'_1, \ldots, n'_{k-1})$ where the content of the sets differs for exactly two symbols $j$ and $t$ and WLOG $n_j = n'_j + 1$ and $n_t = n'_t - 1$. Then by applying Lemma 5.2.2, there exist a universal cycle for $S_1 \cup S_2$.

5.5.3 Labeled graphs

In [11], a number of universal cycle existence questions are given for various labeled graphs. Instead of strings, they consider graphs with a sliding window of size $k$ that represent labeled graphs. In particular, they give the following result:

**Theorem 5.5.3** [11] Universal cycles exist for labeled graphs with precisely $m$ edges (and $k$ vertices).

Since graphs with $m$ edges and graphs with $m + 1$ edges are disjoint and their universal cycles have many graphs with identical $k - 1$ windows, we can apply Lemma 5.2.2 to obtain the following result:
Theorem 5.5.4 Universal cycles exist for labeled graphs with between \( m_1 \) and \( m_2 \) edges (and \( k \) vertices).

It remains an open problem to find efficient constructions for such universal cycles.

5.6 Summary

To conclude, we introduce the Gluing lemma to glue universal cycles together. The lemma provides a new tool to obtain new universal cycles by concatenating two universal cycles together. The lemma is applied to efficiently construct universal cycles for the set of length \( n \) binary strings with weight in the range \( c, c + 1, \ldots, d \) where \( 0 \leq c < d \leq n \) in \( O(1) \)-amortized time per bit. It is also applied to prove the existence of universal cycles for other combinatorial objects including subsets of passwords and labeled graphs.
Chapter 6

Summary and open problems

This thesis was first motivated by the following questions:

1. Is there a simple successor rule to generate the next symbol in a de Bruijn sequence for all values of \( n \) and \( k \)?

2. Do universal cycles exist for some previously unstudied, yet interesting subsets of length \( n \) binary strings and \( k \)-ary strings?

3. If so, are there simple and efficient constructions to generate the universal cycles?

The goal of our research is to discover new universal cycle constructions for common combinatorial objects, and additionally to develop algorithms to generate the universal cycles efficiently. In particular, the following constructions are outlined in this thesis:

1. A shift rule \( f \) to construct de Bruijn sequence for length \( n \) binary strings;

2. An algorithm that generates the de Bruijn sequence by \( f \) in \( O(1) \)-amortized time per bit;

3. A shift rule \( f_k \) to construct a de Bruijn sequence for \( k \)-ary strings of length \( n \);

4. An algorithm that generates the de Bruijn sequence by \( f_k \) in \( O(1) \)-amortized time per symbol;

5. Generalizations of the FKM and greedy constructions to construct universal cycles for sets in \( \mathcal{C}(n, k) \) and \( \mathcal{C}'(n, k) \);
6. The first known universal cycle construction for weight-range binary strings that generates the sequence in $O(1)$-amortized time per bit.

To conclude this thesis, we outline several open problems and future avenues of our research.

### 6.1 Open problems related to shift rule

This section outlines two future research avenues related to our shift rule discussed in Chapter 3.

**Extensions of shift rule**

A natural extension of our shift rule $f$ is to generalize the shift rule to more combinatorial objects. A few natural open questions are as follows:

1. Can the shift rule $f$ be restricted to the set of aperiodic strings?

2. Can the shift rule $f$ be augmented to list all binary strings with a given weight constraint?

Recent work suggests that the answer to both questions is ‘yes’. As a preview, the following successor rules $f_{ap}$ and $f_{max}$ are proposed as answers to questions 1 and 2 respectively (Let $\alpha = b_1 b_2 \cdots b_n$):

$$f_{ap}(\alpha) = \begin{cases} b_2 b_3 \cdots b_n \overline{b_1} & \text{if } b_2 b_3 \cdots b_n 1 \in \mathbb{N}(n) \text{ and } b_2 b_3 \cdots b_n \overline{b_1} \text{ is aperiodic;} \\ b_2 b_3 \cdots b_n b_1 & \text{otherwise.} \end{cases}$$

$$f_{max}(\alpha) = \begin{cases} b_2 b_3 \cdots b_n \overline{b_1} & \text{if } b_2 b_3 \cdots b_n 1 \in \mathbb{N}(n) \text{ and } b_2 b_3 \cdots b_n \overline{b_1} \in \mathbb{B}_0^d(n); \\ b_2 b_3 \cdots b_n b_1 & \text{otherwise.} \end{cases}$$

We suspect these results can be further generalized to more interesting combinatorial objects. It would be interesting to explore if there exist a “unified shift rule” that is applicable to generate the next symbol of universal cycles for a variety of combinatorial objects in some language.
Successor rule perspective for classic constructions

Many results on universal cycle constructions take a global perspective and produce universal cycles by concatenating necklaces according to some ordering. These constructions currently do not have a successor rule. Interesting research avenues include reinterpreting these classic constructions in a successor rule perspective. Recent work suggests that there exist successor rules to describe the FKM construction and the cool-lex construction.

6.2 Open problems related to the FKM and greedy constructions

This section outlines several open problems related to the generalized FKM and greedy constructions in Chapter 4.

A language \( L \) such that \( FKM(S) \) is a universal cycle if and only if \( S \in L \)?

Although the language \( C(n, k) \) includes a broad class of combinatorial objects, there are sets that are not in \( C(n, k) \) while their universal cycles can be constructed by the FKM construction. For example, consider the set \( X = \{1122, 1212, 1221, 1222, 1322, 2112, 2121, 2122, 2132, 2211, 2212, 2213, 2221, 3221\} \subseteq T(4, 3) \). The set \( X \) is closed under rotation, but \( X \notin C(4, 3) \) since \( N(X) = \{1122, 1212, 1222, 1322\} \) is not a \( k \)-suffix language (\( X \) is also not in \( C'(4, 3) \)). However, \( FKM(X) = 1122 \cdot 12 \cdot 1222 \cdot 1322 \) is a universal cycle. A natural open problem is thus to explore a language \( L \) such that \( FKM(S) \) is a universal cycle for \( S \) if and only if \( S \in L \).

When \( Greedy(S) = FKM(S) \)?

In Chapter 4, we prove that \( FKM(S) \) and \( Greedy(S) \) produce the same universal cycle when \( S \in C(n, k) \). However, the algorithms do not always produce the same sequence. As an example, consider the set \( X = \{111, 112, 121, 122, 123, 211, 212, 221, 222, 223, 231, 232, 233, 312, 322, 323, 332, 333\} \subseteq T(3, 3) \). The sequence
Greedy(X) = 2211212231232233

is a universal cycle for X, but FKM(X) = 1 \cdot 112 \cdot 122 \cdot 123 \cdot 2 \cdot 223 \cdot 233 \cdot 3 is not even a universal cycle. An interesting problem is therefore to investigate when Greedy(S) produces the same sequence as FKM(S). Is there a language to include all sets S where Greedy(S) = FKM(S)?

6.3 Other open problems

This section outlines two more interesting open problems related to construction of universal cycles.

The cutting-down problem

The cutting-down problem considers constructions of universal cycles for subsets of k-ary strings of length n which contains t objects, where t is an arbitrary value \( n \leq t \leq k^n \). These sequences are known as cutting-down sequences. The problem was first proposed in [21] for application on de Bruijn card trick, and was listed as an open problem in [52]. Several partial solutions of this problem have been published, but none of these approaches successfully cover all possible values of t.

As we have mentioned in Chapter 4, Moreno gave a partial solution to the problem by providing a construction of cutting-down sequences with length equal to the total length of aperiodic prefixes of some lexicographically largest necklaces of \( B(n) \). We further extend their result to construct cutting-down sequences for more values of t.

When t is equal to the summation of the length of aperiodic prefixes of some lexicographically largest necklaces in \( B(n) \), then we can simply adopt Moreno’s approach to construct the cutting-down sequence. Otherwise, let \( \mathcal{L}_s \) denote a cutting-down sequence obtained by concatenating the aperiodic prefixes of the last s length n necklaces in the lexicographical ordering of \( N(n) \). Let i be some value 0 \leq i < |N(n)| such that \( |\mathcal{L}_i| < t \) but \( |\mathcal{L}_{i+1}| > t \). We can then extend the sequence \( \mathcal{L}_i \) by appending 0s
until we reach the length \( t \), or there are \( n \) 0s. As an example, to construct a cutting-down sequence of length 34 with \( n = 6 \), we first obtain the closest cutting-down sequence \( L_7 \) which has length 30 as follows:

\[
001101 \cdot 001111 \cdot 01 \cdot 010111 \cdot 011 \cdot 011111 \cdot 1.
\]

We are 4 symbols short to produce a cutting-down sequence of length 34. Observe that we can at most be 5 bits short, or otherwise we should include the aperiodic prefix of the necklace that is just lexicographically smaller than the first necklace in the list. To extend the cutting-down sequence \( L_7 \), we insert four 0s in front. The sequence becomes a cutting-down sequence of length 34 with \( n = 6 \) as follows:

\[
0000 \cdot 001101 \cdot 001111 \cdot 01 \cdot 010111 \cdot 011 \cdot 011111 \cdot 1.
\]

However, this approach only guarantees a construction of cutting-down sequences with a range of \( t \) where \( n < t < 2^n - n(2^{\lfloor \frac{n}{2} \rfloor} - 3) - 1 \). Some other extension methods are required to obtain a larger range of \( t \).

**Multi-shift universal cycle and de Bruijn torus**

A *multi-shift universal cycle*, or an *m-shift universal cycle* for \( B(n) \) is a sequence \( U_m \) that contains each length \( n \) binary string exactly once as a factor that starts at a position \( km + 1 \) where \( 0 \leq k < \frac{|U_m|}{m} \). As an example, the sequence

\[
0000100111011100010011011110
\]

is a 2-shift universal cycle since its 16 substrings of length 4 starting at position \( 2k + 1 \) contains each length 4 binary string:

\[
0000, 0010, 1001, 0110, 1010, 1011, 1111, 1100, 0001, 0100, 0011, 1101, 0101, 0111, 1110, 1000.
\]

The problem was discussed in [54] and [55] motivated by the application on Frobenius problem in a free monoid. In [55], Kari and Zu proved that multi-shift universal cycles for \( B(n) \) exist.
A \((k^r, k^s; p, q)_k\) de Bruijn tori is a \(k\)-ary toroidal array of size \(k^r \times k^s\) with the property that every \(p \times q\) matrix appears exactly once contiguously on the tori. As an example, the following matrix is a \((4, 4, 2, 2)_2\) de Bruijn tori:

\[
\begin{bmatrix}
0 & 0 & 0 & 1 \\
0 & 0 & 1 & 0 \\
1 & 0 & 1 & 1 \\
0 & 1 & 1 & 1
\end{bmatrix}
\]

A de Bruijn sequence can be considered as a 1-dimension de Bruijn tori with \(s = 0\), \(r = n\), \(p = n\) and \(q = 1\). De Bruijn torus have been studied extensively [48, 66, 94] and have a variety of interesting applications including robot self-location [94], pseudo-random arrays [66], design of mask configurations for spectrometers [66] and cloth patterns [44].

In [18] and [65], Cock and Ma proved that \((k^r, k^s; p, q)_k\) de Bruijn torus exist when \(r = p\) and \(s = p(q - 1)\). Fan, Fan, Ma and Siu [28] further proved that \((2^r, 2^s; q, q)_2\) de Bruijn torus exist if and only if \(q\) is even. In [48], Hurlbert and Isaak proved when \(k\) is odd, or \(k\) is even and \(q \geq 10\), \((k^r, k^s; q, q)_k\) de Bruijn torus exist if and only if \(q\) is even or \(k\) is a perfect square. However, the existence of de Bruijn torus and constructions for other dimensions are generally not known.

We generalize the idea of de Bruijn tori to universal tori. A universal tori \((\ell, m, p, q)_k\) is a \(k\)-ary toroidal array of size \(\ell \times m\) with the property that the tori encodes \(p \times q\) distinct \(k\)-ary matrix exactly once contiguously on the tori. A multi-shift universal cycle can be transformed into a tori with similar properties as a universal tori. As an example, consider the following sequence:

\[
000110010100001100101.
\]

The sequence is a 2-shift universal cycle for \(B_5^2(4)\). The length 4 substrings in the sequence that start with the underlined characters contain each element in \(B_5^2(4)\) exactly once as follows:

\[
0001, 0110, 1001, 0100, 0000, 0011, 1100, 0010, 1010, 1000.
\]
By rearranging the characters of the 2-shift universal cycle as below, we obtain a tori where each \((2 \times 2)\) array is unique and represents an element in \(B^2_0(4)\) when we do not consider the wrap around in the vertical direction:

\[
\begin{array}{ccccccc}
0 & 0 & 1 & 0 & 0 & 0 & 1 \\
0 & 1 & 0 & 1 & 1 & 0 & 0 \\
\end{array}
\begin{array}{ccccccc}
1 & 0 & 1 & 0 & 1 & 1 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 \\
\end{array}
\]

This approach, however, does not create universal torus for all dimensions, and has the limitation of not taking account of the wrap-around in one direction. It would be interesting to further extend this approach to produce universal torus.
Bibliography


Appendix A

C code of optimized shift-based algorithm to generate a de Bruijn sequence over a binary alphabet

```c
#include<stdio.h>
int n,a[50],b[50];

//------------------------------------------------------
// If b[1..n] is a necklace, return the length of the
// longest aperiodic prefix; otherwise return 0
//------------------------------------------------------
int IsNecklace(){
    int i,p=1;
    for (i=2; i<=n; i++) {
        if (b[i-p] > b[i]) return 0;
        if (b[i-p] < b[i]) p = i;
    }
    if (n%p != 0) return 0;
    return p;
}

//------------------------------------------------------
// Return TRUE iff a[1..n] is 0^n
//------------------------------------------------------
int Zeros() {
    int i;
    for (i=1; i<=n; i++) if (a[i] == 1) return 0;
    return 1;
}

//--------------------------------------------------------------------------
int g() {
    int i,j;

    // Test if rotated string setting last bit to 1 is a necklace
    for (i=1; i<n; i++) b[i] = a[i+1];
    b[n] = 1;
    if (IsNecklace()) return 1;

    // Test if a[1..n] is a necklace; if so j=longest aperiodic prefix
```
for (i=1; i<=n; i++) b[i] = a[i];
j = IsNecklace();
if (j > 0) return j;

// Determine maximal suffix of form 0^j
for (j=0; j < n; j++) if (a[n-j] == 1) break;
return n - j;

void f() {
    int i, j, first_bit;

    // Test if rotated string setting last bit to 1 is a necklace
    for (i=1; i<n; i++) b[i] = a[i+1];
    b[n] = 1;

    // Rotate string left then assign new last bit
    first_bit = a[1];
    for (i=1; i<n; i++) a[i] = a[i+1];
a[n] = first_bit;
    if (IsNecklace()) a[n] = (a[n]+1) % 2;
}

void DB() {
    int i, j;

    for (i=1; i<=n; i++) a[i] = 0;
do {
        j = g();
        for (i=1; i<=j; i++) printf("%d", a[i]);
        for (i=1; i<=n; i++) b[i] = a[i];
        for (i=1; i<=n-j+1; i++) a[i] = b[i+j-1];
        for (i=1; i<j; i++) a[n-j+i+1] = b[i];
f();
    } while ( !Zeros() );
}

int main() {
    printf("Enter n: ");
    scanf("%d", &n);
    DB();
    printf("\n");
}
Appendix B

C code of optimized shift-based algorithm to generate a de Bruijn sequence over a general alphabet

```c
#include <stdio.h>
int n, k, a[50], b[50];

//---------------------------------------------------------------
int g_k()
{
    int j, t, p;

    for (j=1; j<=n; j++) a[n+j] = a[j];

    p = 1;
    t = 2; j = 2;
    do {
        t = t + p*((j-t)/p);
        j = t + 1;
        p = 1;
        while (j <= 2*n && a[j-p] <= a[j]) {
            if (a[j-p] < a[j]) p = j-t+1;
            j++;
            if (j-t+1 == n && a[j] < k && (a[j]+1 > a[j-p] || (a[j]+1 == a[j-p] && (n%p == 0)))) return t - 1;
        }
    } while (p * ((j-t)/p) < n);

    return t - 1;
}

//---------------------------------------------------------------
int f_k()
{
    int i, j, p=1;

    for (i=0; i<n; i++) a[i] = a[i+1];
    if (a[0] == k) {
        for (i=2; i<=n-1; i++) {
            if (a[i-p] > a[i]) {
                a[n] = k;
                return 1;
            }
        }
    }

    return t - 1;
}
```
if (a[i-p] < a[i]) p = i;

if (a[n-p] == 1 && p == 1) return 0;
else if (n%p > 0) a[n] = a[n-p];
else a[n] = a[n-p] - 1;

else a[n] = a[0] + 1;

return 1;

//--------------------------------------------------------------------------
// Generate a DB sequence for k-ary strings of length n in O(1) time per character
//--------------------------------------------------------------------------
void DB() {
    int i,j;

    for (i=1; i<=n; i++) a[i] = 1;
    do {
        j = g_k();
        for (i=1; i<=j; i++) printf("%d", a[i]);
        for (i=1; i<=n; i++) b[i] = a[i];
        for (i=1; i<=n-j+1; i++) a[i] = b[i+j-1];
        for (i=1; i<j; i++) a[n-j+1+i] = b[i];
    } while (f_k());
}

//------------------------------------------------------
int main() {
    printf("Enter n: ");
    scanf("%d", &n);
    printf("Enter k: ");
    scanf("%d", &k);
    DB();
    printf("\n");
}
Appendix C

C code of optimized weighted FKM construction over a binary alphabet

```c
#include <stdio.h>
int n,c,a[100];

void Gen(int t, int p, int w) {
    int i;

    if (t > n) if (n%p == 0) for (i=1; i <= p; i++) printf(\%d, a[i]);
    else {
        // Append 0
        a[t] = 0;
        if (a[t-p] == 0 && c-w < n-t+1) Gen(t+1, p, w);

        // Append 1
        a[t] = 1;
        if (a[t-p] == 1) Gen(t+1, p, w+1);
        else Gen(t+1, t, w+1);
    }
}

int main() {
    printf("Enter n: ");
    scanf("%d", &n);
    printf("Enter c: ");
    scanf("%d", &c);

    a[0] = 0;
    if (n >= c) Gen(1, 1, 0);
    printf("\n");
}
```