Investigations On Nonlinear Streamcipher Systems: Construction and Evaluation Methods

Cees J. A. Jansen
Dit proefschrift is goedgekeurd door de promotor prof. dr. ir. D.E. Boekee
Investigations On Nonlinear Streamcipher Systems: Construction and Evaluation Methods

Cees J. A. Jansen
To Maria, Richard and Irene
Preface

Soon after I got involved in cryptology in 1980, I learned that there was a big gap between the theory and the practical application of cipher systems. In particular, I found that there existed little theory on the nonlinear aspects of streamcipher systems, whereas after reading Golomb's book my interest was definitely raised. Through a detour, caused by the many interesting and relevant problems of key management, I arrived at the topics described in this thesis.

Several results described here were presented at various symposia. The results of Section 2.2 were presented at Eurocrypt 86, Linköping, Sweden, and Section 2.3 was presented at Eurocrypt 87, Amsterdam, The Netherlands. The results of Chapter 5 were presented at the 1988 Benelux Symposium on Information Theory, Mierlo, The Netherlands and at the 1988 IEEE International Symposium on Information Theory, Kobe, Japan. Chapter 5 and Section 6.2 are co-authored by Ir. W.G. Franx.

I want to express my deep gratitude to Prof. D.E. Boekee, who not only encouraged me to 'write it all down', but also spent many lengthy and stimulating discussions to carry us to the essence of this work. I thank Ir. M.W. van Rijswijk, Dr.Ir. C.L.M. van Pul and Ir. W.G. Franx for proof reading the concept and for their many valuable suggestions. I want to express special thanks to Ir. W.G. Franx for the fruitful co-operation on the DeBruijn sequence generator and the switch controlled feedback shift register and, finally, for implementing the 'dawg'. Thanks are also due to Philips Usfa B.V. for the assistance in the printing of the document.
Summary

This thesis deals with the nonlinear aspects of streamcipher systems. New construction methods are proposed and powerful new evaluation methods are presented.

The mixer is identified as the important part of streamcipher systems when the protection against active eavesdropping is considered. This mixer usually is depicted as a simple linear device without memory. Efficient nonlinear mixers are proposed, based on the use of JK-flipflops. For blockciphers various new modes of use are presented and analyzed with respect to their protection against active eavesdropping. The best mode, in this respect, is called the OFBNLF mode. This mode uses simple nonlinear operations on the plaintext and is very effective and easy to implement.

The problem of finding the absolutely shortest (possibly nonlinear) feedback shift register, which can generate a given sequence with characters from some arbitrary finite alphabet, is considered. To this end, a new complexity measure is defined, called the maximum order complexity. A new theory of the nonlinear feedback shift register is developed, concerning elementary complexity properties of transposed and reciprocal sequences, sequences regarded over power alphabets, dual sequences, and feedback functions of the maximum order feedback shift register equivalent. The non-existence of self-complementary and self-reciprocal DeBruijn sequences is shown.

Blumer's algorithm is identified as a powerful tool for determining the maximum order complexity profile of sequences, as well as their period, in linear time and memory.

The complexity of random sequences is considered from a theoretical point of view. The jump behaviour of the maximum order complexity profile is completely analyzed. Unlike the linear complexity profile, which always makes a jump to \( l - c_l \), if a jump occurs when extending the sequence with a character, the maximum order complexity profile can also jump to values less than that. It is also shown that the problem of counting the number of
sequences of given length and complexity, is equal to the difficult problem of counting incomplete paths in a DeBruijn graph of given order. Statistical models of maximum order complexity are used to predict the expected complexity of random sequences. The result is that the expected maximum order complexity grows with twice the logarithm of the sequence length. Statistical experiments strongly support the validity of the models.

For many purposes, including the analysis of nonlinear feedback functions of feedback shift registers, a new view at the algebraic normal form (ANF) is presented. The algebraic normal form transform (ANFT) is introduced, based on the so-called natural ordering of product terms in the ANF. This ANFT is a fast transform that can easily be implemented in hardware or software. It is shown that the ANFT can be applied to random binary functions for the purpose of function complexity analysis.

An algorithm for joining cycles in the binary DeBruijn graph is presented. This algorithm effectively acts as an additional nonlinear feedback function, which can be used with any non-singular feedback shift register. By using appropriate linear feedback functions, the algorithm can generate $O(2^{2n}/\log^2 n)$ binary DeBruijn sequences, requiring only $3n$ bits of storage and at most $4n$ shifts of the feedback shift register to generate every bit of the sequence.

The switch controlled feedback shift register is introduced as a means for generating binary sequences with good complexity properties. Relevant parameters can be chosen such that sequences are obtained which have a maximum order complexity of approximately twice their length. Statistical properties indicate that a correlation attack is possible, however, with high computational complexity. It is also shown that many known construction methods for sequence generators, can be regarded as extensions of the described method.

A second approach to obtaining sequences with good complexity properties is the construction of run permuted sequences. Starting with a DeBruijn sequence of given order, a class of sequences is constructed by permuting the runs of ones and zeroes. It is shown that this class contains all DeBruijn sequences of the given order and many other sequences, all satisfying Golomb’s first and second randomness postulates. It is demonstrated that all the sequences in the class can be generated by means of enumerative coding schemes.

Streamciphers are treated in an information theoretic manner by modelling the keystream generator as a probabilistic source. This keystream source uses a fixed periodic sequence, which is started in a secret random
phase, to emit successive characters. To judge the cryptographic quality of this keystream source, so-called uncertainty profiles are introduced. These uncertainty profiles clearly illustrate the weakness of the keystream generator.

A more complicated source model of a keystream generator is introduced, which is a composite source model. This second model is shown to be superior to the first model from a cryptographic point of view. The conclusion is that keystream generators should be able to generate an ensemble of sequences, from which a particular one is chosen at random, governed by a secret key. The initial phase, however, should be chosen at random, but should not be regarded as secret key information.
## Contents

Preface v
Summary ix
Contents xiii
List of Tables xvii
List of Figures xix

### Introduction 1

#### 1 Cipher Systems 5
  1.1 An Overview 5
  Symmetric versus Asymmetric Ciphers 5
  Blockciphers versus Streamciphers 7
  Key Management 8
  1.2 Streamciphers 10
  Synchronization 12
  1.3 Security of Cipher Systems 12
  Streamcipher Quality 13

#### 2 On Data Integrity and Active Eavesdropping 15
  2.1 Introduction 15
  2.2 Active Eavesdropping with Bit-Streamciphers 18
  2.2.1 Text Feedback and Double Encryption 19
  2.2.2 Two Stage Encryption with a JK-flipflop 20
  2.3 Modes of Blockcipher Algorithms and their Protection Against Active Eavesdropping 23
  2.3.1 Modes of Blockcipher Algorithms 23
2.3.2 Implementations of the OFBNLF Mode .......... 26
2.4 Conclusions ........................................ 28

3 The Shortest Feedback Shift Register Equivalent: A Different View at Complexity of Sequences 29
3.1 The Maximum Order Complexity of Sequences .......... 31
  3.1.1 The Complexity of Transposed and Reciprocal Sequences ........ 35
  3.1.2 The Complexity of Periodic Sequences Regarded Over Power Alphabets .......... 37
  3.1.3 Feedback Functions of the Maximum Order FSR Equivalent ........ 38
  3.1.4 Properties of Feedback Functions in \( \Phi_x \) .......... 42
  3.1.5 Dual Sequences .................................. 44
3.2 The Maximum Order Complexity Profile .......... 50
  3.2.1 The Directed Acyclic Word Graph ........ 50
        Blumer's Algorithm ........ 54
  3.2.2 The DAWG and the Complexity Profile .......... 55
  3.2.3 The Typical Complexity Profile .......... 58
3.3 The Complexity of Random Sequences .......... 66
  3.3.1 The Behaviour of the Maximum Order Complexity Profile ........ 66
  3.3.2 Numbers and their Relation with DeBruijn Graphs .......... 73
  3.3.3 The Complexity Table Reconsidered .......... 78
  3.3.4 Bounds on \( L_j^+ \) and \( L_j^- \) .................. 79
3.4 The Complexity of Random Sequences; a Statistical Approach .......... 81
  3.4.1 Relation with Ziv-Lempel Complexity .......... 81
        3.4.1.1 Lowerbound on the Expected Complexity .......... 82
        3.4.1.2 Upperbound on the Expected Complexity .......... 82
  3.4.2 Heuristic Statistical Models .......... 83
        3.4.2.1 Birthday Statistics .......... 83
        3.4.2.2 Periodic Random Sequences .......... 86
        3.4.2.3 Random Sequences .......... 88
        3.4.2.4 Mutually Excluding m-tuples .......... 91
        3.4.2.5 Conclusions on Heuristic Statistical Models .......... 96
  3.4.3 Experimental Results .......... 97
        3.4.3.1 Complexity versus Sequence Length .......... 97
        3.4.3.2 Complexity versus Alphabet Cardinality .......... 98
3.5 Conclusions .......................................................... 99

4 The Algebraic Normal Form of Arbitrary Functions over Finite Fields .......................................................... 101
4.1 Introduction .......................................................... 101
4.2 The ANF Transform over $GF(2)$ .................................. 102
4.3 Fast Transform Implementation ................................... 105
4.4 Random Binary Functions ......................................... 106
4.5 The ANF Transform over $GF(q)$ ................................. 109
4.6 Conclusions .......................................................... 114

5 Generating Binary DeBruijn Sequences .................................. 115
5.1 Introduction .......................................................... 115
5.2 Basic Concepts ....................................................... 116
5.3 A Universal Algorithm for Joining Cycles in the DeBruijn Graph .......................................................... 118
5.4 Efficient Use of the Cycle Joining Algorithm .................. 123
5.5 Synthesis of the FSR equivalent .................................. 126
5.6 Conclusions .......................................................... 127

6 Binary Sequence Generators Based on Source Coding Algorithms .................................................. 129
6.1 Introduction .......................................................... 129
6.2 Sequences Generated by a Switch Controlled Feedback Shift Register .................................................. 130
6.2.1 The Ziv-Lempel Data Compression Algorithm .......... 130
6.2.2 Generating Sequences with the Ziv-Lempel Decoding Algorithm .................................................. 131
6.2.2.1 The Period of the Generated Sequence ............ 132
6.2.2.2 The Complexity of $z$ .................................... 135
6.2.2.3 Statistical Properties .................................... 137
6.2.3 Improvement of the Generator ................................ 138
6.3 Run Permuted Sequences ........................................... 140
6.3.1 Run-Length Notation of Periodic Sequences ............. 140
6.3.2 Classes of Run Permuted Sequences ....................... 141
6.3.3 Properties of the Sequence Class $C_n$ .................... 145
6.3.4 Generation of Run Permuted Sequences ................... 147
6.4 Conclusions .......................................................... 149
### Contents

**7 Information Theory of Shift Register Sequences** 151  
7.1 Introduction ........................................... 151  
7.2 The Character Uncertainty Profile .................. 152  
7.3 The Phase Uncertainty Profile .................... 155  
7.4 Obtaining High Equivocation of Key ............... 157  
7.5 Conclusions ........................................... 161  

**Bibliography** 163  
**Appendix** 169  
**Curriculum Vitae** 191  
**Samenvatting** 193
List of Tables

2.1 Success rates for some values of \( L \) and \( N \) .............................. 23
2.2 Success rates of cyclic perms and real additions ......................... 28

3.1 The number of binary sequences with complexity from 0 until \( l - 1 \), for lengths \( l \leq 24 \) .................................................. 60
3.2 The number of periodic binary sequences with complexity from 0 until \( p - 1 \), for periods \( p \leq 24 \) .......................................... 62
3.3 Statistical moments of complexity distribution ............................ 64
3.4 Statistical moments of complexity distribution (periodic) ............. 65
3.5 Backwards relationship for the binary case .................................. 72
3.6 Values of \( D_{j,n}^{\alpha} \) in the binary case .................................. 74
3.7 Some values of \( \Delta_j^l \) in the binary case ................................ 77
3.8 Upperbound on the number of draws ....................................... 85
3.9 Upperbound on expected alphabet cardinality ............................. 88
3.10 Estimates of expectation and deviation of complexity for various sequence lengths ................................................ 97
3.11 Estimates of expectation and deviation of complexity for various alphabet cardinalities .............................................. 98

4.1 Group table of all binary non-singular \( 2 \times 2 \) matrices .................. 105

5.1 Number of DeBruijn sequences obtained with linear feedback functions .......................................................... 125

6.1 Linear complexity of \( d \) ..................................................... 135
6.2 Maximum order complexity of a single period of \( z \) ..................... 135
6.3 Correction factor \( \rho_n \) of equation (6.18) ................................. 144
**List of Figures**

1.1 General form of a streamcipher. .................................................. 10

2.1 Active eavesdropping in a symmetric cipher system. .......... 16
2.2 Example of a streamcipher. ......................................................... 18
2.3 Streamcipher with text feedback. .................................................. 19
2.4 Double encryption with a text feedback system. .................. 20
2.5 Two stage encryption with a JK-flipflop. ...................... 21
2.6 Multi stage encryption with JK-flipflops. .................. 22
2.7 CBCPD mode of a blockcipher. ...................................................... 24
2.8 OFBNLF mode of a blockcipher. ............................................... 25
2.9 Different real additions as mixer in OFBNLF. .............. 26
2.10 Cyclic permutations of the plain text in OFBNLF. .......... 27

3.1 The feedback shift register. ....................................................... 31
3.2 DAWG of 110100. ................................................................. 52
3.3 Splitting a DAWG’s node. .......................................................... 53
3.4 The DAWG of a periodic string. ............................................. 57
3.5 FSR generating sequence of Lemma 3.21. ......................... 68
3.6 Jumps in the complexity profile. ........................................... 71
3.7 Backwards relationship of complexity values. ................. 72
3.8 Estimates of expectation and deviation of complexity versus alphabet cardinality. ............................................. 98

4.1 ANFT order reduction. .......................................................... 106
4.2 Binary ANFT for $n = 3$ in recursive form. ..................... 107
4.3 Butterfly form of binary ANFT. ................................................ 107
4.4 Wire crossing as a transform in $GF(2)^2$. ..................... 113
4.5 Equivalent wire crossing in $GF(4)$. ...................................... 113
List of Figures

5.1 DeBruijn graphs of degrees 3 and 4. ........................................... 117
5.2 Joining two cycles in an adjacency quadruple. ............................. 118
5.3 Cycle joining algorithm as additional feedback function. ............... 121

6.1 Ziv-Lempel decoding algorithm as a FSR. ...................................... 131
6.2 A switch controlled feedback shift register. ...................................... 132
6.3 Feedforward equivalent of $d$ generator for $r = 1$. ......................... 136
6.4 Equivalent JK–flipflop generator for $r = 1$. ..................................... 137
6.5 Structure of generalised multiplexed FSR's. ..................................... 138
6.6 Structure of feedforward filtered FSR's. .......................................... 139
6.7 Ternary code tree for all permutations of $abc$. ......................... 148
6.8 Generator structure for run permuted sequences. ........................ 149

7.1 Streamcipher with secret initial phase. ........................................ 153
7.2 CUP and SUP of the ensemble of all DeBruijn sequences of order 4. ................................. 161
7.3 Key multiplicity with a FSR based keystream generator. .................. 162
Introduction

This decade may be considered as the adolescence of the information age. Micro electronics has had an enormous impact on society in the way information is gathered and processed. Distributed computing facilities, like powerful personal computers, and distributed databases connected together within huge networks, have caused a tremendous worldwide flow of information. Gradually, people have become aware of the fact that the secure, reliable and efficient handling of information is of vital importance. Information management was born.

In particular the security aspects of information management, such as secrecy, privacy protection, integrity and authentication, have become important areas of research and development. For more than one reason one could say that we have witnessed what may be described as "cryptography goes public". First, many people have become publicly involved in the research, development, design, manufacture and use of cryptographically secured equipment. Second, the practical use of so-called public key cipher systems has really taken a start, due to the availability of usable hardware. This does not mean the obsolescence of what is widely known as classical ciphers. On the contrary, these symmetric or secret key cipher systems are still by far the most efficient and widely used, especially in applications requiring high throughput rates.

Symmetric cipher systems are usually divided into blockciphers and streamciphers. Again in practice, streamcipher systems make up the vast majority of implemented symmetric cipher systems. The theory of these streamcipher systems has always been somewhat neglected. Although the linear theory has received much attention, the theory of nonlinear streamcipher systems, used so often in practice, has known little progress for a long time. It is in this context that the research, reflected in this thesis, was carried out.
In this thesis nonlinear aspects of streamcipher systems are considered. The analysis and synthesis of nonlinear mixers and highly secure keystream sequences based on nonlinear feedback shift registers is highlighted. The introduction of a new tool for judging the randomness of sequences forms the core of this thesis.

This thesis is not completely self-contained: a certain mathematical maturity is expected. In particular the mathematics of finite fields and an introductory knowledge of linear feedback shift registers are presupposed. These topics can be found e.g. in the books of Lidl & Niederreiter [Lidl 83] and VanTilborg [Tilb 88] respectively.

Chapter 1 introduces cipher systems and presents an overview of important notions such as symmetric versus asymmetric ciphers, blockciphers versus streamciphers and key management. The streamcipher system in its most general form is focussed on and its essential parts are identified. Also a brief introduction in the security aspects of cipher systems is given.

In Chapter 2 the problem of data integrity and the protection against active eavesdropping with streamciphers is considered. It is shown that the classical streamcipher having a linear mixer, i.e. the keystream is added to the plaintext to obtain the ciphertext, is vulnerable under an active attack. Both for bit-streamciphers and character streamciphers efficient methods are given which provide for protection against manipulation of the ciphertext. In particular for blockciphers various new modes of use are investigated and one mode, called OFBNLF mode, is shown to be very effective and easily implementable.

In Chapter 3 a new complexity measure for sequences with characters from arbitrary finite alphabets is introduced as the maximum order complexity. This complexity measure – in this thesis often called ‘complexity’ – denotes the minimum number of memory cells a feedback shift register must at least have to generate a given sequence. It is to be understood that the feedback function of the shift register is not restricted to linear functions, but may be any memoryless mapping.

The basic properties of maximum order complexity are explored and the complexity of transposed and reciprocal sequences, as well as that of periodic sequences regarded over power alphabets is investigated. Feedback functions of the maximum order feedback shift register equivalent are considered and their properties examined. Dual sequences are introduced and the notions of self-dual, self-complementary and self-reciprocal sequences
are discussed.

The maximum order complexity profile is introduced and an algorithm is identified which is able to determine this complexity profile of any sequence in linear time and memory. Using a computer program implementation of this algorithm the typical maximum order complexity profiles of binary random sequences of length up to 24 bits are determined.

The maximum order complexity of random sequences is also considered from a theoretical point of view. To this end, a theory is developed explaining the typical behaviour of the maximum order complexity profile. The relation between the total number of sequences of given complexity and the number of incomplete paths in the DeBruijn graph is shown. Bounds on the number of sequences of given length and complexity are determined.

A statistical approach is applied to the maximum order complexity of random sequences. The relation with the Ziv-Lempel complexity measure is explained and bounds on the expected complexity are derived. Various statistical models are employed to model the progression of complexity with the sequence length. Finally some experimental results are given and compared with the theory.

Chapter 4 presents a new view at the algebraic normal form (ANF) of functions over finite fields, based on a matrix structure, which is shown to hold for every finite field. The ANF transform (ANFT) is introduced for the binary case, resulting in a fast transform implementation analogous to the fast fourier transform. The ANFT is then applied to random binary functions for the purpose of function complexity analysis. Finally, the ANF transform over $GF(q)$ is derived.

Chapter 5 aims at presenting a new algorithm for the generation of binary DeBruijn sequences. DeBruijn sequences are introduced and an algorithm for joining cycles in a DeBruijn graph is presented. It is shown that under certain conditions this algorithm can be used very efficiently to produce a great number of DeBruijn sequences, requiring only a small amount of time and memory. It is also shown that the algorithm can be adapted to synthesize the maximum order feedback shift register equivalent of a sequence of given length and maximum order complexity.

The purpose of Chapter 6 is to describe binary sequence generators that generate sequences with good maximum order complexity properties. The first sequence generator emerges from the Ziv-Lempel data compression algorithm and uses a switch controlled feedback shift register. The prop-
erties of sequences generated in this way such as period, linear complexity, maximum order complexity and statistics are investigated. Improvements of the generator are mentioned and a classification of some related, known, sequence generators is given.

The second sequence generator is based on run-length and enumerative source coding principles. The runs of a DeBruijn sequence are permuted hence obtaining an entire class of sequences which satisfy Golomb's first and second randomness postulates perfectly. The properties of this sequence class, in particular the maximum order complexity of its sequences, are considered. It is also shown how to generate these sequences efficiently.

In Chapter 7 the information theory of shift register sequences is developed. A probabilistic model of a keystream generator is introduced and its cryptographic strength analyzed. To this end, the character- and phase uncertainty profiles are defined. In order to obtain high equivocation of key, a second keystream generator model is introduced and the sequence uncertainty profile defined. Based on the behaviour of the uncertainty profiles a keystream generator structure is proposed.
Chapter 1

Cipher Systems

1.1 An Overview

A cipher system, or cipher for short, is defined by Shannon [Shan 49] as a family of transformations $E = \{E_k | k \in K\}$ of a message space $M$ into the cryptogram space $C$. For all $k \in K$ the transformation $E_k$ corresponds to enciphering with the key $k$ which is in the key space $K$ of the cipher. The cardinality of this key space, $|K|$, is often referred to as the key multiplicity or key diversity of the cipher. Also, for all $k \in K$ the transformation $E_k$ must be reversible (non-singular), so that unique deciphering is possible if the key $k$ is known. The inverse transformation $E_k^{-1}$ of $E_k$ is often denoted by $D_k$.

In order for a cipher $E$ to be of any practical use to cryptography, there have to exist algorithms that allow an efficient calculation of the values $E_k(m)$ and $D_k(c)$ for all $k \in K$, $m \in M$ and $c \in C$. Usually one assumes that there exists an encryption algorithm which allows for an efficient encryption based on knowledge of the key alone, while there only might exist a decryption algorithm which allows for an efficient decryption if and only if some additional information is available. This last observation leads to the division of ciphers into symmetric ciphers (no additional information needed) and asymmetric ciphers (additional information is needed).

Symmetric versus Asymmetric Ciphers

A cipher is called symmetric if there exist algorithms which allow for an efficient encryption and decryption from knowledge of the key alone. Because of this property the particular key used in the enciphering process of
a symmetric cipher should be kept secret from those who are not allowed to read the message. For this reason symmetric ciphers are often called *secret key* ciphers. Another name that is frequently used is *classical* cipher. An example of a classical cipher is the *Data Encryption Standard* (DES) [Konh 81], which is a *product* cipher consisting of rather simple *transpositions* and *substitutions* [Mass 88].

A cipher is called an *asymmetric* cipher if there exists an algorithm which allows for an efficient encryption based on knowledge of the key alone, while there only exists an algorithm which allows for an efficient decryption if besides knowledge of the key one is also allowed to use some secret additional side information. If this secret information is not available, decryption is assumed to be computationally infeasible for virtually all keys from the key space and virtually all the cryptograms from the cryptogram space, even if the encryption algorithm is known. Because of this property of asymmetric ciphers there is no need for keeping the particular key used for encryption secret, as long as the additional side information, needed for efficient decryption, is kept secret. Therefore, asymmetric ciphers are often referred to as *public key* ciphers. Also *two-key* cipher is used frequently, referring to the existence of two keys for such a cipher: the public key used for encryption and the secret key (the secret additional side information) used for decryption. An example of an asymmetric cipher is the *RSA* cipher system, invented by Rivest, Shamir and Adleman [Rive 78], which is based on the power function modulo a composite number.

In general one assumes that the encryption algorithm of an asymmetric cipher is known to everyone. Each user who wants to communicate securely, generates a public key plus corresponding private (secret) key and publishes the public key in some kind of public directory, while keeping the private key secret. If user *A* wants to send a message to a user *B* in a secure way, he enciphers the message under the public key of *B* taken from the public directory, after which he sends the cryptogram to user *B*. After reception, user *B* deciphers the cryptogram with his private key. In order to protect against an adversary *X*, who acts as *B* towards *A* and as *A* towards *B*, the integrity of the public directory should be guaranteed by some authority which is trusted by every user. It is exactly this property of public key ciphers which has not received the attention it deserves. It is often heard that the key management problem associated with using a cipher system is solved by means of a public key cipher for key distribution. However, the certified distribution of public keys is usually not mentioned, but is in many cases performed in a classical way, the only difference with secret key
distribution being that confidentiality is not necessary but authenticity is.

**Blockciphers versus Streamciphers**

The plaintext symbols to be enciphered are usually divided into fixed size blocks for practical reasons. These blocks are then enciphered sequentially. There are two essentially different ways to encipher the entire plaintext. One could encipher each plaintext block independently with the same encipherment transformation, in which case one speaks of a *Blockcipher*. In fact blockciphers are simple substitution ciphers, i.e. memoryless mappings from the space of plaintext blocks into the space of ciphertext blocks. Consequently, identical plaintext blocks give rise to identical ciphertext blocks and therefore the blocklength should be large to prevent simple cryptanalysis. Examples of blockciphers are DES and RSA.

Another way to encipher the plaintext is to encipher each plaintext block with a varying encipherment transformation, where the variation is on a block sequence base such as time or storage location. Therefore, identical plaintext blocks usually do not result in identical ciphertext blocks. These ciphers are called *Streamciphers*. In streamciphers the variation of the encipherment transformation inherently implies the presence of memory, whose internal state changes with every subsequent block according to some rule. Unlike with blockciphers the length of the plaintext blocks need not be large to have a secure cipher system. For this reason one usually speaks of plaintext characters instead of blocks. For example a blocklength of 1 binary digit is widely used. The presence of memory is in fact the essential distinction between blockciphers and streamciphers. Examples of streamciphers are the DES in any of its feedback modes, the *running key generator* (RKG) or *Vigenère* cipher [Konh 81] and the *one-time pad* or *Vernam* cipher [Davi 84].

In practice blockciphers are of limited use and are often employed in such a way that a streamcipher is obtained. In this context one speaks of the *modes of operation* or the *modes of use* of a blockcipher. Well-known in this respect are the *ECB, CBC, CFB* and *OFB* modes of operation see e.g. [Meye 82,Davi 84]. The ECB mode is the native mode of a blockcipher as in this mode the plaintext is divided into blocks and each block is enciphered independently.

From the point of view of modes of use, streamciphers and blockciphers may be seen to be equivalent, i.e. blockciphers can be used as streamciphers and vice versa. Obviously blockciphers can always be used as streamciphers
by adding memory, however, it is not always possible to use a streamcipher as a blockcipher efficiently. Despite all this, one perceives the tendency to come up with entirely different designs and implementations for both types of cipher systems.

**Key Management**

Many cipher systems have been published in the literature, and their properties and their relative strengths (i.e. the degree of difficulty involved in recovering the plaintext, or even the key, from the cryptogram) have been widely discussed. Much less attention has been paid to the problem of obtaining the necessary key material at the encryption and decryption ends.

It will be apparent that certain problems are inherent in the handling of key material within a cipher system. On the one hand the need to preserve the secrecy of certain key material is essential; on the other hand keys must be generated, distributed, copied, transported, stored, updated, published, authenticated and, finally, destroyed. **Key management** is defined as the secure and efficient execution of all the aforementioned operations. The problems associated with these operations, of course, play an important role during the installation of a cryptographic system; in addition – because of the limited period of validity of the key material, the dynamics of the system and the constant possibility of compromise and collusion – they are also continuously present throughout the systems operational lifetime.

In designing a key management system capable of providing secure and efficient solutions to these problems, many factors must be taken into account. Some of these factors are:

1. **Symmetric or asymmetric cipher system**
   As was mentioned in the beginning of this chapter.

2. **Types of network**
   There are many ways of characterising communications networks, e.g. data rate, reliability, signalling methods, switching (circuit, message or packet), etc. For the purpose of designing key management systems, the following classification seems appropriate:
   
   a. Multiple subscriber networks with end-to-end security, e.g. the public telephone service.
   
   b. Point-to-point networks with link security, e.g. terminal to host connections.
c. Broadcast networks such as radio or telex with store-and-forward switches.


3. Security services
The combination of a cipher system and a key management system offers certain security services to its users. These services include:

a. Authenticity
b. Confidentiality
c. Integrity
d. Non-repudiation

Not all these security services have to be available in every cryptographic system. The provision of specific services depends largely on the user requirements and on what major threats are perceived to be applicable.

Nowadays a variety of key management systems can be encountered in practice. These key management systems consist of (combinations of) the following principles:

1. On-line Key Distribution Centre
   Usually called KDC, the key distribution centre shares unique secret keys with all the subscribers and generates and distributes session keys whenever a pair of subscribers want to have a secure communication [Jone 85, Diff 88].

2. Off-line Key Distribution Centre
   The off-line KDC generates unique communication keys for each pair of subscribers. All the keys necessary for each subscriber are then distributed via a secure channel and stored inside the subscriber's terminal or key storage device [Jans 84].
   To reduce the amount of storage required for large networks, the use of key storage reduction schemes has been proposed which offers a tradeoff between collusion resistance and key storage requirements [Jans 86c].

3. Public Key Distribution Methods
   Asymmetric ciphers are used for secret key exchange or the Diffie-Hellman scheme is employed to establish a common secret key [Diff 88]
Together with these methods mechanisms are available to support a decentralised key management such as automatic key selection based on key signatures \cite{Jans 86b}.

Experience has demonstrated that, too often, the key management system represents a heavy financial or procedural burden for the owners or operators of a cryptographic system. It may be welcomed that in designing modern cryptographic equipment this problem has received much attention \cite{Jans 87c}.

1.2 Streamciphers

As was already mentioned in Section 1.1 streamciphers encipher the plaintext character by character with a varying encipherment transformation. The most general form of a streamcipher is given by the following expressions:

\[
\begin{align*}
    c_n &= E_K(k_n, \ldots, k_{n-1, k}; p_n, \ldots, p_{n-i_2}; c_{n-1}, \ldots, c_{n-i_2}), \\
    p_n &= D_K(k_n, \ldots, k_{n-1, k}; c_n, \ldots, c_{n-i_3}; p_{n-1}, \ldots, p_{n-i_2}), \\
    k_n &= f_K(k_{n-1}, \ldots, k_{n-1, k}; p_{n-1}, \ldots, p_{n-i_3}; c_{n-1}, \ldots, c_{n-i_3}),
\end{align*}
\]

where \( c_n, p_n \) and \( k_n \) denote the \( n \)th ciphertext, plaintext and keystream character respectively. This streamcipher is depicted in Figure 1.1. As can be seen the streamcipher comprises three parts, viz. \( E_K \), \( f_K \) and the
memory cells containing delayed plaintext, keystream and ciphertext characters. The $E_K$ part is usually called the mixer, whereas $f_K$ together with the $k$-memory forms what is called the running key generator. Both the mixer and the running key generator can be dependent on some secret key information. The contents of the memory cells is often called the state of the streamcipher.

It should be noted that $f_K$ need not be a reversible function; in fact it usually never is. However, $E_K$ must always be reversible with respect to $p_n$ and $c_n$. Moreover, the character alphabets $P$, $K$ and $C$, to which $p_n$, $k_n$ and $c_n$ belong, need not all have the same cardinalities; in particular $|K| \geq |P|$ and $|C| \geq |P|$.

Because of the relation between $p_n$ and $c_n$, it suffices to use either $p_{n-1}, \ldots, p_{n-i}$ or $c_{n-1}, \ldots, c_{n-i}$ to obtain plaintext dependency in the mixer or the running key generator. The difference is that, when using the plaintext characters $p_{n-1}, \ldots, p_{n-i}$ only, the encipherment operation forms a feedforward system and the decipherment operation forms a feedback system, whereas when using the ciphertext characters $c_{n-1}, \ldots, c_{n-i}$ only, this is vice versa. Therefore, channel errors will have a different effect on both systems, i.e. the error propagation will be infinite or finite respectively (comparable to the use of IIR and FIR digital filters in communication systems [Rabi 75]). For this reason plaintext dependency is usually not considered in practice.

If there is no dependency on the ciphertext nor on the plaintext, as mentioned above, the cipher is what is usually called a RKG cipher. The RKG part in this case is called an autonomous keystream generator, as its state depends only on its previous state(s) and the mixer is memoryless. In classical systems this mixer usually consists of a simple linear operation such as addition. In Chapter 2 it is shown that there exist much better mixers, which allow for data integrity by the use of nonlinear operations $E$, or by the use of $E$ with cipher feedback. The great advantage of RKG ciphers is that $f_K$ can be designed such as to obtain a guaranteed (large) cycle length, although RKG's are known which lack this property, such as the DES in OFB mode.

Shift register generators with nonlinear feedforward and nonlinear combining function as described e.g. by Rueppel [Ruep 84] and by Siegenthaler [Sieg 83] can be seen as special cases of the general streamcipher system, where $f_K$ is linear and $E_K$ is linear in $p_n$. When using a DeBruijn sequence generator (see Chapter 5), however, $f_K$ is necessarily nonlinear.

At the other end there are streamcipher systems with ciphertext de-
pendency only and no RKG part. As an example the DES in CFB mode has $\text{DES}$ as $f_K$ and addition modulo-2 as $E_K$. In CBC mode, however, the DES has no $f_K$, but $E_K$ is addition modulo-2 plus $\text{DES}$ on the result.

**Synchronization**

The cryptogram, produced by a streamcipher, will in general depend on the particular key $K$ that is used, as well as on the initial contents of the memory cells of the streamcipher. Clearly, the state of the autonomous part, $k_{n-1}, \ldots, k_{n-i}$, must be identical for the encipher and decipher operations. However, if feedback is used with the decipher operation, i.e. $p_n$ depends on $p_{n-1}, \ldots$, then the entire state of the streamcipher should be identical for the encipher and decipher operations. If, on the other hand, there is no feedback with the decipher operation, the state $c_{n-1}, \ldots, c_{n-i}$ needs not be identical for both operations, as this state will automatically assume the right value whenever $i$ consecutive correct ciphertext characters have been received. Only if there is neither feedback with the decipher operation nor an autonomous part can the entire states differ arbitrarily.

The process of adjusting the states at the encipher and decipher sides to one and the same value is generally known as (crypto) synchronization. In this context usually the distinction is made between synchronous and self-synchronizing streamciphers, denoting whether or not the states at the encipher and decipher sides must initially have the same value or not. Self-synchronizing streamciphers have the disadvantage that errors in the received ciphertext propagate through a number of memory cells, thereby causing successive plaintext characters to be in error; a phenomenon called error propagation or error extension.

It is customary to choose the initial state of a streamcipher at random, in order to avoid that identical messages yield identical cryptograms, in which case one effectively has created a blockcipher again.

Practical cipher systems can employ various synchronization mechanisms, depending on the characteristics of the communications channel and signalling formats used. As with key management, synchronization forms a major area in the design of practical streamcipher systems.

**1.3 Security of Cipher Systems**

Although cipher systems are studied for their nice mathematical structures or interesting information theoretic aspects, they are employed in practice
to provide security in several areas, such as:

- **Transec**
  The secure (reliable) transmission of information, i.e. counter measures against jamming. An example is a frequency hopping spread spectrum system, where pseudo-random sequences are used to obtain an unpredictable pattern of transmission frequencies [Torr 81,Jans 88].

- **Comsec**
  The secure communication of information, i.e. the protection against eavesdropping, both active and passive.

- **Compusec**
  The secure handling of information by computer systems, i.e. file encryption, access control, process authentication, multilevel security.

To what extent this security is realized, depends on the security services offered by the cipher system. Moreover, this security also depends on the quality or strength of the cipher.

A cipher system can have the strength to successfully withstand attacks of various kinds. One usually distinguishes between active and passive attacks. With a passive attack the adversary can have knowledge of the ciphertext only, or he can additionally also know the corresponding plaintext. The adversary's objective then is to regain the plaintext or even the secret key used. With an active attack the so-called active eavesdropper manipulates the ciphertext or even chooses the ciphertext in order to send fake messages or to impersonate an entity using the system.

More often than not, it is impossible to prove the strength of a security service offered by a cipher system. If, after thorough cryptanalysis, it seems that the only attack to break the cipher is by exhaustive search of an effective keyspace of sufficient dimension, the cipher is often accepted as being secure. A good example of this forms the DES. Provable secure ciphers are subject of research since the last few years, but no practical cipher has been described yet (Rip van Winkle cipher) [Mass 85]. To this end, we cannot but only endorse H.W. Lenstra jr.'s statement made in the Dutch newspaper “Trouw” of 19 October 1988: “Cryptography is the fruit of our ignorance”.

**Streamcipher Quality**

It was the impracticability of the one-time-pad that led to streamciphers based on running key generators. The perfect secrecy of the one-time-pad
[Shan 49] is approached by not using a random keystream, but rather a keystream generated by some finite state device, acting on a finite length, secret, randomly chosen key. This keystream, produced by a running key generator, should resemble a random keystream as much as possible. In particular, the unpredictability of successive keystream characters should be maintained as long as possible. It turns out that perfect statistical properties and unpredictability are not equivalent, the best example being sequences generated by linear feedback shift registers.

Many people have studied this, seemingly difficult, controversy. Well-known in this respect are Golomb’s randomness postulates [Golo 67], which measure the randomness of a periodic binary sequence, viz. the disparity between ones and zeroes within one period, the run-length distribution and the number of values assumed by the periodic autocorrelation. Lempel and Ziv [Lemp 76] introduced a complexity measure for finite sequences, based on the recursive copying of parts of a sequence. Rueppel [Ruep 84] considered as a measure of randomness the so-called linear complexity profile, denoting the length of the shortest linear feedback shift register which generates that part of the sequence which has already been considered.

The complexity measure proposed in Chapter 3 denotes in a similar fashion the length of the shortest feedback shift register to generate a given (part of a) sequence, where the feedback function may be any function mapping states onto characters. As an example of its power, it declares DeBruijn sequences as non-random, whereas these sequences are considered highly complex according to Lempel and Ziv and some of these sequences are also considered complex according to Rueppel’s linear complexity profile.

One quality aspect of streamciphers, which historically has been neglected, is the question to what extent the mixer frustrates active attacks. Not much literature is available on this topic; a fact supported by the widely used linear mixer. The nonlinear mixers proposed in Chapter 2 can be seen as an improvement to modern streamcipher design.
Chapter 2

On Data Integrity and Active Eavesdropping

2.1 Introduction

Data integrity is concerned with the protection of stored or transmitted data against manipulation, i.e. it deals with the question: has the data been intentionally or unintentionally changed? The emphasis here lies on the word 'changed', viz. not every possible change of the data is considered. In fact one may wish to distinguish between the following types of changes of the transmitted data:

- To disturb the communication and hence reduce the amount of transmitted information. This is usually called jamming.

- To alter the transmitted information arbitrarily, i.e. without being able to know the exact resulting changes.

- To alter the transmitted information, aiming at a specific result.

The latter two operations are known as active eavesdropping, when applied to the encrypted text of a cipher system. Active eavesdropping is depicted in Figure 2.1. It is the objective of an active eavesdropper to manipulate the ciphertext $C$ in such a way that the plaintext $P$ is changed into a different plaintext $P'$ without the knowledge of the actual key that is used. The manipulation operation $S$ consists of e.g. deleting or repeating parts of the cryptogram, or inserting fake cryptograms, but also of performing some arithmetic operation such as addition of data.
As an example of arbitrarily changing the transmitted information, one might consider a communication link which is used to transmit ideally compressed source text that is enciphered. Inserting random ciphertext will cause the receiving system to generate one of all possible source texts, although it is unknown to the active eavesdropper which plaintext will actually result. Especially in electronic payment systems this situation is dangerous. If for example it is known to a fraud that the underlying plaintext of a cryptogram is an amount which starts with the digit 1, he has 80% chance of raising this amount by arbitrarily changing the ciphertext which corresponds to the first digit.

An example of changing the transmitted information, aiming at a specific result, is found in the following situation. Suppose some known message is transmitted, enciphered with a cipher system based on the DES in OFB (Output Feedback Mode, see e.g. [Meye 82]). By simply adding the difference between the plaintext and the desired result to the ciphertext, the manipulator will be successful.

The fact that the plaintext $P$ is (partly) known to an adversary may be regarded as somewhat unrealistic. However, it might contain an authorized or standard message, or it might be a dummy message as is the case with traffic flow security, see [OSI 85]. So the receiver wonders whether the received plaintext is authentic and, moreover, may be confronted with unintentional active eavesdropping in the form of random transmission errors.

It should be clear that in order to avoid the threat of arbitrarily altered plaintext the data has to be redundant in some way, to allow the detection of this alteration. This plaintext redundancy can be the inherent (or natural) redundancy of the source text, or it may be added redundancy. Added redundancy may be non-cryptographic such as channel error detecting codes or cryptographic such as a message authentication code (MAC) or its streamcipher equivalent: pseudo-randomly injected characters. In the
In the case of added redundancy the detection can be accomplished automatically by the system, but natural redundancy usually is suited better for detection by the user at the receiving end. In the sequel methods are described that reduce the threat of specifically altered plaintext to that of arbitrarily altered plaintext.

There are several methods known from the literature to protect against active eavesdropping, see e.g. [Meye 82]. To judge the suitability of these and other methods we have found the following criteria to be useful:

- The amount of error extension.
The number of additional erroneous plaintext bits at the receiver if only one bit-error occurs in the ciphertext, or the number of additional erroneous plaintext blocks (comprising several bits) if only one ciphertext block is in error.

- The amount of text expansion.
The number of blocks added to the plaintext or the ciphertext by a particular method.

- The detection delay.
The number of text blocks that must be received before one can possibly detect active eavesdropping.

- The probability of success for the active eavesdropper.
The probability of producing a desired plaintext at the receiver.

- The implementation complexity.
A subjective meaning about the difficulty of a possible implementation in hardware or software.

In Section 2.2 some methods for bit-stream ciphers are discussed and a new method is introduced, which can be implemented efficiently and gives minimal error extension. Section 2.3 discusses various modes of blockcipher algorithms and their protection against active eavesdropping. Also new modes are introduced which are superior to the ones widely in use, with respect to the above mentioned criteria.
2.2 Active Eavesdropping with Bit-Stream Ciphers

In a classical bit-streamcipher the plaintext consists of a stream of binary digits to which is added modulo 2 a so called running keystream (or just keystream) which is generated by a keystream generator, depending on some key. The plaintext is obtained from the ciphertext by addition of the same keystream modulo 2. The binary adder is often referred to as the 'mixer'.

In order for both the encrypting and the decrypting keystream generators to produce the same keystream, it is necessary that they both use the same key and are synchronised. Synchronization is usually inevitable with streamcipher systems and it often comes down to sending information about the status of the memory cells of the keystream generator from transmitter to receiver. Due to the synchronous operation of the keystream generators at both ends this system is inherently secure against deleting, repeating and inserting parts of a ciphertext message. The major threat against this cipher system, however, is the modification of a message by addition of a bitstream to the ciphertext.

Consider Figure 2.2, where the ciphertext $C$ is modified by addition of a bitstream $S$ (denoted by $\oplus$). The manipulated ciphertext $C'$ therefore is equal to $C + S$ and because of the linearity of the mixer the modified plaintext $P'$ is equal to $P + S$. Clearly, the problem here is that the mixing function is one fixed function which is also linear.

In general the mixing function $f(P, K)$ can be seen as a set of functions $\mathcal{T} = \{f_K\}$, such that

$$C = f_K(P) \quad \text{and} \quad P = f_K^{-1}(C)$$
for all $K$. If $P$ and $C$ take on values from $GF(2)$ there are only two functions in $\mathcal{T}$, viz. the identity and the complement, which are both linear. If, however, $P$ and $C$ take on values from some larger field, there will be more functions in $\mathcal{T}$, some linear, some nonlinear. By grouping together more than one bit one can create larger fields and hence find a solution to counter the threat of active eavesdropping. This will be the subject of Section 2.3. In this section we will consider a solution based on multiple encryption to achieve the same goal.

### 2.2.1 Text Feedback and Double Encryption

If one makes the keystream generator dependent on either the ciphertext or the plaintext, as shown in Figure 2.3, it immediately becomes clear that the active eavesdropper will be unsuccessful if he changes the ciphertext. The false ciphertext will work its way through the memory elements of the receiving keystream generator, thereby causing successive keystream bits to be in error, mutilating the corresponding plaintext. In the case of plaintext feedback the received plaintext may even remain in error for ever.

Although good against active eavesdropping, the disadvantage of text feedback systems is exactly their error extension of many bits, which makes these systems highly susceptible to random channel errors.

A better alternative with respect to error extension can be found in adding a simple cipher feedback system, i.e. applying double encryption as depicted in Figure 2.4. Here one has exact control over the amount of error extension and it can be chosen such that the probability of success for the active eavesdropper is sufficiently low.

Obviously, if the additional cipher feedback system has only a few bits of memory it becomes difficult again to find enough functions $f$ that depend
on Key 2. However, it appears to be possible to use as little as one bit of additional memory if Key 2 is replaced by a keystream which is independent of the one used in the mixer.

**2.2.2 Two Stage Encryption with a JK-flipflop**

A special case of double encryption is shown in Figure 2.5. Here, two independent keystreams $K_1$ and $K_2$ are used for encryption of the plaintext. The first keystream is added to the plaintext as usual; $K_2$, however, is used together with a JK-flipflop to form an additional cipher feedback system comprising one bit of memory.

A JK-flipflop is a common building block for digital electronic circuitry. It has two inputs $j$ and $k$ and one output $q$, which are related as follows:

$$q_n = (j_{n-1} + k_{n-1} + 1)q_{n-1} + j_{n-1},$$  \hspace{1cm} (2.1)

where the subscripts denote the discrete time intervals at which the flipflop is clocked. From equation (2.1) it follows that the ciphertext is related to the plaintext and both keystreams as follows:

$$C_n = (K_{2,n-1} + 1)C_{n-1} + K_{1,n-1} + P_{n-1}.$$  \hspace{1cm} (2.2)

It can be seen from equation (2.2) that an active eavesdropper can successfully change one bit with probability one half.

The described method of two stage encryption can easily be extended to more than two stages, as shown in Figure 2.6. In this way the probability of success for the active eavesdropper can be made arbitrarily small at the cost of an increased error extension and the generation of multiple
keystreams. In this general situation, using $N$ stages, the transmitted and received plaintexts and the ciphertext are related as follows:

$$C_n = P_{n-N} + K_{1,n-N} + \sum_{i=1}^{N} \kappa_{i,n-N} C_{n-i}, \quad (2.3)$$

$$C'_n = C_n + S_n, \quad (2.4)$$

$$P'_n = P_n + S_n + \sum_{i=1}^{N} \kappa_{i,n} S_{n-i}, \quad (2.5)$$

where the variables $\kappa_{i,n}$ depend on the various keystreams $K_{i+1,n}$ in a rather complicated way. By straightforward calculation, however, one can derive the following expressions for the $\kappa_{i,n}$:

$$\kappa_{1,n} = \sum_{i=0}^{N-1} (K_{i+2,n+i} + 1),$$

$$\kappa_{2,n} = \sum_{i=0}^{N-2} (K_{i+2,n+i} + 1) \sum_{j=i+1}^{N-1} (K_{j+2,n+j-1} + 1),$$

$$\kappa_{3,n} = \sum_{i=0}^{N-3} (K_{i+2,n+i} + 1) \sum_{j=i+1}^{N-2} (K_{j+2,n+j-1} + 1) \sum_{k=j+1}^{N-1} (K_{k+2,n+k-2} + 1),$$

$$\vdots$$

$$\kappa_{N,n} = \prod_{i=0}^{N-1} (K_{i+2,n-N} + 1).$$

From the above expressions it can be seen that, if all the keystreams $K_{2,n}, K_{3,n}, \ldots, K_{N+1,n}$ are taken to be equal for all $n$ and this one keystream is really random, many of the resulting variables $\kappa_{i,n}$ assume the values 0 and 1 equally often and are statistically independent.
It follows from equation (2.5) that the probability of success depends only on the number of additional stages $N$ and on the number of 1's in $S_n$. If we define the success rate $\rho$ as the probability of success considered over all possible non-zero bitstreams $S_n$ of length $L$, where the bits are supposed to be independent and identically distributed random variables assuming the values 0 and 1 with equal probability, the following expression for the success rate is obtained:

$$\rho = \frac{(1 + 2^{-N})^L - 1}{2^L - 1}. \quad (2.6)$$

This result has the following explanation:

- Changing one bit has a probability of success of $2^{-N}$ if there are $N$ stages used and the $\kappa_{i,n}$, $i = 1, 2, \ldots, N$ are i.i.d. variables assuming the values 0 and 1 equally often.

- Producing a desired plaintext bitstream of $L$ bits has a probability of success of $2^{-wN}$, where $w$ denotes the number of bits that have to be changed in the bitstream.

- Averaging over all bitstreams of length $L$ gives the desired result.

For some values of $N$ and $L$ the success rate according to equation (2.6) is given in Table 2.1.

The described method with JK-flipflops is new in the context of data integrity. These devices were proposed for use in keystream generators already some time ago by Pless, see [Ples 77]. The system she proposed was shown to be weak, however, first by Rubin, see [Rubi 79], and later by Siegenthaler, [Sieg 85]. Our method can also be seen as an extension of the ideas of Sancho [Sanc 87], who proposed using multiple keystreams in a memoryless mixer.
<table>
<thead>
<tr>
<th>$L$</th>
<th>$N = 1$</th>
<th>$N = 2$</th>
<th>$N = 3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.500</td>
<td>0.250</td>
<td>0.125</td>
</tr>
<tr>
<td>2</td>
<td>0.417</td>
<td>0.188</td>
<td>0.089</td>
</tr>
<tr>
<td>3</td>
<td>0.339</td>
<td>0.136</td>
<td>0.061</td>
</tr>
<tr>
<td>4</td>
<td>0.271</td>
<td>0.096</td>
<td>0.040</td>
</tr>
<tr>
<td>5</td>
<td>0.213</td>
<td>0.066</td>
<td>0.026</td>
</tr>
<tr>
<td>6</td>
<td>0.165</td>
<td>0.045</td>
<td>0.016</td>
</tr>
<tr>
<td>7</td>
<td>0.127</td>
<td>0.030</td>
<td>0.010</td>
</tr>
<tr>
<td>8</td>
<td>0.097</td>
<td>0.019</td>
<td>0.006</td>
</tr>
</tbody>
</table>

Table 2.1: Success rates for some values of $L$ and $N$.

### 2.3 Modes of Blockcipher Algorithms and their Protection Against Active Eavesdropping

Blockcipher algorithms are used in a variety of modes for message encryption and authentication purposes. The different modes not all offer the same protection against active eavesdropping. In this section various known and new modes and their behaviour under addition, deletion, repetition and insertion of ciphertext blocks are discussed. Also some implementation examples of the best mode with respect to its protection against active eavesdropping are presented and their performance discussed.

The methods as discussed in Section 2.2 are also applicable to blockciphers. These methods are not considered here, however, as they may give rise to block-error extension, even if only one bit error occurs.

#### 2.3.1 Modes of Blockcipher Algorithms

Let $E = \{E_k | k \in K\}$ be a blockcipher, where the encryption operators $E_k$, $k \in K$ map $m$-bit blocks onto $m$-bit blocks, i.e. $E_k : GF(2)^m \rightarrow GF(2)^m$. Furthermore, let $D_k$, $k \in K$ denote the decryption operators, i.e. $D_k \equiv E_k^{-1}$, for all $k \in K$. Blockciphers with the above mentioned properties can be used in many different modes. Well-known are the ECB (*Electronic Code Book*), CBC (*Cipher Block Chaining*), CFB (*Cipher FeedBack*) and OFB (*Output FeedBack*) modes, see e.g. [Meye 82]. There are many more possible modes, however, such as PBC and PFB, where the roles of plaintext $P$ and ciphertext $C$ have been interchanged.
Two new modes we have investigated are the CBCPD (*Cipher Block Chaining with Plaintext Difference*) and the OFBNLF (*Output Feedback with Non-Linear Function as mixer*) modes. These modes are depicted in Figures 2.7 and 2.8 respectively. The OFBNLF mode can be regarded as a combination of the OFB and the ECB modes, and can in fact be implemented as such. However, in Section 2.3.2 it will be shown that there exist simpler implementations.

All the modes mentioned so far are described by the following equations:

- **ECB:** \( C_n = E_k(P_n) \) \( P_n = D_k(C_n) \)
- **CBC:** \( C_n = E_k(P_n + C_{n-1}) \) \( P_n = D_k(C_n) + C_{n-1} \)
- **CFB:** \( C_n = P_n + E_k(C_{n-1}) \) \( P_n = C_n + E_k(C_{n-1}) \)
- **OFB:** \( C_n = P_n + R_n \) \( P_n = C_n + R_n \)
- **PBC:** \( C_n = E_k(P_n) + P_{n-1} \) \( P_n = D_k(C_n + P_{n-1}) \)
- **PFB:** \( C_n = P_n + E_k(P_{n-1}) \) \( P_n = C_n + E_k(P_{n-1}) \)
- **CBCPD:** \( C_n = E_k(P_n + P_{n-1} + C_{n-1}) \) \( P_n = D_k(C_n) + C_{n-1} + P_{n-1} \)
- **OFBNLF:** \( C_n = E_{R_n}^*(P_n) \) \( P_n = D_{R_n}^*(C_n) \)

Here, \( C_n \) and \( P_n \) denote the \( n \)th ciphertext and plaintext blocks respectively; \( R_n \) is the \( n \)th block of pseudo-random bits and \( E^* \) and \( D^* \) denote a nonlinear mixing function and its inverse.

From the equations describing the modes, one can easily deduce what happens if the \( n \)th ciphertext block is deleted or repeated, or if some block \( S \) is added to it. As an example the deciphered plaintext blocks in the case...
of deletion and addition are given.

<table>
<thead>
<tr>
<th>Cipher TEXT</th>
<th>Cn-1</th>
<th>Cn+1</th>
<th>Cn+2</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECB</td>
<td>Pn-1</td>
<td>Pn+1</td>
<td>Pn+2</td>
<td>...</td>
</tr>
<tr>
<td>CBC / CFB</td>
<td>Pn-1</td>
<td>?</td>
<td>Pn+2</td>
<td>...</td>
</tr>
<tr>
<td>OFB / PBC /</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PFB / CBCPD</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OFBNLF</td>
<td>Pn-1</td>
<td>?</td>
<td>?</td>
<td>?...</td>
</tr>
</tbody>
</table>

Here, a '?' denotes an a priori unknown outcome of the decryption operation.

It clearly can be seen that active eavesdropping will not be successful with the OFBNLF mode, but that occasional errors in the ciphertext will not give rise to block-error extension in the decrypted message. Also there is no detection delay unlike with the CFB and PFB modes, and no text expansion.
2.3.2 Implementations of the OFBNLF Mode

As was already mentioned in Section 2.3.1, the OFBNLF mode can be regarded as a combination of the OFB and ECB modes. The ECB part, however, can be implemented in a much simpler way. The purpose of this ECB part is to keep the probability of success sufficiently low if an active eavesdropper carries out some fixed transformation on a ciphertext block, such as addition. A solution to this problem is to pseudo-randomly choose mixing functions out of a total set $T$, as was already indicated in Section 2.2. This set of functions should be chosen such that only a small fraction of its functions is linear in some operation, in order to keep the probability of success sufficiently low.

One way to create a set of mixing functions is by using different real additions. An example of this method is depicted in Figure 2.9, where the set of functions consists of four different real additions on 4-bit blocks, i.e. 4 bits modulo 2, $2 \times 2$ bits modulo 4, 1 bit modulo 2 and 3 bits modulo 8, and 4 bits modulo 16. The performance of this method with respect to the success rate seems rather difficult to evaluate, but by computer analysis it appears that the success rate for an active eavesdropper, who uses real additions, tends to $50\%$ for large blocklengths.

Another way of creating a set of mixing functions is to split the mixing function up into an invertible transformation on the plaintext block, followed by an addition. A specific example of this method is depicted in Figure 2.10, where cyclic permutations of the plaintext block are chosen as transformations. Assuming that all the additions are modulo 2, it is seen

![Figure 2.9: Different real additions as mixer in OFBNLF.](image-url)
that in order to be successful an active eavesdropper has to guess the cyclic permutation of the plaintext block difference correctly. In other words, if \( CYC_k(X) \) denotes the cyclic shift of the bits of block \( X \) \( k \) times (\( k \) interpreted as an integer), then \( S = CYC_k(P + P') \). If the difference \( P + P' \) is periodic with period \( p \), then the probability of success is \( 1/p \). By assuming equally likely text blocks the success rate \( p \) is easily determined:

\[
\rho = \frac{\sum_{d|L} \frac{1}{p} \mu(d)2^{p/d} - 1}{2^L - 1},
\]

where \( \mu(\cdot) \) denotes the Möbius function. Equation (2.7) is explained as follows:

- There are \( 2^L - 1 \) difference blocks, where the zero-difference case has obviously been excluded, as it will always be successful.

- The number of blocks, cyclic with period \( p \), is equal to \( \sum_{d|p} \mu(d)2^{p/d} \).

- Averaging over all periods \( p \) which divide the block length \( L \) yields the desired result.

In Table 2.2 the success rates of both cyclic permutations and real additions are given for some block lengths.

Figure 2.10: Cyclic permutations of the plain text in OFBNLF.
Data Integrity

<table>
<thead>
<tr>
<th>$L$</th>
<th>Cyclic Perms.</th>
<th>Real Add.</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0.667</td>
<td>0.833</td>
</tr>
<tr>
<td>3</td>
<td>0.429</td>
<td>0.786</td>
</tr>
<tr>
<td>4</td>
<td>0.333</td>
<td>0.690</td>
</tr>
<tr>
<td>5</td>
<td>0.226</td>
<td>0.645</td>
</tr>
<tr>
<td>6</td>
<td>0.206</td>
<td>0.590</td>
</tr>
<tr>
<td>7</td>
<td>0.150</td>
<td>0.565</td>
</tr>
<tr>
<td>8</td>
<td>0.129</td>
<td>0.510</td>
</tr>
</tbody>
</table>

Table 2.2: Success rates of cyclic perms and real additions.

2.4 Conclusions

In this chapter we have discussed the relation between data integrity and protection against active eavesdropping. In particular suitability criteria were introduced for protective measures.

For bit-streamcipher systems existing methods such as text feedback and double encryption were treated. A new method based on the use of JK-flipflops was introduced and shown to be very effective against active eavesdropping and efficiently implementable.

In the case of blockcipher algorithms an overview of the various modes of operation was presented and their behaviour with respect to active eavesdropping discussed. A new mode, called OFBNLF, was introduced and it was demonstrated that it offers good protection against active eavesdropping, has no text expansion, no block-error extension, no detection delay and can be implemented efficiently. Considering its properties, this OFBNLF mode deserves recommendation as one of the standardized modes of use of blockcipher algorithms.
Chapter 3

The Shortest Feedback Shift Register Equivalent: A Different View at Complexity of Sequences

This entire chapter is devoted to the problem of finding the shortest possible feedback shift register (FSR) that can generate a given sequence of characters from some alphabet. Figure 3.1 shows such a FSR, which consists of a number of memory cells $M_1, \ldots, M_n$ that can contain one character each, and some feedback function $F$, sometimes called a substitution table or truth table, that outputs one character for every combination of input characters. At periodic time intervals, governed by some clocking mechanism, the contents of memory cell $M_i$ are transferred into memory cell $M_{i+1}$ and $M_1$ receives the output value of $F$.

For the purpose of finding the shortest FSR with a given sequence a complexity measure will be defined and its properties examined. It should be understood that we do not restrict ourselves to some particular type of feedback function, such as linear functions. The latter type of feedback function used with FSR's has been studied extensively and many publications exist. In fact, almost any introductory text on cryptology has one or more chapters devoted to this subject. Well-known in this respect are the notions of linear feedback shift register equivalent and linear complexity profile, see for example [Mass 69] and [Ruep 84]. We will use terms as
maximum order feedback shift register equivalent and maximum order complexity profile, to denote the situation where the feedback function may be any memoryless mapping, linear or nonlinear. The name 'maximum order complexity' is chosen, because unlike with linear (or first order-) complexity, quadratic (or second order-) complexity, etc. there is no restriction on the feedback function, except that it must be memoryless. In the sequel the words 'maximum order' will often be omitted when maximum order complexity is meant.

The motivation for this work is based on the following considerations:

- Maximum order complexity may be an additional figure of merit to qualify pseudo-random sequences or even be better than the existing ones, depending on its typical behaviour.

- How difficult is it to determine the shortest FSR equivalent of a given sequence of characters?

- It can be very effective to have an equivalent FSR from an eavesdropper's point of view.

The last consideration is immediately clear. Imagine the use of a pseudo-random sequence generator which produces only one sequence, but the phase of this sequence is unknown as it depends on the actual key being used. In this situation the equivalent shortest FSR needs to be determined only once and it will synchronize automatically if the plain text is known. The problem of finding the FSR equivalent is also addressed in [Cohn 69], where an in-principle but difficult solution for the nonlinear case is briefly described.

In the first section of this chapter maximum order complexity will be defined and its basic properties shown. Some strongly related topics concerning feedback functions and properties of sequences will also be covered. Section 2 discusses a linear time-and-memory algorithm to determine the maximum order complexity profile of finite or periodic sequences over arbitrary alphabets. The third section is concerned with random sequences and establishes the typical behaviour of the complexity profile. In Section 4 a statistical approach towards the complexity of random sequences is used to come to asymptotic results.
3.1 The Maximum Order Complexity of Sequences

Consider the following problem. Given a sequence \( s = (\alpha_0, \alpha_1, \ldots, \alpha_{l-1}) \) of length \( l \), with characters \( \alpha_i \in \mathcal{A} \), where the alphabet \( \mathcal{A} \) is some finite set. How many sections (i.e. memory cells) should a feedback shift register at least have in order to generate the sequence \( s \)? So regardless of what the (memoryless) feedback function would have to be, linear or nonlinear. To this end, the following complexity measure is defined:

**Definition 3.1** The maximum order complexity \( c(s) \) of a sequence \( s = (\alpha_0, \alpha_1, \ldots, \alpha_{l-1}) \) with characters \( \alpha_i \in \mathcal{A} \), where the alphabet \( \mathcal{A} \) is some finite set, is defined to be the length \( L \) of the shortest feedback shift register for which there exists a memoryless feedback mapping, such that the FSR can generate the sequence \( s \).

Maximum order complexity is expressed as being \( L \) characters. By this it is implicitly assumed that the memory cells can only contain characters from the alphabet \( \mathcal{A} \).

Associated with any feedback function \( F \) is a substitution table or truth table, which can be seen as a list of argument values with the corresponding function values. The memory cells of the FSR provide for the argument values and hence the truth table is determined by the sequence \( s \). In order for a feedback function to be memoryless, each argument value must correspond with one function value, i.e. the function \( F \) is single-valued.
Definition 3.2 A feedback function $F$ is said to have a proper truth table if and only if each argument value corresponds to one function value, or equivalently the function is single valued.

In general it is possible that a truth table is not specified completely by the sequence it generates, in which case there are no function values specified for one or more argument values.

Maximum order complexity has a number of basic properties, viz.:

Proposition 3.1

1. $c(s)$ is equal to the shortest length $l$, such that all the subsequences of $s$ of length $l$, have unique successor characters.

2. For a sequence $s$ consisting of two or more possibly repeated different characters, the complexity $c(s)$ is equal to the length-plus-one of the longest subsequence that occurs at least twice with different successor characters.

3. The complexity of a sequence is 0 iff this sequence consists of one possibly repeated character.

4. The maximum value of the complexity of a sequence of length $l$ is $l - 1$. A sequence of length $l$ has a complexity of $l - 1$ iff the sequence consists of $l - 1$ consecutive copies of some character, followed by an unidentical character.

5. A complexity $c(s) = c$ implies that, either all $c$-long subsequences of $s$ are distinct, or only one of all $c$-long subsequences is repeated and the others are distinct.

Proof. These properties can easily be proven:

1. Suppose $c(s)$ would be less than the shortest length for which all subsequences have unique successor characters, then there would be at least two identical entries in the truth table of the feedback function with different function values. As this is not a proper truth table for a memoryless function the assumption must be wrong.

   Also, by the definition of maximum order complexity, $c(s)$ cannot be greater than the aforementioned shortest subsequence length.

2. In order for the truth table of the feedback function to be proper, $c(s)$ must be one greater than the length of the longest subsequence which occurs at least twice with different successor characters.
3. A sequence consisting of only one possibly repeated character can of course be generated without any memory cell. A sequence consisting of only two different characters cannot be generated without memory cells as the successor of a character depends upon the character itself.

4. This follows immediately from 1. and 2.

5. This is implied by 1. Finite sequences always have one subsequence which has no successor character. Hence, this one subsequence can occur more than once in the sequence.

Periodic sequences of period $p$ are denoted by $s = (\alpha_0, \alpha_1, \ldots, \alpha_{p-1})^\infty$. With the period we mean the least integer $p$, such that $\forall_{i \geq 0} [\alpha_{i+p} = \alpha_i]$. For periodic sequences we have the following property:

**Proposition 3.2**

1. The minimum complexity of a periodic sequence of period $p$ is $\lceil \log_a p \rceil$, where $a = |\mathcal{A}|$, the cardinality of the character alphabet.

2. The maximum complexity of a periodic sequence of period $p$ is $p - 1$.

**Proof.**

1. This follows immediately from the fact that a FSR of length $L$ is a finite state device with at most $a^L$ distinct states.

2. The proposition is proven by showing that all the $p - 1$ long subsequences of a periodic sequence are necessarily unique, and by giving an example of a periodic sequence which has a complexity of $p - 1$. Without loss of generality, assume that the two subsequences $(\alpha_0, \alpha_1, \ldots, \alpha_{p-2})$ and $(\alpha_i, \alpha_{i+1}, \ldots, \alpha_{i+p-2})$, with the indices taken modulo $p$, are equal. As a consequence the characters $\alpha_0, \alpha_i, \alpha_{2i}, \ldots$ are all equal and as a result the sequence is periodic with a period less than $p$, contradicting the assumption.

The maximum complexity $p - 1$ is attained by e.g. the sequence $(0,0,\ldots,0,1)^\infty$. 

\[ p \]
The Shortest FSR Equivalent

Complexity is defined for sequences with characters from arbitrary alphabets. From a practical point of view the alphabet cardinality should be finite, so that there exist memory cells to store the characters in. As mentioned before the feedback function consists in its most general form of a substitution- or truth table. For long sequences this table will become very large and therefore infeasible to implement. In the more practical case one does not use a truth table, but rather some logic circuitry which performs arithmetic over $GF(q)$, the finite field with $q$ elements. It is, however, always possible to employ an appropriate finite field in which the finite alphabet $A$ is embedded, e.g. $GF(27)$ or $GF(32)$ for $\{a,b,\ldots,z\}$. It seems natural for a FSR to also consider the complexity or degree of difficulty of the feedback function itself. One could consider for example the number of terms and highest degree in some representation of the function like the Disjunctive Normal Form (DNF, see e.g. [MacW 78, pg. 370]) or the Algebraic Normal Form (ANF, see e.g. [Ruep 84, pg. 54]). This problem will be treated in Chapter 4. As is the case with many complexity measures, the relation between high or low complexity and cryptographically good or bad sequences is not straightforward. Just as with linear complexity high maximum order complexity sequences are not necessarily cryptographically good, as demonstrated by the sequence $(00\ldots01)$ of length $l$ and complexity $l - 1$. Clearly, one has to find out the typical complexity values of good sequences or better even the typical complexity profile as done by Rueppel in [Ruep 84, Ch. 4] for linear complexity.

From our definition of complexity it can be seen that in case the feedback function turns out to be linear, maximum order complexity is equal to linear complexity. This situation occurs with the so-called pseudo-noise or PN-sequences (sometimes called maximum-length or ML-sequences) of period $2^e - 1$, see e.g. [Golo 67].

Example 3.1 Consider the following sequence of length 25, obtained with an unbiased Dutch dice: $s_D = (6544552566433434162531433)$. It has a complexity of 3 characters, as all the subsequences of length 3 are distinct, but subsequence $(43)$ has two different successors.

Example 3.2 The binary sequence $(110011010)$ of length 9 has complexity 4 bits, whereas the linear complexity is 5 bits. The truth table of the equivalent FSR (it will sometimes be called the truth table of the sequence) contains 9-4=5 specified entries out of the maximum of 16. Therefore a feedback function $F(x_0, x_1, x_2, x_3)$ can be determined, e.g.:

$$F = x_0x_1\bar{x}_3 \lor x_0\bar{x}_1x_3 \lor x_1x_2.$$
Here, the DNF representation is used which is translated into usual $GF(2)$ arithmetic as follows:

$$a \lor b = a + b + ab,$$

$$\bar{a} = a + 1.$$

Note that if this sequence is to be continued periodically, denoted by $(110011010)\circ$, the complexity increases to 5, the linear complexity to 7 and the feedback function changes to: $x_0x_1 \lor x_1x_2x_4 \lor x_0x_2x_3 \lor x_0x_2x_4$.

### 3.1.1 The Complexity of Transposed and Reciprocal Sequences

Consider again a sequence $s = (\alpha_0, \alpha_1, \ldots, \alpha_{l-1})$, with $\alpha_i \in A$. The transposed sequence $t = Ts = (\beta_0, \beta_1, \ldots, \beta_{l-1})$, with $\beta_i \in B = T_A$ is defined to be the sequence which is obtained by substituting each character $\alpha_i$ of $s$ by a character $\beta_i$ from the alphabet $B$, where the transposition operator $T$ induces a one-to-one correspondence between the $\alpha_i$ and the $\beta_i$ for all $i$, $0 \leq i \leq l - 1$. So if the original and the transposed alphabets are essentially the same, the substitution is a permutation of the alphabet characters. Note, however, that the cardinalities of the alphabets $A$ and $B$ need not necessarily be the same; it is sufficient that $B$ contains at least as many characters as the number of different characters $\alpha$ present in the sequence $s$. For these transposed sequences we have the following result:

**Proposition 3.3** For all sequences $s$ the complexity of $s$ and the complexity of its transpose $Ts$ have the same value.

**Proof.** One simply substitutes all the specified entries in the truth table as well as their contents. This certainly does not increase the complexity as the substitution is one-to-one and the truth table remains proper. But an improper truth table cannot be made proper by a one-to-one substitution, implying that the complexity cannot be decreased either. □

For binary sequences we have as a special result:

**Corollary 3.4** For all binary sequences $s = (s_0, s_1, \ldots, s_{l-1})$ the complexity of $s$ and its complement $\bar{s} = (s_0 + 1, s_1 + 1, \ldots, s_{l-1} + 1)$ have the same value.

**Proof.** This follows immediately from the previous proposition as every 1 is substituted by a 0 and vice versa. □
As a consequence, in the binary case the complementary sequence is generated by a feedback function which has inclusion and multiplication interchanged in the DNF representation.

Next we restrict ourselves to periodic sequences of period $p$. For this type of sequences we have the following results:

**Lemma 3.5** Let the vector denoting the contents of the memory cells of a FSR be called the state of the shift register. Every FSR which generates a periodic sequence of period $p$ traverses $p$ distinct states which all have unique predecessor states, i.e. there cannot be two states that have one and the same successor state.

**Proof.** The proof is trivial: suppose there exist two states that have the same successor state, then, as any state has only one successor state due to the memoryless feedback function $F$, the shift register will cycle towards only one of the two states, thereby excluding the other state. As this contradicts the assumption made, the result follows. □

**Proposition 3.6** A periodic sequence $s = (a_0, a_1, \ldots, a_{p-1})^\infty$ and its reciprocal $s^* = (a_{p-1}, \ldots, a_1, a_0)^\infty$ have the same complexity.

**Proof.** First suppose that the complexity of the reciprocal sequence is greater. Then there must be at least two identical subsequences of length $c(s)$ in the reciprocal sequence with different successor characters. This, however, implies that the truth table of $s$ must have two entries that differ only in one position and have the same successor characters. As a consequence the FSR which generates $s$ must have two different states with the same successor state. This situation is ruled out by Lemma 3.5, implying that the complexity of the reciprocal cannot be greater. Now suppose that the complexity of the reciprocal is less. By exchanging the roles of $s$ and $s^*$ it follows from the first part of the proof that the complexity of $s$ cannot be greater than the complexity of $s^*$. Therefore a periodic sequence and its reciprocal must have the same complexity. □

It can easily be seen that Proposition 3.6 does not hold for non-periodic sequences in general. For example the sequence $(aa\cdots ab)$ of length $l$ has complexity $l - 1$, its reciprocal $(ba\cdots aa)$ has complexity $1$. 
3.1.2 The Complexity of Periodic Sequences Regarded Over Power Alphabets

Consider a periodic sequence \( s = (\alpha_0, \alpha_1, \ldots, \alpha_{p-1})^\infty \), with \( \alpha_i \in A \). This sequence can be transformed into a sequence of characters from the power alphabet \( A^n \) (where \( A^n \) denotes the \( n \)-fold Cartesian product \( A \times \cdots \times A \)) in a rather trivial way, viz. by grouping together \( n \) consecutive characters and regarding these groups as characters from the power alphabet. The period \( p' \) of this power sequence satisfies the equation:

\[
p' = \frac{p}{\gcd(n, p)}.
\]  

(3.1)

From (3.1) it can be seen that \( p' \leq p \), with equality iff the period \( p \) and the group size \( n \) are relatively prime.

Let \( c_n = c_n(s) \) denote the complexity of the \( n \)-th power sequence of \( s \). The following results hold:

**Lemma 3.7** If \( p > n \geq c_1 \), the power sequence obtained by grouping together \( n \) consecutive characters of a sequence \( s \) with period \( p \) and complexity \( c_1 \), has a complexity \( c_n = 1 \).

**Proof.** For the case \( n < p \) the result follows immediately from the definition of complexity, which implies that every subsequence of \( c_1 \) characters is followed by a unique subsequence of the same length. \( \Box \)

Note that if \( n = p \), we have a power sequence consisting of only one character, so \( c_p = 0 \).

**Proposition 3.8** Let \( \gcd(n, p) = 1 \) for all \( n \) under consideration. For increasing \( n \), the successive complexities \( c_n \) are monotonically non-increasing.

**Proof.** The longest subsequence which occurs at least twice in the original sequence has length \( c_1 - 1 \) by definition. As \( n \) and \( p \) are relatively prime this longest subsequence will also give rise to the longest subsequence in the power sequence. Hence, the longest subsequence which occurs at least twice in the power sequence will have length \( \lfloor (c_1 - 1)/n \rfloor \). Therefore the complexity of the power sequence is given by the equation:

\[
c_n = \left\lfloor \frac{c_1 - 1}{n} \right\rfloor + 1,
\]  

(3.2)

which proves the proposition. \( \Box \)
Let $c_n^w$ denote the weighted complexity, i.e. $c_n^w = n c_n$. So $c_n^w$ merely expresses the number of elementary memory cells necessary to generate the power sequence and, hence, also the sequence itself. From (3.2) one can see that $\forall_{1 \leq n \leq c_1} [c_n^w \geq c_1]$. For large values of $c_1$, however, $c_n^w$ will be equal to $c_1$ for values of $n$ greater than 1, but much less than $c_1$.

**Example 3.3**

\begin{align*}
(010111011000110)^\infty & \quad p = 15 \quad c_1 = 6 \quad c_1^w = 6 \\
(113120302323012)^\infty & \quad p' = 15 \quad c_2 = 3 \quad c_2^w = 6 \\
(27306)^\infty & \quad p' = 5 \quad c_3 = 1 \quad c_3^w = 3
\end{align*}

**Example 3.4**

\begin{align*}
(1100100)^\infty & \quad p = 7 \quad c_1 = 5 \quad c_1^w = 5 \\
(3021210)^\infty & \quad p' = 7 \quad c_2 = 3 \quad c_2^w = 6 \\
(6231144)^\infty & \quad p' = 7 \quad c_3 = 2 \quad c_3^w = 6 \\
(C993264)^\infty & \quad p' = 7 \quad c_4 = 2 \quad c_4^w = 8 \\
(024936C)^\infty & \quad p' = 7 \quad c_5 = 1 \quad c_5^w = 5
\end{align*}

In this example extended hexadecimal notation is used, i.e. $\bar{9} = 11001$ and $C = 01100$.

The motivation for the exercise of this subsection is twofold. At first one is interested in the minimum $c_n^w$ of a sequence from the point of view of circuit complexity, viz. the number of memory cells and the degree of difficulty of the feedback function. Secondly, one may be interested in generating the basic sequence with increased speed ($n$ times as fast) or with a number of characters in parallel.

### 3.1.3 Feedback Functions of the Maximum Order FSR Equivalent

The maximum order feedback shift register equivalent of a sequence $s$ is defined as the FSR of length $c(s)$ and a feedback function such that the
FSR can generate the sequence $s$. In this subsection we restrict ourselves to sequences of characters which are elements from some finite field $GF(q)$. In particular binary sequences are considered, as these sequences are the most widely used in many areas including cryptography. For finite field sequences it is customary to use the truth table to derive an analytical expression for the feedback function. In general the truth table of a sequence will not be specified for all $q^c$ possible entries, if $c$ is the complexity of the sequence. This is due to the fact that not necessarily all $q^c$ possible FSR states occur in a particular sequence. The consequence is that there exists an entire class $\Phi_z$ of feedback functions which all give rise to the same sequence $s$. For this class of feedback functions the following result is obtained:

**Proposition 3.9** Let $\Phi_z$ denote the class of feedback functions of the maximum order feedback shift register equivalent of the periodic sequence $s$ over $GF(q)$, where $s$ has complexity $c$ and period $p$. The number $|\Phi_z|$ of functions in the class $\Phi_z$ satisfies:

$$|\Phi_z| = q^{q^c-p}.$$  

**Proof.** There are $p$ specified entries in the truth table out of a total of $q^c$. The remaining $q^c - p$ entries can each be assigned one out of $q$ different values.

A similar result holds for non-periodic sequences, if only the distinct states are considered of which there may be less than $l - c$.

It is interesting to note that the smallest class, which contains only one feedback function, occurs when $p = q^c$, i.e. for the so-called DeBruijn sequences (see e.g. [DeBr 46]). There are $q$ functions in the class for maximum length sequences, where $p = q^c - 1$; one function maps the all-zero state onto itself and the other ones link the all-zero state to the maximum length sequence through a branchpoint. The largest class occurs for sequences with maximum complexity, $c = p - 1$. In the binary case, $q = 2$, the largest class contains exactly as many feedback functions as there are DeBruijn sequences of order $p$.

So in general $\Phi_z$ contains more than one function and one is able to search for functions exhibiting certain properties such as the least order product function or the function with the least number of terms. This problem is very well-known in switching theory, where truth tables with unspecified entries, so-called don't cares, are considered. One of the methods minimizes the number of terms and their orders in the inclusive-or sum of products of variables or their complements: the DNF representation as
was noted before. See for example [McCl 57] or [Laar 84]. A feedback function with minimized DNF will be called a MDNF function from now on.

As is well-known, the feedback function completely determines the cycle structure of a feedback shift register, i.e. it determines how successor states are assigned to all the states of a FSR. From the way a MDNF function is obtained it can be seen that this function gives rise to only one cycle, viz. the periodic sequence, and the maximum number of branchpoints which connect all the remaining states to that one cycle. The next result shows that there always exists a feedback function in $\Phi_a$ which gives rise to cycles only and consequently no branchpoints.

**Proposition 3.10** For every feedback shift register that generates a periodic binary sequence $s = (s_0, s_1, \ldots, s_{p-1})^\infty$ of period $p$ and complexity $c$, there exists a feedback function of the form:

$$F(x_0, x_1, \ldots, x_{c-1}) \equiv x_0 + F'(x_1, \ldots, x_{c-1}).$$

**Proof.** Proving this proposition is equivalent to proving that there exists a function $F'$, such that:

$$\forall 0 \leq i \leq p-1 [F'(s_{i+1}, s_{i+2}, \ldots, s_{i+c-1}) = s_{i+c} + s_i],$$

where the indices are taken modulo $p$. However, the truth table of $F'$ is proper unless it contains two identical entries with different function values. Assume that there are two such entries, for $i = j$ and $i = k$, then either:

a) $[(s_j, s_{j+1}, \ldots, s_{j+c-1}) = (s_k, s_{k+1}, \ldots, s_{k+c-1})] \land [s_{j+c} \neq s_{k+c}]$,

or:

b) $[(s_j, s_{j+1}, \ldots, s_{j+c-1}) = (s_k, s_{k+1}, \ldots, s_{k+c-1})] \land [s_{j+c} = s_{k+c}].$

But a) cannot occur per definition, as the complexity is $c$ and b) is ruled out by Lemma 3.5. As this contradicts the assumption, the proposition is proven.

Due to Golomb's theorem 1 of the nonlinear theory [Golo 67, pg. 115], Proposition 3.10 implies that there always exists a feedback function in the class $\Phi_a$ for which there are no branchpoints in the cycle structure, or equivalently only closed cycles. For the general case $GF(q)$, $q > 2$, there also exist functions which give rise to cycles only. This is because it is always possible to avoid branchpoints by specifying unspecified states
in such a way that they have different successor states from the already specified states. These functions need not necessarily have a form analogous to that of Proposition 3.10, i.e. \( F(x_0, x_1, \ldots, x_{c-1}) = bx_0 + F'(x_1, \ldots, x_{c-1}) \), with \( b \in GF(q) \setminus \{0\} \). See [Lai 87].

**Example 3.5** The sequence \((2234114)^\infty\) over \(GF(5)\), with period 7 and complexity 2 forms a counter example. From the 4 possible truth tables for \( F' \) it is easily observed that there is no function of the discussed type:

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>1</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>4</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>2</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

If a FSR is used with a feedback function \( F \) of the Proposition 3.10 type to generate a periodic binary sequence \( s = (s_0, s_1, \ldots, s_{p-1})^\infty \) of complexity \( c \), the corresponding feedback function which causes the shift register to generate the reciprocal sequence \( s^* \) is easily determined.

**Lemma 3.11** Let \( F \) denote the feedback function which causes a FSR to generate a periodic binary sequence \( s = (s_0, s_1, \ldots, s_{p-1})^\infty \) of complexity \( c \) and let \( F^* \) denote the feedback function of the FSR generating the reciprocal sequence \( s^* \) of \( s \). Then we have:

\[
F^*(x_0, x_1, \ldots, x_{c-1}) = x_0 + F'(x_{c-1}, \ldots, x_1).
\]

**Proof.** For all \( p \) different FSR states we have by definition:

\[
F(s_i, s_{i+1}, \ldots, s_{i+c-1}) = s_{i+c},
\]

\[
F^*(s_{i+c}, s_{i+c-1}, \ldots, s_{i+1}) = s_i.
\]

But, as a consequence of Proposition 3.10, this is equivalent to:

\[
s_i + F'(s_{i+1}, \ldots, s_{i+c-1}) = s_{i+c},
\]

\[
s_{i+c} + F^*(s_{i+c-1}, \ldots, s_{i+1}) = s_i.
\]
So we have for all \( p \) different FSR states:

\[
F^*(s_{i+c-1}, \ldots, s_{i+1}) = F'(s_{i+1}, \ldots, s_{i+c-1}).
\]

This means that \( F^* \) is essentially the same function as \( F \), but with the arguments \( x_1, \ldots, x_{c-1} \) in reversed order. 

In the next subsection it will be shown that similar properties hold for other types of feedback functions as well.

### 3.1.4 Properties of Feedback Functions in \( \Phi_z \)

In this subsection only periodic binary sequences are considered and a closer look at the function class \( \Phi_z \) will be taken.

Let \( \Sigma \) denote the set of all states which occur in a FSR that generates a periodic binary sequence \( \mathcal{s} = (s_0, s_1, \ldots, s_{p-1})^\infty \) of complexity \( c \), by means of a feedback function \( F \in \Phi_z \).

States are denoted by \( S^i = (s_i, s_{i+1}, \ldots, s_{i+c-1}) \), with \( 0 \leq i \leq p - 1 \) and the indices taken modulo \( p \).

Hence, \( \Sigma = \{ S^i \} \subseteq GF(2)^c \) and \( F : GF(2)^c \mapsto GF(2) \).

Now suppose that \( F_a \in \Phi_z \) and \( F_b \in \Phi_z \), with \( F_a \neq F_b \), i.e. both functions yield different function values for at least one argument value. Or equivalently:

\[
\forall \sigma \in \Sigma \ [F_a(\sigma) = F_b(\sigma)], \quad \text{and} \quad \exists \tau \in \Sigma \ [F_a(\tau) = F_b(\tau) + 1], \quad \text{with } \Sigma := GF(2)^c \setminus \Sigma.
\]

Let \( F_m \in \Phi_z \) be the MDNF function as described in the previous subsection, then, if \( \Sigma \neq \emptyset \), we have:

\[
\exists (\sigma, \tau) \in \Sigma \times \Sigma \ [F_m(\sigma) = F_m(\tau)].
\]

This is the branchpoint situation. Next, let \( F_a \) be of the Proposition 3.10 type, then we have \( F_a(s_i, s_{i+1}, \ldots, s_{i+c-1}) = s_i + F'_a(s_{i+1}, \ldots, s_{i+c-1}) \) for all states. Also, let \( F_b \) be an arbitrary function in \( \Phi_z \), which can in general be written as:

\[
s_i F'_1(s_{i+1}, \ldots, s_{i+c-1}) + s_i F'_2(s_{i+1}, \ldots, s_{i+c-1}).
\]

Then for \( (s_i, s_{i+1}, \ldots, s_{i+c-1}) \in \Sigma \) we have:

\[
|s_i = 0| \quad \Rightarrow \quad [F'_a(s_{i+1}, \ldots, s_{i+c-1}) = F'_2(s_{i+1}, \ldots, s_{i+c-1})],
\]

\[
|s_i = 1| \quad \Rightarrow \quad [F'_a(s_{i+1}, \ldots, s_{i+c-1}) = F'_1(s_{i+1}, \ldots, s_{i+c-1})].
\]
This, however, means that not for all \( \sigma \in \Sigma \) it holds that \( F_1 = F_2 + 1 \), because \( \Sigma \) does not contain all possible vectors of length \( c \), i.e. there may be a pair of states with:

\[
([s_i, s_{i+1}, \ldots, s_{i+c-1}] \in \Sigma] \land ([\overline{s}_i, s_{i+1}, \ldots, s_{i+c-1}] \in \overline{\Sigma}].
\]

Hence, there may be two states \( \sigma_1, \sigma_2 \in \Sigma \) for which \( F_1(\sigma_1) = F_2(\sigma_1) \) and \( F_1(\sigma_2) = F_2(\sigma_2) + 1 \).

These properties are in fact the reason why no general expression can be found, which relates a feedback function of the reciprocal sequence to the feedback function of the sequence itself. There exist conditions, however, which relate the feedback functions \( F \in \Phi_\Sigma \) and \( F^* \in \Phi_{\Sigma^*} \) by means of certain permutations of the arguments of \( F \) and \( F^* \). In order to establish these conditions, we have the following additional definitions:

\( \Sigma^* \) is the set of states of the reciprocal sequence \( s^* \),

\( \Sigma_i = \{ \sigma \in \Sigma | F(\sigma) = i \} \) with \( i = 0, 1 \),

\( \Pi \) is the set of permutations \( P, Q, R, \ldots \) which act on states, i.e. the bitorder of a state on which \( P \) operates is permuted.

The following lemma describes the conditions that must be satisfied in order for a feedback function \( F^* \) to be identical to the corresponding feedback function \( F \) up to a permutation of its arguments.

**Lemma 3.12** Let \( F^* \) denote the feedback function which generates the reciprocal of a sequence \( s \), where the sequence itself is generated by \( F \). Also let \( \forall \sigma \in \Sigma [F^*(\sigma) = F (P \sigma)] \). Then the following conditions hold:

1. \( P^2 = I \), where \( I \) is the identity permutation,
2. \( P \Sigma_0 = \Sigma_0^* \) and \( P \Sigma_1 = \Sigma_1^* \),
3. \( P \Sigma_0^* = \Sigma_0 \) and \( P \Sigma_1^* = \Sigma_1 \).

**Proof.** Condition 1 follows from the fact that the reciprocal of the reciprocal is the identity. Conditions 2 and 3 follow from the definitions of \( \Sigma^* \), \( \Sigma_i \) and condition 1.

The above conditions have a direct influence on the feedback functions, as these are determined by the sets \( \Sigma_0 \) and \( \Sigma_1 \). As a consequence, these sets may have to be extended with some additional states, taken from \( \overline{\Sigma} \), to satisfy all conditions.
Note that Proposition 3.10 and Lemma 3.11 imply that there exists at least one permutation \( P \) which yields a feedback function \( F^* \), i.e.:

\[
P: (s_i, s_{i+1}, \ldots, s_{i+c-1}) \rightarrow (s_i, s_{i+c-1}, \ldots, s_{i+1}).
\]

For sequences that are self-reciprocal, conditions 2 and 3 are exactly the same. Self-reciprocal sequences are defined to be periodic sequences which are identical with their reciprocals up to a certain phase-shift, e.g. \((10100)^\infty\).

### 3.1.5 Dual Sequences

A dual sequence \( s^\perp \) to \( s \) is defined as the sequence which is obtained by chaining together all the states in \( \overline{S} \) of \( s \). This does not mean that there always exists exactly one such sequence. As a matter of fact, there may be more than one dual sequence or there may be no dual sequence at all. Dual sequences exist iff all the states of \( \overline{S} \) can be chained together, which is equivalent to the condition that there must be a closed path between all the states in \( \overline{S} \) in the DeBruijn graph (see e.g. [Golo 67, pg. 129]) of order \( c \), where \( c \) is the complexity of the sequence. If there exist two or more closed paths, equivalently two or more dual sequences exist.

Dual sequences have a number of interesting properties, as given by the following proposition:

**Proposition 3.13** Let \( s \) be a periodic binary sequence with period \( p \). If the dual sequence \( s^\perp \) exists, the following properties hold:

1. **Complexity** \( c(s^\perp) \leq c(s) \).
2. **The period** \( p^\perp \) of \( s^\perp \) satisfies \( p^\perp = 2^c - p \).
3. **The sequences** \( s \) and \( s^\perp \) are generated by the same FSR; its feedback function is necessarily of the Proposition 3.10 type.
4. **Weight** \( w(s^\perp) = 2^{c-1} - w(s) \), where \( w(s) \) denotes the Hamming weight in one period of \( s \), i.e. the number of non-zero elements in the sequence \((s_0, s_1, \ldots, s_{p-1})\).

**Proof.**

1. By definition of duality; the states in \( \overline{S} \) are distinct \( c \)-long vectors, resulting in a proper truth table.
2. The maximum realizable period with a FSR of \( c \) sections is \( 2^c \), hence \( |\overline{S}| = 2^c - p \).
3. From the definition of duality and property 1, it follows that the FSR has the same number of sections and the same feedback function. This feedback function is of the Proposition 3.10 type, because the cycle structure contains exactly two cycles and no branchpoints.

4. As there are only cycles and no branch points, exactly one half of all the states in $\Sigma \cup \overline{\Sigma} = GF(2)^c$ give rise to a feedback function value of 1 and the other half a 0.

□

These properties can be useful in designing feedback functions for nonlinear feedback shift register sequences. For example the sequence $(11010)^\infty$ of period 5, complexity 4 and weight 3, can be generated by 4 different Proposition 3.10 type feedback functions; only one of these feedback functions will also generate the dual sequence $(11110010000)^\infty$ of period 11, complexity 4 and weight 5.

Again it is interesting to see that maximum-length sequences have the all-zero sequence as their dual and that DeBruijn sequences have no duals, as $\overline{\Sigma}$ is empty, and therefore are defined here to be self-dual.

Sequences may exhibit various properties as we have already seen. For instance sequences can be self-reciprocal or self-dual, but also self-complementary. A sequence is called self-complementary, if the complementary sequence is identical to the sequence itself up to a certain phase-shift. One of the interesting questions is whether or not there exist sequences that have two or even all three of the properties. Obviously, self-complementary sequences must be balanced (the weight is equal to the length divided by 2) and have even length. DeBruijn sequences are balanced, even-length sequences and are self-dual by definition. So an interesting question is: do there exist self-complementary or self-reciprocal DeBruijn sequences? It turns out that there are neither self-complementary nor self-reciprocal DeBruijn sequences of order greater than 2.

**Proposition 3.14** There do not exist self-complementary DeBruijn sequences over $GF(2)$ of order greater than 2.

**Proof.** Let $F(x_0, x_1, \ldots, x_{n-1})$ be a feedback function for a FSR which generates a DeBruijn sequence $\xi$ of order $n$, then $F$ can be written as $x_0 + F'(x_1, \ldots, x_{n-1})$ according to Proposition 3.10. The feedback function $F_c$ which generates the complementary sequence $\overline{\xi}$, clearly satisfies:

$$F_c(x_0, x_1, \ldots, x_{n-1}) = F(x_0 + 1, x_1 + 1, \ldots, x_{n-1} + 1) + 1 = x_0 + F'(x_1 + 1, \ldots, x_{n-1} + 1).$$
In order for the sequences $\bar{s}$ and $\bar{g}$ to be equivalent, the corresponding feedback functions must be identical. This in turn implies that:

$$F'(x_1, \ldots, x_{n-1}) \equiv F'(x_1 + 1, \ldots, x_{n-1} + 1).$$  \hfill (3.3)

As a consequence, the coefficients of all terms in the Algebraic Normal Form of $F'$ must be equal. Here, the ANF of $F'$ is written as:

$$a_0 + a_1 x_1 + \cdots + a_{n-1} x_{n-1} + a_{12} x_1 x_2 + \cdots + a_{(n-2)(n-1)} x_{n-2} x_{n-1} + \cdots + a_{12 \cdots (n-1)} x_1 x_2 \cdots x_{n-1}.$$  

It can easily be verified that condition (3.3) is satisfied if $a_{12 \cdots (n-1)} = 0$. Anticipating the results of Chapter 4, we have that the sum of all function values in the truth table of $F'$ is zero:

$$a_{12 \cdots (n-1)} = \sum_{(x_1, \ldots, x_{n-1}) \in GF(2)^{n-1}} F'(x_1, \ldots, x_{n-1}).$$  \hfill (3.4)

However, by a result of Golomb, see [Golo 67, pg. 122], for $n > 2$ the number of cycles in the cycle structure is even or odd according to whether the number of 1's in the truth table of $F'$ is even or odd. Hence, condition (3.4) implies that the cycle structure contains an even number of cycles, which contradicts the fact that $F$ generates a DeBruijn sequence. \hfill \Box

It seems impossible to use properties of feedback functions to show that there do not exist self-reciprocal DeBruijn sequences of order greater than 2. However, their non-existence follows immediately from the symmetry of the DeBruijn graphs.

**Proposition 3.15** There do not exist self-reciprocal DeBruijn sequences over $GF(2)$ of order greater than 2.

**Proof.** We will give the following heuristic proof. Self-reciprocity implies that the same number of states must be traversed when going from the all-zero state to the all-one state, as when going from the all-one state to the all-zero state. This number of states equals $2^{n-1} - 1$, which is always odd, except for $n = 1$. As a consequence, the states at either sides of the all-zero and the all-one states have to be pairwise reciprocal. However, as can be seen from Figure 5.1, there are also self-reciprocal (or symmetric) states
other than the all-zero and the all-one states, if \( n > 2 \). These symmetric states can be used only once, resulting in an asymmetry between the two paths from the all-zero to the all-one states and vice versa. This asymmetry immediately implies the proposed property of the DeBruijn sequences. □

It should also be clear that self-reciprocal sequences of either even or odd period can in principle be constructed by avoiding the appropriate symmetric states.

**Example 3.6 Feedback function and cycle structure.**
Consider the sequence \( s = (1010000)^{\infty} \) of period 7 and complexity 4. So there are 512 different feedback functions for this sequence. The truth table is:

<table>
<thead>
<tr>
<th>( x_0 )</th>
<th>( x_1 )</th>
<th>( x_2 )</th>
<th>( x_3 )</th>
<th>( F )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0 1 0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 1 0 0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 0 0 0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 0 0 0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 0 0 1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 0 1 0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 1 0 1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

From the truth table we find the MDNF function to be \( F(x_0, x_1, x_2, x_3) = \bar{x}_0 \bar{x}_1 \bar{x}_3 \). Moreover, \( \Sigma = \{10, 4, 8, 0, 1, 2, 5\} \), written down in ordered decimal form, i.e. states represented by decimal integers and the order is equal to the way the FSR cycles through its states. The MDNF function links all 9 remaining states in \( \bar{\Sigma} \) to the only cycle through branchpoints as follows:

- \( 9 \rightarrow 2 \)
- \( 13 \rightarrow 10 \)
- \( 3 \rightarrow 6 \rightarrow 12 \rightarrow 8 \)
- \( 11 \rightarrow 6 \)
- \( 7 \rightarrow 14 \rightarrow 12 \)
- \( 15 \rightarrow 14 \).

Reducing the truth table by \( x_0 \) gives the following truth table of \( F' \):
As there are only 5 specified entries in this truth table, there are 8 different $F'$ functions, i.e. functions which result in cycles only. The cycle structure for these 8 functions is as follows:

$F_0': (3, 6, 12, 9) (7, 14, 13, 11) (15)$
$F_1': (3, 7, 14, 13, 11, 6, 12, 9) (15)$
$F_2': (3, 6, 13, 11, 7, 14, 12, 9) (15)$
$F_3': (3, 7, 14, 12, 9) (13, 11, 6) (15)$
$F_4': (3, 6, 12, 9) (7, 15, 14, 13, 11)$
$F_5': (3, 7, 15, 14, 13, 11, 6, 12, 9)$
$F_6': (3, 6, 13, 11, 7, 15, 14, 12, 9)$
$F_7': (3, 7, 15, 14, 12, 9) (6, 13, 11)$

Here, the function index $i$ of $F_i'$ is the decimal representation of the remaining values of the $F'$ table entries 7, 6 and 3.

In this example we have two dual sequences, which are each other's reciprocals, i.e. $(1111011100)_\infty$ and $(1111000110)_\infty$. Observe that they have period 9, complexity 4 and weight 6, in agreement with the theory.

Example 3.7 Reciprocal feedback function
The sequence $s = (111101110000)_\infty$ and its reciprocal $s^* = (11110000110)_\infty$ have period 11 and complexity 4, so there exist 32 different feedback functions for each sequence. These sequences have the following sets of states:

$$\Sigma = \{15, 14, 13, 11, 6, 12, 8, 0, 1, 3, 7\}, \quad \Sigma^* = \{10, 4, 9, 2, 5\},$$
$$\Sigma_0 = \{15, 11, 6, 12, 8\}, \quad \Sigma_1 = \{14, 13, 0, 1, 3, 7\},$$
$$\Sigma_0^* = \{15, 14, 12, 8, 3\}, \quad \Sigma_1^* = \{0, 1, 6, 13, 11, 7\}.$$
Notice that $\Sigma^* = R\Sigma = \Sigma$ and therefore $\Sigma^* = \Sigma$. Now let
\[ \forall \sigma \in \Sigma [F^*(\sigma) = F(P\sigma)]. \]

We then have:
\[ P\Sigma_0 = \{15, 14, 3, 9, 8\}, \quad P\Sigma_1 = \{11, 13, 0, 4, 6, 7\}, \]
\[ P\Sigma^*_0 = \{15, 11, 9, 8, 6\}, \quad P\Sigma^*_1 = \{0, 4, 3, 13, 14, 7\}. \]

So it can clearly be seen that both $\Sigma_0$ and $\Sigma^*_0$ have to be extended with state 9 and both $\Sigma_1$ and $\Sigma^*_1$ with state 4. Note that these additional states do not belong to $\Sigma$, but rather to $P$. The remaining 'undefined' states are 10, 2 and 5 and are all invariant under $P$, which implies that there are 8 different feedback functions among the total of 32, which have the permutation property $P$. As a check, it is easily verified that the two functions of the form $F = x_0 + F'$ are among these 8.

By chaining together all states of $\Sigma$ the dual sequence $s^\perp = (10100)^\infty$ is obtained. This dual sequence is self-reciprocal and is dual to both $s$ and $s^*$. If we had started with this dual sequence itself, we would have found a total number of $|\Phi_s| = 2048$ different feedback functions, with:
\[ \Sigma^+_0 = \{10, 9, 5\}, \quad \Sigma^+_1 = \{4, 2\}, \]
\[ P\Sigma^+_0 = \{10, 12, 5\}, \quad P\Sigma^+_1 = \{1, 2\}. \]

So that the states 1 and 12 are additionally determined and there are 7 independent choices for the remaining states: 15, 14 and 11, 13, 6 and 3, 8, 0, 7. Therefore there are 128 $P$-property feedback functions that will generate the reciprocal sequence. However,
\[ R\Sigma^+_0 = \{5, 9, 10\} \quad \text{and} \quad R\Sigma^+_1 = \{2, 4\}, \]
so that no other states are additionally determined and there are again 7 independent choices for the remaining states: 15, 14 and 7, 13 and 11, 6, 12 and 3, 8 and 1, 0. Therefore there are 128 $R$-property feedback functions that will generate the reciprocal sequence in this case. The foregoing clearly demonstrates that there are more than one expressions for the reciprocal feedback function and in fact no general expression can be given.

There exist, however, sequences for which every feedback function can be permuted according to $P$, to obtain its reciprocal. One such sequence is $(1111011001000)^\infty$ of period 13 and complexity 4. This particular sequence also has the property that it lacks a dual sequence, as can easily be verified with $\Sigma = \{0, 5, 10\}$. There simply is no way of chaining these three states together.
3.2 The Maximum Order Complexity Profile

The complexity measure defined in the previous section would be of limited practical use if it were hard to determine the complexity in real situations where usually sequences of considerable length occur. Moreover, one is interested in the local (or dynamical) behaviour of the complexity after processing each character in a sequence successively. This interest is easily explained if one compares two sequences which have the same complexity: their complexity profiles may differ entirely, although the end values of their complexities are the same. Therefore, the maximum order complexity profile is proposed as a measure of “goodness” for sequences, i.e. a measure which shows how well a given sequence resembles a real random sequence. For this purpose an algorithm is proposed which can be used to determine the maximum order complexity profile of arbitrary sequences.

Considering random sequences as the best there are, it makes sense to investigate the expected maximum order complexity profile of these sequences. As a first attempt a computer search was carried out, the results of which are tables that list the number of binary sequences of given length and corresponding complexity.

3.2.1 The Directed Acyclic Word Graph

In [Blum 83] Blumer et al. describe a linear-time and -memory algorithm to build a Directed Acyclic Word Graph (DAWG) from a given string of letters, using a mechanism of suffix pointers as described in [McCr 76]. This DAWG is then used to recognize all substrings (or words) in the string, or for source coding purposes as in [Blum 85].

The DAWG consists of at most 2l nodes connected by at most 3l edges, where l is the length of the string. The nodes represent equivalence classes of substrings and the edges are labeled with string letters. An edge points from one node to another if and only if the first equivalence class contains a substring, which extended with the edge’s letter belongs to the other equivalence class. The suffix pointer is an edge which points from a node to the node representing the equivalence class with the longest common suffix of all strings of the first node’s equivalence class. Two substrings are defined to be equivalent if and only if their endpoint sets are equal. An endpoint set of a given substring is defined as the set containing all positions within a string where the given substring ends.
Example 3.8 The string $w = 110100$ gives rise to the following set of all possible substrings, denoted $\text{SUB}(w)$:

$$\text{SUB}(w) = \{\lambda, 1, 0, 11, 10, 01, 00, 110, 101, 010, 100, 1101, 1010, 0100, 11010, 10100, 110100\}.$$

Here $\lambda$ denotes the empty string. The endpoint positions within a string are denoted as follows:

$$\begin{array}{ccccccc}
1 & 1 & 0 & 1 & 0 & 0 & : w, \\
0 & 1 & 2 & 3 & 4 & 5 & 6 : \text{endpoints}.
\end{array}$$

For the given string $w$ the endpoint sets for all substrings are:

$$
\begin{align*}
E_w(\lambda) &= \{0, 1, 2, 3, 4, 5, 6\}, \\
E_w(1) &= \{1, 2, 4\}, \\
E_w(10) &= \{3, 5, 6\}, \\
E_w(01) &= \{4\}, \\
E_w(00) &= \{6\}, \\
E_w(11) &= \{2\}, \\
E_w(101) &= \{4\}, \\
E_w(010) &= \{5\}, \\
E_w(100) &= \{6\}, \\
E_w(110) &= \{3\}, \\
E_w(1010) &= \{5\}, \\
E_w(0100) &= \{6\}, \\
E_w(1101) &= \{4\}, \\
E_w(10100) &= \{6\}, \\
E_w(11010) &= \{5\}, \\
E_w(110100) &= \{6\}.
\end{align*}
$$

From these endpoint sets the following substrings are seen to be equivalent:

$$
\begin{align*}
01 & \equiv_w 101 & \equiv_w 1101, \\
00 & \equiv_w 100 & \equiv_w 0100 & \equiv_w 10100 & \equiv_w 110100, \\
010 & \equiv_w 1010 & \equiv_w 11010.
\end{align*}
$$

As can be seen the 17 substrings form 9 equivalence classes. It is customary to represent each equivalence class by its shortest substring. Doing so we have the following set of equivalence classes of $w$, denoted by $\text{EQ}(w)$:

$$\text{EQ}(w) = \{\lambda, 1, 0, 11, 10, 01, 00, 110, 010\}.$$

Figure 3.2 shows the corresponding DAWG of $w$, where the dotted lines with arrows are the suffix-pointers.

The edges of a DAWG are divided into primary and secondary edges. An edge is called primary if and only if it belongs to the primary path, which is the longest path from the source to a node. With the length of a path the number of edges in that path is meant. The depth of a node is the length of the primary path from the source to that node. In the foregoing example the depths of the nodes, denoted by $d(\cdot)$ are as follows:
The Shortest FSR Equivalent

Figure 3.2: DAWG of 110100.

\[ d(\lambda) = 0, \quad d(1) = 1, \quad d(0) = 1, \]
\[ d(11) = 2, \quad d(10) = 2, \quad d(01) = 4, \quad d(00) = 6, \]
\[ d(110) = 3, \quad d(010) = 5. \]

Looking at the nodes with more than one outgoing edge, i.e. \( \lambda, 1, 0 \) and 10, it can be seen that 10 is the deepest node with this property.

Let \( BN(w) \) be defined as the set of equivalence classes that have more than one outgoing edge. We call this set the set of branchnodes of \( w \). Clearly, \( BN(w) \subseteq EQ(w) \). The maximum depth \( \hat{d}(w) \) of a string \( w \) is defined as:

\[ \hat{d}(w) := \max_{a \in BN(w)} d(a). \]

In Example 3.8 we have \( \hat{d}(110100) = 2 \). The next example illustrates the building process of the DAWG.

Example 3.9 The string \( w' = 1101001 \) is equal to \( w \) concatenated with
'1'. It gives rise to the following set of all possible substrings:

\[ SUB(w') = SUB(w) \cup \{001, 1001, 01001, 101001, 1101001\} \]

where \( SUB(w) \) is calculated in Example 3.8.

From \( SUB(w') \) it follows that there are 5 additional endpoint sets, which are all the same, viz. \{7\}. However, as \( E_w(01) = \{4, 7\} \), substring 01 is no longer equivalent with substrings 101 and 1101. Therefore equivalence class 01 is now split up into two classes 01 and 101. This interesting phenomenon of Blumer's algorithm is depicted in Figure 3.3.

![Figure 3.3: Splitting a DAWG's node.](image)

The algorithm is presented here as in [Blum 83]. It consists of three procedures that can readily be programmed in a computer programming language like PASCAL or C.
Blumer's Algorithm:

\textit{builddawg}(w)

1. Create a node named \textit{source}.
2. Let \textit{currentsink} be \textit{source}.
3. For each letter \(a\) of \(w\) do: Let \textit{currentsink} be \textit{update}(\textit{currentsink},a).
4. Return \textit{source}.

\textit{update}(\textit{currentsink},a)

1. Create a node named \textit{newsink}.
2. Create a primary edge labelled \(a\) from \textit{currentsink} to \textit{newsink}.
3. Let \textit{currentnode} be \textit{currentsink}.
4. Let \textit{suffixnode} be undefined.
5. While \textit{currentnode} isn't \textit{source} and \textit{suffixnode} is undefined do:
   a. Let \textit{currentnode} be the node pointed to by the suffix pointer of \textit{currentnode}.
   b. If \textit{currentnode} has a primary outgoing edge labelled \(a\) then let \textit{suffixnode} be the node that this edge leads to.
   c. Else, if \textit{currentnode} has a secondary outgoing edge labelled \(a\) then:
      1. Let \textit{childnode} be the node that this edge leads to.
      2. Let \textit{suffixnode} be \textit{split}((\textit{currentnode}, \textit{childnode})).
   d. Else, create a secondary edge from \textit{currentnode} to \textit{newsink} labelled \(a\).
6. If \textit{suffixnode} is still undefined, let \textit{suffixnode} be \textit{source}.
7. Set the suffix pointer of \textit{newsink} to point to \textit{suffixnode}.
8. Return \textit{newsink}.

\textit{split}((\textit{parentnode}, \textit{childnode}))

1. Create a node called \textit{newchildnode}.
2. Make the secondary edge from parentnode to childnode into a primary edge from parentnode to newchildnode (with the same label).

3. For every primary and secondary outgoing edge of childnode, create a secondary outgoing edge of newchildnode with the same label and leading to the same node.

4. Set the suffix pointer of newchildnode equal to that of childnode.

5. Reset the suffix pointer of childnode to point to newchildnode.

6. Let currentnode be parentnode.

7. While currentnode isn't source do:
   a. Let currentnode be the node pointed to by the suffix pointer of currentnode.
   b. If currentnode has a secondary edge to childnode, make it a secondary edge to newchildnode (with the same label).
   c. Else, break out of the while loop.

8. Return newchildnode.

3.2.2 The DAWG and the Complexity Profile

From the foregoing examples it should already be clear that the DAWG is a useful tool to determine the maximum order complexity of a given sequence. In this subsection it is shown that Blumer’s algorithm can indeed be exploited to determine the maximum order complexity profile of any sequence in linear time and memory.

For the sake of uniformity in notation we will speak of a sequence $s$ instead of a string $w$ and of a subsequence instead of a word or substring. All set definitions of the previous subsection are translated in this way. The following proposition relates the complexity of a sequence to its maximum depth in the DAWG.

**Proposition 3.16** The complexity $c(s)$ of a sequence $s$ with characters from some finite alphabet $A$ satisfies:

$$
c(s) = \begin{cases} 
0; & BN(s) = \emptyset, \\
\hat{d}(s) + 1; & \text{else}.
\end{cases}
$$
Proof. If there are no branchnodes, the sequence \( s \) consists of only one, possibly repeated character and therefore has zero complexity. If there are branchnodes, the deepest node in fact labels the longest subsequence which occurs at least twice with a different successor character. From the definition of complexity and its properties as given on page 32 it follows immediately that in this case \( c(s) = d(s) + 1 \).

Proposition 3.17 Blumer's algorithm can be used to determine the complexity profile of a sequence \( s \) with characters from some finite alphabet \( A \) in linear time and memory.

Proof. Blumer's algorithm builds a DAWG in linear time and memory. It therefore suffices to show how the maximum depth can be determined from the DAWG in linear time and memory. By looking at the algorithm in detail, it can be seen that only step 5d of \textit{update}, which creates a secondary outgoing edge from a node, causes a possible change in the maximum depth. Hence, it is only necessary to keep track of the maximum depth by testing the depth of the node being equipped with an additional edge against the existing maximum depth. This testing operation clearly does not change the order of the algorithm.

As Blumer et al. did in their paper, it should be noted that the linearity of their algorithm is with regard to the total processing time related to the length of the sequence.

Blumer's algorithm can also be used very well to determine the period of a periodic sequence, as can be seen from the DAWG in the next example.

Example 3.10 The string \( w = 100100100 \) is periodic with period 3. Carrying out the partitioning into equivalence classes as demonstrated in Example 3.8, the result becomes:

\[
\begin{align*}
EQ(w) &= \{\lambda, 1, 0, 10, 00, 01, 010, 001, 0100, 0010, 00100\}, \\
BN(w) &= \{\lambda, 0\}, \\
\hat{d}(w) &= 1.
\end{align*}
\]

The import of this example lies in the behaviour of the suffix pointers, as can be seen from Figure 3.4. The number of edges between e.g. the last node and the node pointed to by the last node's suffix pointer is equal to 3, which is the period of the string.
Figure 3.4: The DAWG of a periodic string.

**Proposition 3.18** Blumer's algorithm can be used to determine the period of a periodic sequence in linear time and memory. To this end, it is necessary and sufficient to examine $p + d + 1$ characters, so at most $2p - 1$ characters, where $p$ denotes the period of the sequence.

**Proof.** From Example 3.10 it can be seen that the number of edges between a node and the node pointed to by the first node's suffix pointer (we will call this the suffix length of a node) can be determined in linear time and memory. Hence, it suffices to show that the suffix length of the last node after processing $p + d + 1$ characters is equal to the period.

1. Consider the following sequence:

$$\mathcal{S} = (\alpha_0, \ldots, \alpha_{p-1}, \underbrace{\alpha_0, \ldots, \alpha_{j-1}}_{d+1}).$$
As \( d + 1 = c \), the complexity of \( s \), it follows that the second subsequence \( \alpha_0, \ldots, \alpha_{j-1} \) is unique up to an exact copy at the beginning of \( s \). This implies that the DAWG’s last equivalence class contains \( \alpha_{p-1}, \alpha_0, \ldots, \alpha_{j-1} \) as the shortest subsequence, since this subsequence has \( p + j \) as its endpoint within \( s \). This shortest subsequence has \( \alpha_0, \ldots, \alpha_{j-1} \) as its longest suffix, which occurs at first at the beginning of \( s \), exactly \( p \) places back in the equivalence class with endpoint \( j \). Therefore the difference in depth between the endnode and the node pointed to by the endnode’s suffix pointer is equal to the period \( p \).

2. Next let \( c < j < p \). Because of the uniqueness of subsequences of length \( c \) the longest suffix of the shortest subsequence in the endnode’s equivalence class occurs only \( p \) places back in the sequence, hence the suffix length being again equal to \( p \).

3. Now let \( j < c \). In this case the uniqueness of any subsequence of length \( j \) is not guaranteed, implying that the suffix length may be shorter than \( p \).

From the foregoing it is clear that Blumer’s algorithm is indeed a powerful tool to obtain both the complexity profile and the period of an arbitrary sequence. To achieve this the original algorithm needs only a few additional rules.

### 3.2.3 The Typical Complexity Profile

In order to get an idea of the expected complexity profile of random sequences a computer program has been used to exhaust all \( 2^l \) binary sequences of length \( l \) and to list the number of sequences with certain complexity. Explicit use has been made of the fact that a sequence and its complement have the same complexity. The results are shown in Tables 3.1 and 3.3. Table 3.1 shows the exact distribution of the complexity for sequence lengths from 1 until 24 and Table 3.3 lists the average and variance of the complexity for these lengths.

The same has also been done for periodic sequences with periods from 1 until 24, where only essentially different sequences are considered, i.e. without the phase-shifted versions. The corresponding results are shown in Tables 3.2 and 3.4 respectively. In Tables 3.3 and 3.4 \( \varepsilon(c) \) and \( \gamma(c) \)
denote the square root of the second moment and the cubic root of the third moment respectively.

From the tables a number of interesting observations can be made. Let \( N_c^l \) denote the number of binary sequences of length \( l \) and complexity \( c \). We see:

1. \( N_c^l = 0 \), if \( l - c \leq 0 \).

2. Let \( L_c^+ \) be defined as the maximum number of sequences of maximum order complexity \( c \), then \( N_c^l \leq L_c^+ \), with equality if \( l \geq 2^c + c \).

3. Let \( L_k^- \) be defined as the maximum number of sequences of maximum order complexity \( l - k \) and length \( l \geq 2k \), then \( N_{l-k}^l \leq L_k^- \), with equality if \( l = 2k \).

4. \( N_c^l = N_c^{l+k} \), for \( l \leq 2c \) and \( k \geq 0 \).

5. The average complexity seems to approach \( 2 \log_2 l \).

6. The difference between the random and the periodic-random cases decreases with increasing value of \( l \).

These observations have the following explanations:

1. This observation is obvious, because the maximum order complexity of any sequence is less than or equal to its length minus one.

2. It can be seen that there is indeed a maximum number of sequences with a given complexity. This maximum number is clearly less than the total number of sequences that can be obtained with a \( c \)-long feedback shift register, as this can also generate sequences of lower complexity. The latter number of sequences can be upper- and lower-bounded as will be shown in Section 3.3.

The value of \( l \) for which \( N_c^l \) attains its maximum \( L_c^+ \) is indeed \( 2^c + c \); as all \( 2^c \) possible shift register states then have a unique successor state, no further increase in \( N_c^l \) is possible. For \( l = 2^c + c - 1 \), \( N_c^l \) includes all cyclic shifts of all DeBruijn sequences of order \( c \). These DeBruijn sequences may be extended periodically, thereby not increasing \( N_c^l \), or extended by a 'wrong' bit and thereby increasing \( N_c^l \) with those sequences which are ultimately periodic with period less than \( 2^c \). Examples are \((11101000111)\) and \((11101000110)\) of periods 8 and 7 and \((10001110100)\) and \((10001110101)\) of periods 8 and 2.
Table 3.1: The number of binary sequences with complexities from 0 until \( l - 1 \), for lengths \( l \leq 24 \).
The Maximum Order Complexity Profile

<table>
<thead>
<tr>
<th></th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
<th>21</th>
<th>22</th>
<th>23</th>
<th>( l )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>9</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>10</td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>11</td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>12</td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>13</td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>14</td>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>15</td>
</tr>
<tr>
<td>16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>16</td>
</tr>
<tr>
<td>17</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>17</td>
</tr>
<tr>
<td>18</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>18</td>
</tr>
<tr>
<td>19</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>19</td>
</tr>
<tr>
<td>20</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>20</td>
</tr>
<tr>
<td>21</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>21</td>
</tr>
<tr>
<td>22</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>22</td>
</tr>
<tr>
<td>23</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>23</td>
</tr>
<tr>
<td>24</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>24</td>
</tr>
</tbody>
</table>

Table 3.1 (cont’d): The number of binary sequences with complexities from 0 until \( l - 1 \), for lengths \( l \leq 24 \).
The number of periodic binary sequences with complexities from 0 until \( p - 1 \), for periods \( p \leq 24 \).

<table>
<thead>
<tr>
<th>( p )</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td></td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>2</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>3</td>
<td>4</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>2</td>
<td>10</td>
<td>8</td>
<td>6</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>14</td>
<td>18</td>
<td>14</td>
<td>4</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>17</td>
<td>40</td>
<td>16</td>
<td>12</td>
<td>10</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>14</td>
<td>64</td>
<td>46</td>
<td>30</td>
<td>14</td>
<td>8</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>13</td>
<td>100</td>
<td>104</td>
<td>54</td>
<td>38</td>
<td>16</td>
<td>6</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>12</td>
<td>142</td>
<td>194</td>
<td>134</td>
<td>70</td>
<td>38</td>
<td>18</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>20</td>
<td>188</td>
<td>366</td>
<td>303</td>
<td>132</td>
<td>74</td>
<td>36</td>
<td>20</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>32</td>
<td>268</td>
<td>644</td>
<td>558</td>
<td>324</td>
<td>170</td>
<td>106</td>
<td>40</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>16</td>
<td>390</td>
<td>1122</td>
<td>1110</td>
<td>732</td>
<td>348</td>
<td>176</td>
<td>92</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
<td>538</td>
<td>1918</td>
<td>2162</td>
<td>1448</td>
<td>806</td>
<td>410</td>
<td>214</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
<td>703</td>
<td>3297</td>
<td>4105</td>
<td>2966</td>
<td>1803</td>
<td>816</td>
<td>416</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
<td>842</td>
<td>5506</td>
<td>7914</td>
<td>5960</td>
<td>3560</td>
<td>1880</td>
<td>958</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
<td>1085</td>
<td>9046</td>
<td>14800</td>
<td>12070</td>
<td>7340</td>
<td>4176</td>
<td>1916</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
<td>1310</td>
<td>14680</td>
<td>27942</td>
<td>23836</td>
<td>15132</td>
<td>8302</td>
<td>4312</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td></td>
<td>1465</td>
<td>23160</td>
<td>51611</td>
<td>47571</td>
<td>31069</td>
<td>17380</td>
<td>9583</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td></td>
<td>1544</td>
<td>36428</td>
<td>94660</td>
<td>94132</td>
<td>63504</td>
<td>35894</td>
<td>19056</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td></td>
<td>1570</td>
<td>56232</td>
<td>172188</td>
<td>185014</td>
<td>128876</td>
<td>74422</td>
<td>39740</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 3.2 (cont'd): The number of periodic binary sequences with complexities from 0 until \( p - 1 \), for periods \( p \leq 24 \).
<table>
<thead>
<tr>
<th>$l$</th>
<th>$\delta(c)$</th>
<th>$VAR(c)$</th>
<th>$\epsilon(c)$</th>
<th>$\gamma(c)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
</tr>
<tr>
<td>2</td>
<td>0.500000</td>
<td>0.250000</td>
<td>0.707106</td>
<td>0.793700</td>
</tr>
<tr>
<td>3</td>
<td>1.000000</td>
<td>0.500000</td>
<td>1.224745</td>
<td>1.401020</td>
</tr>
<tr>
<td>4</td>
<td>1.625000</td>
<td>0.734375</td>
<td>1.837117</td>
<td>2.080084</td>
</tr>
<tr>
<td>5</td>
<td>2.125000</td>
<td>0.859375</td>
<td>2.318405</td>
<td>2.685843</td>
</tr>
<tr>
<td>6</td>
<td>2.656250</td>
<td>0.975585</td>
<td>2.833946</td>
<td>3.300972</td>
</tr>
<tr>
<td>7</td>
<td>3.125000</td>
<td>1.015625</td>
<td>3.283481</td>
<td>3.869834</td>
</tr>
<tr>
<td>8</td>
<td>3.546875</td>
<td>1.091553</td>
<td>3.697550</td>
<td>4.403689</td>
</tr>
<tr>
<td>9</td>
<td>3.921875</td>
<td>1.173584</td>
<td>4.068745</td>
<td>4.895270</td>
</tr>
<tr>
<td>10</td>
<td>4.287109</td>
<td>1.251553</td>
<td>4.430673</td>
<td>5.363188</td>
</tr>
<tr>
<td>11</td>
<td>4.625977</td>
<td>1.308349</td>
<td>4.765292</td>
<td>5.801648</td>
</tr>
<tr>
<td>12</td>
<td>4.934082</td>
<td>1.380909</td>
<td>5.072088</td>
<td>6.210003</td>
</tr>
<tr>
<td>13</td>
<td>5.216064</td>
<td>1.461373</td>
<td>5.354316</td>
<td>6.589070</td>
</tr>
<tr>
<td>14</td>
<td>5.484741</td>
<td>1.521007</td>
<td>5.621690</td>
<td>6.944802</td>
</tr>
<tr>
<td>15</td>
<td>5.728088</td>
<td>1.618142</td>
<td>5.867635</td>
<td>7.276428</td>
</tr>
<tr>
<td>16</td>
<td>5.960724</td>
<td>1.694045</td>
<td>6.101170</td>
<td>7.588446</td>
</tr>
<tr>
<td>17</td>
<td>6.178391</td>
<td>1.769217</td>
<td>6.319947</td>
<td>7.881653</td>
</tr>
<tr>
<td>18</td>
<td>6.383629</td>
<td>1.842598</td>
<td>6.526355</td>
<td>8.157901</td>
</tr>
<tr>
<td>19</td>
<td>6.580772</td>
<td>1.901183</td>
<td>6.723671</td>
<td>8.419902</td>
</tr>
<tr>
<td>20</td>
<td>6.763187</td>
<td>1.974201</td>
<td>6.907598</td>
<td>8.666514</td>
</tr>
<tr>
<td>21</td>
<td>6.938872</td>
<td>2.033202</td>
<td>7.083866</td>
<td>8.900583</td>
</tr>
<tr>
<td>22</td>
<td>7.106244</td>
<td>2.088495</td>
<td>7.251703</td>
<td>9.123092</td>
</tr>
<tr>
<td>23</td>
<td>7.265850</td>
<td>2.140045</td>
<td>7.411654</td>
<td>9.334858</td>
</tr>
<tr>
<td>24</td>
<td>7.418194</td>
<td>2.188617</td>
<td>7.564272</td>
<td>9.536652</td>
</tr>
</tbody>
</table>

Table 3.3: Statistical moments of complexity distribution.
<table>
<thead>
<tr>
<th></th>
<th>$\xi(c)$</th>
<th>$VAR(c)$</th>
<th>$\varepsilon(c)$</th>
<th>$\gamma(c)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
</tr>
<tr>
<td>2</td>
<td>1.000000</td>
<td>0.000000</td>
<td>1.000000</td>
<td>1.000000</td>
</tr>
<tr>
<td>3</td>
<td>2.000000</td>
<td>0.000000</td>
<td>2.000000</td>
<td>2.040828</td>
</tr>
<tr>
<td>4</td>
<td>2.666667</td>
<td>0.222222</td>
<td>2.708013</td>
<td>2.975103</td>
</tr>
<tr>
<td>5</td>
<td>3.666667</td>
<td>0.222222</td>
<td>3.696846</td>
<td>4.017286</td>
</tr>
<tr>
<td>6</td>
<td>3.888889</td>
<td>0.543209</td>
<td>3.958114</td>
<td>4.768727</td>
</tr>
<tr>
<td>7</td>
<td>4.666667</td>
<td>1.333333</td>
<td>4.807402</td>
<td>5.585146</td>
</tr>
<tr>
<td>8</td>
<td>5.000000</td>
<td>1.333333</td>
<td>5.131601</td>
<td>6.299045</td>
</tr>
<tr>
<td>9</td>
<td>5.464286</td>
<td>1.534439</td>
<td>5.602933</td>
<td>6.864245</td>
</tr>
<tr>
<td>10</td>
<td>5.696970</td>
<td>1.887971</td>
<td>5.860327</td>
<td>7.380941</td>
</tr>
<tr>
<td>11</td>
<td>6.161290</td>
<td>2.328824</td>
<td>6.347466</td>
<td>7.906448</td>
</tr>
<tr>
<td>12</td>
<td>6.286567</td>
<td>2.001461</td>
<td>6.443787</td>
<td>8.258533</td>
</tr>
<tr>
<td>13</td>
<td>6.660317</td>
<td>2.605251</td>
<td>6.853107</td>
<td>8.670591</td>
</tr>
<tr>
<td>14</td>
<td>6.812231</td>
<td>2.536663</td>
<td>6.995938</td>
<td>8.989496</td>
</tr>
<tr>
<td>15</td>
<td>7.040330</td>
<td>2.683983</td>
<td>7.228432</td>
<td>9.270708</td>
</tr>
<tr>
<td>16</td>
<td>7.224510</td>
<td>2.700576</td>
<td>7.409056</td>
<td>9.535607</td>
</tr>
<tr>
<td>17</td>
<td>7.445396</td>
<td>2.897083</td>
<td>7.637473</td>
<td>9.804656</td>
</tr>
<tr>
<td>18</td>
<td>7.583196</td>
<td>2.791799</td>
<td>7.765092</td>
<td>10.01734</td>
</tr>
<tr>
<td>19</td>
<td>7.768718</td>
<td>2.970862</td>
<td>7.957628</td>
<td>10.24094</td>
</tr>
<tr>
<td>20</td>
<td>7.904366</td>
<td>2.926246</td>
<td>8.087351</td>
<td>10.43165</td>
</tr>
<tr>
<td>21</td>
<td>8.052274</td>
<td>3.018397</td>
<td>8.237567</td>
<td>10.61744</td>
</tr>
<tr>
<td>22</td>
<td>8.188500</td>
<td>3.009961</td>
<td>8.370274</td>
<td>10.79130</td>
</tr>
<tr>
<td>23</td>
<td>8.323468</td>
<td>3.078560</td>
<td>8.506391</td>
<td>10.96105</td>
</tr>
<tr>
<td>24</td>
<td>8.443350</td>
<td>3.057746</td>
<td>8.622523</td>
<td>11.11491</td>
</tr>
</tbody>
</table>

Table 3.4: Statistical moments of complexity distribution (periodic).
3. An explanation for this observation will follow from the results of Section 3.3.

4. This form of symmetry in $N_c^l$ will also be explained in Section 3.3.

5. The behaviour of the average complexity as a function of the sequence length $l$ is studied in Section 3.4.

6. This will follow from the models used in Section 3.4.

3.3 The Complexity of Random Sequences

In this section the behaviour of the maximum order complexity of random sequences is viewed at. In particular the typical complexity profile of random sequences is investigated and various properties are shown. The problem of accounting for the numbers which appear in the complexity table, Table 3.1, is shown to be equivalent to the problem of counting paths in a DeBruijn graph. Also some features of the complexity table as mentioned in the previous section are proven.

3.3.1 The Behaviour of the Maximum Order Complexity Profile

Let $s = (\alpha_0, \alpha_1, \ldots, \alpha_{l-1})$ be a sequence of length $l$ and complexity $c(s)$ with characters $\alpha_i \in A$, where the character alphabet $A$ is some finite set. In the sequel we will use $c_i$ to denote the complexity $c(\alpha_0, \alpha_1, \ldots, \alpha_{l-1})$. All the $c_i$-long subsequences are denoted by $s' = (\alpha_i, \alpha_{i+1}, \ldots, \alpha_{i+c_i-1})$, for $i = 0, 1, \ldots, l - c_i$.

As with linear complexity, the value $l/2$ forms a boundary value, which determines whether the complexity profile jumps to a higher value or remains the same.

**Proposition 3.19** If the sequence $s$ of length $l$, mentioned above, has complexity $c_l \geq l/2$ then all its $c_i$-long subsequences are unique.

**Proof.** Suppose there exists a pair of subsequences $s^i = s^j$, with $i \neq j$. We distinguish the following two situations:

i) $l$ is even, $c_i = l/2$ and $i = 0, j = l - c_i$. Then we have that $(\alpha_0, \ldots, \alpha_{l/2-1}) = (\alpha_{l/2}, \ldots, \alpha_{l-1})$. Thus $s$ is a repetition of a
sequence of length \( l/2 \) and can therefore have a complexity of at most \( l/2 - 1 \), which contradicts the assumption.

ii) \( j - i < l - c_l \), i.e. \( c_l > l/2 \) or \( c_l = l/2 \), \( j - i \neq l - c_l \). In this case the two subsequences overlap and we have \((\alpha_i, \alpha_{i+1}, \ldots, \alpha_{i+c_l-1}) = (\alpha_{i+k}, \alpha_{i+k+1}, \ldots, \alpha_{i+k+c_l-1})\), with \( 0 < k < c_l \). Thus \( s \) consists of some prefix of length \( i \), followed by a periodic sequence of period \( k \). From the definition of complexity it can be seen that the complexity of such a sequence is at most \( i + k - 1 \). Moreover, \( i + k + c_l \leq l \) so that: \( c_l \leq i + k - 1 \leq l - c_l - 1 \leq l/2 - 1 \), which again contradicts the assumption.

As an immediate consequence of Proposition 3.19 it can be seen that the complexity profile does not jump if \( c_l \geq l/2 \).

**Corollary 3.20** If the sequence \( s \) of length \( l \), mentioned above, has complexity \( c_l \geq l/2 \), then the value of the complexity will not increase if the sequence is extended with \( \alpha_l \), regardless what character it is. Equivalently \( c_{l+1} = c_l \)

**Proof.** From Proposition 3.19 it follows that \( s \) has \( l - c_l + 1 \) distinct subsequences of \( c_l \) characters. All these subsequences, except for the last one, each specify an entry in the truth table of the feedback function of the equivalent FSR. Extending the sequence with one character \( \alpha_l \) merely specifies an additional truth table entry which was not yet specified. Hence, the complexity cannot increase. \( \square \)

For the case that a sequence has a complexity strictly greater than half the sequence length an even stronger result can be obtained.

**Lemma 3.21** Let \( s = (\alpha_0, \ldots, \alpha_{l-1}, T\alpha_0, \ldots, T\alpha_{l-1}, T^2\alpha_0, \ldots, T^2\alpha_{l-1}, \ldots) \) denote a sequence in which \( T \) is an arbitrary substitution operator acting on elements of \( \mathcal{A} \). Then \( s \) has complexity \( c(s) \leq l \).

**Proof.** The sequence can indeed always be generated by an \( l \)-long FSR, where the feedback function consists of the substitution itself as depicted in Figure 3.5. \( \square \)
Proposition 3.22 If the sequence $s$ has complexity $c_l > 1/2$ then there does not exist a substitution $T$ such that any $c_l$-long subsequence of $s$ can be transformed by $T$ into some other $c_l$-long subsequence of $s$.

Proof. Suppose there exists a pair of subsequences $s^i = T s^j$, with $i \neq j$. As with Proposition 3.19, ii) any two subsequences will always overlap and we have $s^{i+k} = T s^i$, i.e. a sequence of the following form: $s = (a_0, ..., a_{i-1}, a_i, ..., a_{i+k-1}, T a_i, ..., T a_{i+k-1})$. Using Lemma 3.21 and following the same steps as with the proof of Proposition 3.19, ii), we find that $c_l \leq l/2$ which contradicts the assumption. □

An immediate consequence of Proposition 3.22 is the following result:

Corollary 3.23 A binary sequence of length $l$ and complexity $c_l > 1/2$ does not contain complementary subsequences.

Proof. The operator $T$ in this case replaces a 0 by a 1 and vice versa, which is clearly a substitution. □

From Propositions 3.19 and 3.22 one could already come to the conclusion that it is highly unlikely for a random sequence of considerable length $l$ to have a complexity exceeding $l/2$.

We now proceed by examining the behaviour of the complexity profile if it jumps to a higher value after extension with a next character. For this purpose we introduce the so called state sequence notation of a sequence.

Let $S^i = (a_i, a_{i+1}, ..., a_{i+c_l-1})$ denote the $i$th state vector of a FSR which generates the sequence $s = (a_0, a_1, ..., a_{l-1})$ of length $l$ and complexity $c_l$. The following definition is proposed:

Definition 3.3 The state sequence $S$ is defined as the sequence of states through which the FSR cycles when generating the sequence $s$, i.e. $S = (S^0, S^1, ..., S^{l-c_l+1})$. 
Note that by definition of complexity all the states in the statesequence have unique successor states and therefore the statesequence regarded as a sequence of characters has complexity equal to one. From this fact it can easily be seen that the most general form of a statesequence is:

$$ S = (S_0, S_1, \ldots, S_{i-1}, S_i, \ldots, S_{i+p-1}, \ldots, S_{i+p-1}, S_i, \ldots, S_{i+k-1}), \quad (3.5) $$

where \( p > 0, i, m \geq 0 \) and \( 0 \leq k < p \). The length \( l \) of the corresponding sequence \( s \) clearly satisfies:

$$ l = c_I + i + mp + k - 1. \quad (3.6) $$

We will now determine the maximum value to which the complexity can jump when the sequence is extended with one character.

**Lemma 3.24** If the state sequence \( S \) has a periodic part with \( m > 0 \) and a suffix with \( k > 0 \) then the complexity \( c_I \) is at most \( i + p - 1 \).

**Proof.** The conditions imply that there is at least one repeated state. Therefore Proposition 3.19 implies that \( c_I < l/2 \). Taking \( m \) and \( k \) equal to 1 yields \( l = c_I + i + p \) and hence:

$$ c_I < \frac{c_I + i + p}{2}, $$

or:

$$ c_I < i + p. $$

Letting \( m \) and \( k \) be greater than 1 does not increase the complexity as the uniqueness of successor states is not violated. \( \square \)

**Proposition 3.25** If the sequence \( s \) has complexity \( c_I < l/2 \) then extending the sequence with \( \alpha_l \) may increase the value of the complexity to a maximum of \( l - c_I \), i.e. \( c_{I+1} \leq l - c_I \).

**Proof.** The following situations may arise:

i) All states \( S^i \) are unique, implying by Corollary 3.20 that \( c_{I+1} = c_I \),

ii) \( S \) is extended periodically, again implying that \( c_{I+1} = c_I \),
iii) \( S \) is extended such that the uniqueness of successor states is violated.

In the latter situation let the violating state be denoted by \( S^{t+k} \), then it can be seen from the state sequence notation (3.5) that the longest prefix which \( S^{t+k} \) and \( S^{i+k} \) have in common consists of \( (m - 1)p + k \) states. In addition to this, the states \( S^{t-1} \) and \( S^{i+p-1} \) differ only in the first (leftmost) character, as both map to state \( S^{i} \). Therefore the longest common prefix has a length of \( (m - 1)p + k + c_{t} - 1 \) characters and hence the new complexity value will become:

\[
    c_{t+1} = (m - 1)p + k + c_{t}. \tag{3.7}
\]

Combining this fact with Lemma 3.24 and equation (3.6), we obtain:

\[
    c_{t+1} \leq (m - 1)p + k + i + p - 1
    = l - c_{t},
\]

which proves the proposition. \( \square \)

It is interesting to see that, whereas the linear complexity profile always jumps to \( l - c_{t} \), the maximum order complexity profile can also jump to values less than that. A natural question to ask now is whether all values between \( c_{t} \) and \( l - c_{t} \) are possible for \( c_{t+1} \). The following proposition answers this question by giving a more precise lowerbound to \( c_{t+1} \).

**Proposition 3.26** Let \( c_{t+1} > c_{t} \). If \( l < c_{t} + a^{c_{t}} \), the least value of \( c_{t+1} \) is \( c_{t} + 1 \), else it is \( l + 1 - a^{c_{t}} \), where \( a = |A| \) is the cardinality of the character alphabet.

**Proof.** The total number of different states \( i + p \) is clearly upperbounded by \( a^{c_{t}} \). Using equations (3.6) and (3.7) we obtain:

\[
    l \leq (m - 1)p + k - 1 + c_{t} + a^{c_{t}}
    = c_{t+1} - 1 + a^{c_{t}}.
\]

Finally:

\[
    c_{t+1} \geq l + 1 - a^{c_{t}}.
\]

However, \( c_{t+1} \geq c_{t} + 1 \) implies that this lowerbound is only significant if \( l - a^{c_{t}} \geq c_{t} \). Hence, we find that:

\[
    c_{t+1} \geq \begin{cases} 
    c_{t} + 1, & \text{if } l < c_{t} + a^{c_{t}}, \\
    l + 1 - a^{c_{t}}, & \text{else}. 
    \end{cases} \tag{3.8}
\]

\( \square \)
From the facts that $c_{t+1} - c_t = (m - 1)p + k$ and $p$ is only upperbounded by $a^{c_t} - i$ it is easily understood that every value from the lowerbound up to and including the upperbound can indeed occur for $c_{t+1}$. Hence, the number of possible values, $\#(c_{t+1})$, for $c_{t+1}$ satisfies:

$$\#(c_{t+1}) = \begin{cases} 
    l - 2c_t, & \text{if } l < c_t + a^{c_t}, \\
    a^{c_t} - c_t, & \text{else.}
\end{cases}$$

(3.9)

Proposition 3.26 shows exactly how the complexity profile jumps from one value to some other value if a sequence is extended with some character $a_i$, a phenomenon illustrated by Figure 3.6. Another way to look at the jump

![Figure 3.6: Jumps in the complexity profile.](image)

behaviour of the complexity profile is the following. Assume that $c_t = l - j$ and $c_{t-1} = j - n$, for positive $n$. What are the possible values of $j$ and $l$ for certain values of $n$, i.e. what are the possible values of $c_{t-1}$? The answer to this question is given by the following proposition.

**Proposition 3.27** Let $c_t = l - j$ and $c_{t-1} = j - n$, with $1 \leq n \leq j < l$ then $n$ is additionally restricted by the inequality $n \leq j - \log_a j$ and hence $c_{t-1} \geq \log_a j$.

**Proof.** The conditions $c_t = l - j$ and $c_{t-1} = j - n$ have been chosen because of the following reason. Proposition 3.25 states that $c_t \leq l - 1 - c_{t-1}$ if $c_{t-1} < (l - 1)/2$. Putting $c_t = l - j$ implies that $c_{t-1} \leq j - 1$ or equivalently $c_{t-1} = j - n$, with $n \geq 1$. Substituting these values for $c_t$ and $c_{t-1}$ in equation (3.7) yields:

$$l = 2j - n + (m - 1)p + k.$$  

(3.10)
But from equation (3.6) it follows that:

\[ l = j - n + i + mp + k - 1. \]  

(3.11)

From (3.10) and (3.11) we obtain \( j = i + p \), which is upperbounded by \( a^c_i = a^{j-n} \). Hence, we obtain the inequality \( j \leq a^{j-n} \) and the result follows.

\( \square \)

Figure 3.7 illustrates this backwards relationship between successive complexity values. For the binary case an example is given in Table 3.5.

<table>
<thead>
<tr>
<th>( c_i )</th>
<th>( c_{i-1} )</th>
<th>( l \leq )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( l - 1 )</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>( l - 2 )</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>( l - 3 )</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>( l - 4 )</td>
<td>2-3</td>
<td>8</td>
</tr>
<tr>
<td>( l - 5 )</td>
<td>3-4</td>
<td>10</td>
</tr>
<tr>
<td>( l - 6 )</td>
<td>3-5</td>
<td>12</td>
</tr>
<tr>
<td>( l - 7 )</td>
<td>3-6</td>
<td>14</td>
</tr>
<tr>
<td>( l - 8 )</td>
<td>3-7</td>
<td>16</td>
</tr>
<tr>
<td>( l - 9 )</td>
<td>4-8</td>
<td>18</td>
</tr>
</tbody>
</table>

Table 3.5: Backwards relationship for the binary case.
3.3.2 On Numbers and their Relation with DeBruijn Graphs

Although up to this point we have gained insight into the behaviour of the maximum order complexity profile, we still have no explanation for the numbers as they appear in Table 3.1. The results of the previous section indicate that one should try to count the sequences of certain length which jump from one complexity value to another or remain at the same complexity value. To this end, let $D^j_{l,n}$ denote the number of sequences of length $l$ and complexity $j$, which take on complexity $n$ if they are extended with one character. Using once more equations (3.6) and (3.7), one obtains if $n > j$:

$$i + p = l + 1 - n, \quad (3.12)$$
$$\quad (m - 1)p + k = n - j. \quad (3.13)$$

With these equations one can easily obtain all combinations of $i, p, m$ and $k$ for given $l, j$ and $n$ and the problem is reduced to counting all sequences with the obtained values for $i$ and $p$. This problem is equivalent to finding a path through $i + p$ distinct states in a DeBruijn graph of order $j$, where the last $p$ states form a cycle, i.e. a closed path. There seems to be no simple general solution to this problem for arbitrary order and alphabet size and therefore this approach seems unfruitful.

**Example 3.11** The values of $D^j_{l,n}$ in the binary case are given by Table 3.6. For $j = 3$ and $n = 16$ we find from (3.12) and (3.13) that $i + p = 4$ and $(m - 1)p + k = 13$ and we obtain the following combinations of $i, p, m$ and $k$:

<table>
<thead>
<tr>
<th>$i$</th>
<th>$p$</th>
<th>$m$</th>
<th>$k$</th>
<th># seq</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4</td>
<td>4</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>5</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>7</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>13</td>
<td>0</td>
<td>6</td>
</tr>
</tbody>
</table>

By examining the DeBruijn graph of order 3 the number of sequences with each combination can be counted. Note that these numbers depend only on the difference $l + 1 - c_{l+1}$ and $c_i$ itself (and of course on the alphabet size). From Table 3.6 it can be seen that indeed one half of the sequences remains at the same complexity value after extension with one bit and the other half jumps to the complexity values as described by Proposition 3.26, if $j \leq 3$. 
Table 3.6: Values of $D_{ij}$ in the binary case.

<table>
<thead>
<tr>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>19</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>18</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>17</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>16</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>15</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>14</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>13</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>11</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>9</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

The Shortest PSR Equivalent
Table 3.6 (cont.): Values of \( D^u \) in the binary case.

<table>
<thead>
<tr>
<th>( m )</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>19</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>18</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>17</td>
<td>0</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>16</td>
<td>0</td>
<td>0</td>
<td>28</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>15</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>86</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>14</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>250</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>13</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>680</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1792</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Complexity of Random Sequences
The second approach in explaining the numbers of Table 3.1 is to examine the numbers $D_{j,i}^l$ for $j < l/2$. As was mentioned with Proposition 3.25, there are two situations where the complexity remains the same if $c_i < l/2$, i.e.:

1. the sequence is extended periodically,

2. all the states in the state sequence are unique and the number of states is less than $a^{c_i}$.

In the first situation $D_{j,i}^l$ clearly remains the same as there is only one choice for the next character. In the second situation $D_{j,i}^l$ increases with $l$ as there are sequences for which there is more than one choice for the next character. Hence, it seems meaningful to examine how $D_{j,i}^l$ grows with increasing sequence length $l$. To this end, let $\Delta_j^l = D_{j,i}^{l+1} - D_{j,i}^l$. For the binary case $\Delta_j^l$ is given in Table 3.7 for some values of $l$ and $j$. This table exhibits some interesting properties, e.g. if $l = c_i + 2^{c_i} - 1$ then $\Delta_j^l = 2^{2^{c_i}-1}$. This is exactly equal to $2^j G_j$ where $G_j$ is the number of DeBruijn sequences of order $j$. The explanation is that for this length all possible phases of all DeBruijn sequences contribute to $D_{j,i}^l$.

For values of $l$ less than $c_i + 2^{c_i} - 1$, the number of sequences which add their contributions to $D_{j,i}^l$ becomes more and more difficult to count. The problem is now to count incomplete paths in a DeBruijn graph of order $c_i$, i.e. counting paths with $2^{c_i} - 1$, $2^{c_i} - 2$, etc. states. For example, if $l = c_i + 2^{c_i} - 2$ then we can deduce from the table $\Delta_j^l = G_j(2(2^{c_i} - 1) + 2^{c_i} - 2)$. In fact we have the following proposition:

**Proposition 3.28** If a binary sequence has complexity $c_i$ and length $l = c_i + 2^{c_i} - 2$ then the number of different state sequences, having $2^{c_i} - 1$ states, is exactly equal to $G_j(2(2^{c_i} - 1) + 2^{c_i} - 2)$, where $G_j$ is the number of binary DeBruijn sequences of order $j$.

**Proof.** There are evidently $G_j$ cyclicly inequivalent state sequences with $2^{c_i}$ states. Both the all-zero and the all-one states can always be deleted, no matter where they are positioned, because the successor of either state can also be directly the successor of its predecessor, viz. these states have cycle length one. Deleting either state gives two choices and with each choice the resulting state sequence may be shifted cyclicly over $2^{c_i} - 1$ possible phases, hence accounting for the term $2(2^{c_i} - 1)$.

The other states, however, can only be deleted if they start or end the state sequence, because they do not have the property that their successors
Table 3.7: Some values of $\Delta_j^l$ in the binary case.
can directly be the successors of their predecessors, viz. these states have
cycle length greater than one. Moreover, both deletions result in the same
state sequence. As there are $2^c - 2$ states beside the all-zero and all-one
states, the total number is explained.

Shorter paths can in principle be counted in the same way, however, one
has to take into account pairs, triplets, etc. of states that form cycles of
length 2, 3, etc., which can be deleted together at any position in the state
sequence. Deriving a general expression, however, seems to be a difficult
problem. Hence, the same has to be said for the numbers $D_{i,j}^l$.

From Table 3.7 it can be seen that the maximum value of $\Delta_j^l$ occurs if
$l = 2^c$, and that $\Delta_j^l$ is divisible by $G_j$ if and only if $l \geq 2^c$. We conjecture
that in the binary case these properties hold for all $c_l \geq 2$.

3.3.3 The Complexity Table Reconsidered

Table 3.1 as shown in Section 3.2 was already examined and partly ex­
plained in that section. In particular the limit value $L_f^l$ follows immediately
from the results of the previous subsection.

Let us now consider the symmetry property $N_{c_l}^l = N_{c_{l+1}}^l$. First, there is
the following lemma:

Lemma 3.29 If a sequence of length $l$ has complexity $c_l < l/2$ and after
extension with one character $c_l+1 > c_l$, then $\forall j \geq 0 \left[ D_{c_l,c_{l+1}}^l = D_{c_l+1,c_{l+1}+j}^{l+i} \right]$.

Proof. The proof follows immediately from equation (3.12), which states
that $i + p = l + 1 - c_{l+1} = l + j + 1 - (c_{l+1} + j)$, whereas the number of
sequences depends only on the value of $i + p$. □

Now the symmetry property can be proven.

Proposition 3.30 The number of sequences of length $l + n$ and complexity
$l + n - j$ is equal to the number of sequences of length $l$ and complexity
$l - j$, for all positive $n$, if $j \leq l/2$.

Proof. First note that the maximum complexity value is equal to $l - 1$,
which occurs only with sequences of the following type:

$s = (\alpha, \alpha, \ldots, \alpha, \beta), \ \alpha, \beta \in A, \ \alpha \neq \beta$. 

\[ s = (\alpha, \alpha, \ldots, \alpha, \beta), \ \alpha, \beta \in A, \ \alpha \neq \beta. \]
This follows from the definition of complexity. Hence, the number of such sequences is equal to \(a(a-1)\), with \(a = |A|\).

Next, note that from the backward relationship of Proposition 3.27 it follows that:

\[
N_{l-j}^l = aN_{l-j}^{l-1} + \sum_{i=I_j}^{j-1} D_{i,l-j}^{l-1}, \quad 1 \leq j \leq l/2, \quad (3.14)
\]

with \(I_j = \lfloor \log_a j \rfloor \).

Equation (3.14) can be applied repeatedly for decreasing values of the sequence length \(l'\), until this length reaches its least possible value, determined by the condition \(j \leq l'/2\). This yields the following expression for \(N_{l-j}^l\):

\[
N_{l-j}^l = a \left( aN_{l-j}^{l-2} + \sum_{i=I_j}^{j-2} D_{i,l-j}^{l-2} \right) + \sum_{i=I_j}^{j-1} D_{i,l-j}^{l-1} \\
= a^{j-1}N_{l-j}^{l-j+1} + \sum_{k=0}^{j-2} a^k \sum_{i=I_{l-j-k}}^{j-k-1} D_{i,l-j}^{l-k-1} \\
= a^{j-1}(a - 1) + \sum_{k=0}^{j-2} a^k \sum_{i=I_{l-j-k}}^{j-k-1} D_{i,l-j}^{l-k-1}. \quad (3.15)
\]

From equation (3.15) and Lemma 3.29 the result now follows.

From the foregoing the limit value \(L_j^-\) is almost self-evident:

**Corollary 3.31** The number of sequences of length \(l\) and complexity \(l - j\), \(N_{l-j}^l\), is always less than or equal to the maximum value \(L_j^-\), with equality if and only if \(l \geq 2j\).

**Proof.** For \(l \geq 2j\) equation (3.15) is in fact also the exact expression for \(L_j^-\), as for greater values of \(l\) its righthandside remains constant. However, if \(l < 2j\) equation (3.14) is no longer valid, because \(l - j = c_l < l/2\), in which case the number of \(D\)'s that contribute is less than maximum. For example if \(j = l/2 + 1\) then \(D_{j-1,l-j}^{l-1}\) does not exist.

### 3.3.4 Bounds on \(L_j^+\) and \(L_j^-\)

The numbers \(L_j^+\) and \(L_j^-\) play a fundamental role in the maximum order complexity profile. It seems to be difficult to obtain analytical expressions,
as was mentioned before. The reason for this is the difficulty of counting incomplete paths in a DeBruijn graph; paths that may also end in a cycle.

Here we present upper- and lowerbounds on $L_j^+$ and $L_j^-$, which illustrate their behaviour. For clarity we recall what the meaning of these numbers is:

- $L_j^+$ is the number of sequences which have maximum order complexity $j$ and length $\geq j + a$, where $a$ is the cardinality of the character alphabet.

- $L_j^-$ is the number of sequences which have maximum order complexity $l - j$ and length $l \geq 2j$.

Bounds on $L_j^+$ are easily derived, viz. $L_j^+$ is clearly greater than the number of sequences which constitute all different phases of all DeBruijn sequences of order $j$, but less than the number of all possible FSR’s of length $j$ with all possible initial fills. In the general case of arbitrary alphabet size the number of DeBruijn sequences is derived in e.g. [Lint 74, Ch. 9]. Hence, we have the following bounds:

$$(a!)^{a^{j-1}} < L_j^+ < a^{a^{j+1}}. \quad (3.16)$$

In contrast with $L_j^+$, which has a double exponential behaviour, as can be seen from (3.16), $L_j^-$ increases only exponentially with increasing $j$. From equation (3.15) we immediately obtain the lowerbound:

$$L_j^- \geq a^{j-1}(a - 1). \quad (3.17)$$

In order to establish an upperbound on $L_j^-$ we first upperbound $D_{n,j-1}^{l-1}$ by:

$$D_{n,l-j}^{l-1} \leq ja^n a^{j-1}. \quad (3.18)$$

This result can be explained as follows. Counting $D_{n,l-j}^{l-1}$ is equivalent with counting distinct paths in a DeBruijn graph of order $n$ and alphabet size $a$, with $i + p$ distinct states, the last $p$ of which form a cycle, for all combinations of $i$ and $p$ such that $i + p = j$ and $p \geq 1$. For each of the $j$ different values of $p$ the number of initial states is clearly upperbounded by $a^n$ and for each successive state there are $a$ choices at maximum. For example $D_{l-1}^{l-1} = 2a(a - 1)$, whereas the upperbound (3.18) yields $D_{l-1}^{l-1} \leq 2a^2$. 
Next, we substitute the upperbound (3.18) in equation (3.15) and obtain:

\[
L_j^\leq a^{j-1}(a - 1) + \sum_{k=0}^{j-2} a^k \sum_{i=I_j-k}^{j-k-1} \sum_{i=I_j-k} a^{i}\]

\[
\leq a^{j-1}(a - 1) + \sum_{k=0}^{j-2} (j - k) a^{j-1} \sum_{i=I_j-k} a^{i}
\]

\[
\leq a^{j-1}(a - 1) + \sum_{k=0}^{j-2} (j - k) a^{j-1} \{ a^{j-k} - (j - k) \} / (a - 1)
\]

\[
= \frac{ja^{2j}}{(a - 1)^2} - \frac{a^j(a - 1)}{(a - 1)^3} + a^{j-1}(a - 2) - \frac{a^{j-1}j(j + 1)(2j + 1)}{6}. \quad (3.19)
\]

For example for the binary case, \(a = 2\), and \(j = 2\), the upperbound (3.19) yields \(L_j^\leq 10\), whereas the exact value is 8.

It appears that the integer sequences

1) 1, 2, 10, 105, 3823, ....

2) 1, 4, 14, 43, 125, 340, 896, ....

are fundamental to the maximum order complexity profile in the binary case. These integer sequences do not appear in Sloane's book on this topic [Sloa 73], although this book contains an enormous amount of such sequences. They may, however, be considered as an interesting contribution to the work in that area.

3.4 The Complexity of Random Sequences; a Statistical Approach

3.4.1 Relation with Ziv-Lempel Complexity

In [Lemp 76] Lempel and Ziv introduced a new complexity measure for finite sequences over arbitrary alphabets. This Ziv-Lempel complexity is defined as the number of subsequences the sequence is parsed into, in such a way that not a single subsequence occurs earlier in the sequence.

In this section the relation between Ziv-Lempel complexity and maximum order complexity as defined in Section 3.1 will be established and used to obtain a lower- and an upperbound on the expected complexity of random sequences.
3.4.1.1 Lowerbound on the Expected Complexity

Consider a sequence $s = (\alpha_0, \alpha_1, \ldots, \alpha_{t-1})$ of length $l$ with characters $\alpha_i \in \mathcal{A}$, where $\mathcal{A}$ is the alphabet from which the sequence is built and which has a cardinality $|\mathcal{A}| = a$. Suppose $s$ has a complexity of $c$ characters, then obviously all subsequences of length $\geq c$ are unique. Therefore, as a result of the Ziv-Lempel parsing process, the substrings cannot have a length exceeding $c$. Hence, we have the following relation between complexity $c$ and Ziv-Lempel complexity $c_{ZL}$:

$$c_{ZL} \geq \frac{l}{c} \text{ or: } c \geq \frac{l}{c_{ZL}}. \quad (3.20)$$

This result can be explained as follows: because of the innovation character that ends each subsequence, the longest subsequence in fact determines $c$. As the number of subsequences equals $c_{ZL}$ by definition, the sequence length divided by the the number of subsequences therefore is a lowerbound on the length of the longest subsequence and hence on $c$.

Expression (3.20) is useful to establish a lowerbound on the expected complexity of random sequences, i.e.:

$$\mathcal{E}(c) \geq \mathcal{E}\left(\frac{l}{c_{ZL}}\right) \geq \frac{l}{\mathcal{E}(c_{ZL})} = \frac{l}{\log_a l} = \log_a l. \quad (3.21)$$

In (3.21) we have taken $\mathcal{E}(c_{ZL})$ to be exactly $l/\log_a l$, which is of course only the case asymptotically.

3.4.1.2 Upperbound on the Expected Complexity

Let us suppose that a sequence $s$ is parsed into $c_{ZL}$ subsequences and there is one longest subsequence $\hat{s}$ with length $\hat{l}$. Obviously $c \leq \hat{l}$ holds, even with strict inequality if this longest subsequence is not exhaustive, as may be the case for the last subsequence. The minimally consumed sequence length by all the remaining subsequences, $l_m$, satisfies:

$$l_m = a + 2a^2 + 3a^3 + \cdots + ka^k$$

$$= \sum_{i=1}^{k} ia^i$$

$$= \frac{ka^{k+1}}{a-1} - \frac{a^{k+1} - a}{(a-1)^2}, \quad (3.22)$$
for $k$ such that:
\[ a + a^2 + a^3 + \cdots + a^k = c_{2L} - 1, \]
or equivalently:
\[ k + 1 = \log_a(c_{2L}(a - 1) + 1). \] (3.23)

For the binary case ($a = 2$) equation (3.22) combined with (3.23) reduces to:
\[ l_m = (c_{2L} + 1) \log_2(c_{2L} + 1) - 2c_{2L}. \] (3.24)

We now have the following upperbound on $c$:
\[ c \leq \hat{l} = l - l_m = l - (c_{2L} + 1) \log_2(c_{2L} + 1) + 2c_{2L}. \] (3.25)

Assuming that $E(c_{2L}) \to l/\log_a l$ for large $l$, using (3.25) we finally arrive at:
\[ E(c) \leq \frac{l}{\log_2 l}(2 + \log_2 \log_2 l). \] (3.26)

The upperbound on the expected complexity as given by (3.26) is not very significant. The problem here seems to be that the actual probability distribution of the subsequence lengths is not taken into account explicitly. The probability of a very large and exhaustive maximum subsequence is likely to be very small. In the next section a more meaningful upperbound will be established using heuristic statistical models.

### 3.4.2 Heuristic Statistical Models

The motivation to look for heuristic statistical models comes from the fact that the direct analytical approach from the previous section has not answered all the questions which arised from Tables 3.1 to 3.4 from Section 3.2. Besides this we have learned from Golomb's book [Golo 67, pp. 175-189] that this is a useful alternative, which can provide insight into the matter if it is combined with and supported by relevant statistical experiments for the real situation.

As a start, a demonstration of the birthday statistics follows, which seems quite fundamental for the rest of this section.

#### 3.4.2.1 Birthday Statistics

Consider drawing from a homogeneous distribution of $A$ distinct elements. The question that is posed here is: 'How often do we expect to draw before we have two identical elements?'. So whenever an element is drawn that
has already been drawn before, the process of drawing elements is stopped. The answer is given in this section.

Let $Pr[k]$ denote the probability that the number of draws is $k$, obviously $2 \leq k \leq A + 1$, then we have:

$$Pr[k] = \frac{A!}{(A - k + 1)!} \left(\frac{k - 1}{A^k}\right).$$

Indeed:

$$\sum_{k=2}^{A+1} Pr[k] = \sum_{j=1}^{A} \binom{A}{j} \frac{j!}{A^{j+1}} = 1,$$

according to [Prud 86, 4.2.3.30].

Now let $E(k)$ denote the expected number of draws, then:

$$E(k) = \sum_{j=1}^{A} \binom{A}{j} \frac{j!}{A^{j+1}} (j + 1) = \sum_{j=1}^{A} (j + 1) f_j$$

$$= A + 1 - \sum_{k=1}^{A-1} \sum_{j=1}^{k} f_j.$$

Again from [Prud 86, 4.2.3.30] we have:

$$\sum_{j=1}^{k} f_j = 1 - \frac{(k + 1)!}{A^{k+1}} \binom{A}{j},$$

so that:

$$E(k) = 2 + \sum_{k=2}^{A} \binom{A}{k} k! A^{-k}$$

$$= 2 + \sum_{k=2}^{A} \prod_{j=0}^{k-1} \left(1 - \frac{j}{A}\right).$$

The product in (3.27) can very well be approximated as follows:

$$\prod_{j=0}^{k-1} \left(1 - \frac{j}{A}\right) = e^{\sum_{j=0}^{k-1} \ln(1 - \frac{j}{A})} \approx e^{\sum_{j=0}^{k-1} - \frac{j}{A}} = e^{-\frac{k(k-1)}{2A}}.$$
Table 3.8: Upperbound on the number of draws.

<table>
<thead>
<tr>
<th>$A$</th>
<th>$\mathcal{E}(k)$</th>
<th>(3.29)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2.500</td>
<td>3.772</td>
</tr>
<tr>
<td>4</td>
<td>3.219</td>
<td>4.507</td>
</tr>
<tr>
<td>8</td>
<td>4.245</td>
<td>5.545</td>
</tr>
<tr>
<td>16</td>
<td>5.704</td>
<td>7.013</td>
</tr>
<tr>
<td>32</td>
<td>7.774</td>
<td>9.090</td>
</tr>
<tr>
<td>64</td>
<td>10.706</td>
<td>12.027</td>
</tr>
<tr>
<td>128</td>
<td>14.855</td>
<td>16.180</td>
</tr>
<tr>
<td>256</td>
<td>20.726</td>
<td>22.053</td>
</tr>
<tr>
<td>512</td>
<td>29.031</td>
<td>30.359</td>
</tr>
<tr>
<td>1024</td>
<td>40.776</td>
<td>42.106</td>
</tr>
</tbody>
</table>

where $\preceq$ denotes an approximation by a tight upperbound. Hence:

$$\mathcal{E}(k) \preceq 2 + \sum_{k=2}^{A} e^{-\frac{k(k-1)}{2A}}.$$  

This summation can again be approximated by the following integral:

$$\sum_{k=2}^{A} e^{-\frac{k(k-1)}{2A}} \leq \int_{1}^{A} e^{-\frac{k(k-1)}{2A}} dk = \sqrt{2A} e^{\frac{1}{8A}} \int_{\frac{1}{2\sqrt{2A}}}^{\frac{2A-1}{2\sqrt{2A}}} e^{-z^2} dz.$$  

(3.28)

As can be seen the expression at the right-hand side goes to $\sqrt{2A} \cdot 1 \cdot \frac{1}{2} \sqrt{\pi}$ if $A \to \infty$, hence we have the upperbound:

$$\mathcal{E}(k) \leq 2 + \sqrt{\frac{\pi}{2}} A.$$  

(3.29)

In Table 3.8 the upperbound (3.29) is compared to the exact values of the expected number of draws for a number of values of $A$. It can be seen that the upperbound can even be tightened by subtracting a constant from it; a result from the integral approximation in (3.28).

Concluding, one can say that if elements are drawn independently at random, with replacement, from a set containing $A$ elements, we expect to obtain about $1 + \sqrt{\frac{\pi}{2}} A$ elements before an element is repeated. In the case that the elements are substrings of a string of characters from some alphabet, we can say that the total stringlength may vary typically between $\mathcal{E}(k) + \log_a A - 1$ and $\mathcal{E}(k) \log_a A$, where the logarithm base $a$ is the cardinality of the character alphabet. With $A = a^c$ the total stringlength $L$ will
therefore be bounded as follows:

\[ a^y \sqrt{\frac{\pi}{2}} \leq L \leq ca^y \sqrt{\frac{\pi}{2}}. \] (3.30)

For large enough \( L \) one can consider \( c \) as a result and use (3.30) to obtain bounds on \( c \). For example if \( L = 2^{100} \) one obtains: \( 185 \leq c \leq 200 \). This result looks obvious considering (3.29) and it illustrates the "square root law" in birthday statistics.

### 3.4.2.2 Periodic Random Sequences

Consider the following experiment: we want to draw a sequence of \( l \) distinct characters independently and at random from some alphabet. We start with an alphabet \( \mathcal{A} \) with \( |\mathcal{A}| = A = a^c \) characters (\( a \)-ary symbols say). Obviously, the condition \( c = \lfloor \log_a l \rfloor \) must be fulfilled. Now if, while drawing characters, we encounter a repetition we start all over again, but this time with an alphabet of \( a^{c+1} \) characters, then \( a^{c+2} \) characters, etc. until there are \( l \) unique characters. The question we ask ourselves now is: what is the expected exponent \( n \) of \( a \) we end the experiment with? Obviously, we have that \( c \leq n < \infty \). The approach to answering this question is rather straightforward probability theory.

The probability \( p_n \) of having a sequence \( (a_0, a_1, \ldots, a_l-1) \) with \( l \) unique symbols from some alphabet \( \mathcal{A} \), with \( |\mathcal{A}| = a^n = A \), is:

\[ p_n = \prod_{i=1}^{l-1} \left( 1 - \frac{i}{A} \right); \quad n \geq \lfloor \log_a l \rfloor. \]

The probability \( Pr[\hat{n}] \) of having \( l \) unique symbols from an alphabet with \( A = a^n \), but with no alphabet of lower cardinality is:

\[ Pr[\hat{n}] = p_n \prod_{j=c}^{n-1} q_j. \] (3.31)

In the above equation (3.31) \( q_j = 1 - p_j \) and \( p_j \) is as defined above. The induced probability distribution (3.31) can be seen as a generalisation of the geometric distribution, where all the probabilities \( p_j \) are constant. It can easily be verified that all the probabilities sum up to 1, i.e.:

\[ \sum_{n=c}^{\infty} p_n \prod_{j=c}^{n-1} q_j = 1 - \prod_{j=c}^{\infty} q_j = 1, \]
because \( p_j \to 1 \) for \( j \to \infty \). An expression for the expected exponent of the alphabet cardinality can now be obtained:

\[
\mathcal{E}(n) = \sum_{n=c}^{\infty} np_n \prod_{j=c}^{n-1} q_j
\]

\[
= c + q_c + q_c q_{c+1} + \cdots
\]

\[
= c + \sum_{n=c}^{\infty} \prod_{j=c}^{n} q_j
\]

\[
= c + \sum_{n=c}^{\infty} \prod_{j=c}^{n} \left\{ 1 - \prod_{i=1}^{i-1} \left( 1 - \frac{i}{a^j} \right) \right\}.
\]

(3.32)

The above expression (3.32) is rather complicated, therefore we establish an upperbound, which will turn out to be a good approximation.

First note that \( \forall j \geq 0 \ [q_j \leq 1] \) and observe that:

\[
q_j = 1 - \prod_{i=1}^{l-1} \left( 1 - \frac{i}{a^j} \right) \leq \left( \frac{l}{2} \right) a^{-j},
\]

(3.33)

and also:

\[
\prod_{j=c}^{n} q_j \leq q_n.
\]

(3.34)

Equation (3.32) can now be upperbounded by first applying (3.34) and then breaking up the summation into two parts, the first one having 1 as an upperbound for \( q_j \) and the second one using (3.33), i.e.:

\[
\mathcal{E}(n) \leq c + \sum_{n=c}^{\infty} q_n
\]

\[
\leq c + \sum_{n=c}^{c+N-1} 1 + \sum_{n=c}^{\infty} \left( \frac{l}{2} \right) a^{-n}
\]

\[
\leq c + N + \frac{a}{a-1}
\]

\[
= \left\lfloor \log_a l \right\rfloor + \left[ \log_a \frac{l-1}{2} \right] + \frac{a}{a-1}
\]

\[
\leq 2 \log_a l - \log_a 2 + \frac{a}{a-1} + 2,
\]

(3.35)

where \( N = \left\lfloor \log_a \frac{l-1}{2} \right\rfloor \). The upperbound (3.35) is quite good as is shown in Table 3.9 for \( a = 2 \).
Table 3.9: Upperbound on expected alphabet cardinality.

Valid sequences of the type discussed in this section were called periodic random sequences because of the fact that these sequences may be regarded as consisting of one period of a periodic sequence which has no repeated subsequences. As with Section 3.4.2.1, the relation with the expected complexity of random sequences is that the characters $\alpha_i \in \mathcal{A}$ may be seen as substrings of width $n$ digits. All these substrings are unique as for example is the case with the DeBruijn sequences [DeBr 46] of order $n$.

The uniqueness of substrings is of course an unnecessary constraint, as complexity does not imply uniqueness of substrings but uniqueness of successor symbols. This more realistic situation is treated in the next section.

3.4.2.3 Random Sequences

Again consider the experiment of Section 3.4.2.2, but now we want to draw a sequence of $l$ characters at random from some alphabet, such that every character has a unique successor (except of course for the last character, which has no successor).

We now start with an alphabet $\mathcal{A}$ with $|\mathcal{A}| = a > 1$ characters. If we draw a character whose predecessor already occurs in the sequence, but with a different successor there, we start all over again, but now with an alphabet of $a^2$ characters, then $a^3$ characters, etc. until we have drawn a sequence with unique successors.

Here also the question is: what is the expected exponent $n$ of the alphabet cardinality we end with? Note that $1 \leq n < \infty$ in this case.

An example of a valid sequence of length 11 according to the above description is: $(\alpha \beta \gamma \alpha \beta \gamma \alpha \beta \gamma \alpha \beta)$, with $\alpha, \beta, \gamma \in \mathcal{A}$. This sequence is a concatenation of $3^2$ periods of a subsequence $(\alpha \beta \gamma)$ which has period 3. Another example is the following sequence of the same length: $(\alpha \beta \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma)$, with $\alpha, \beta, \gamma \in \mathcal{A}$. Here the sequence is a concatenation of the transient sequence $(\alpha \beta)$ of length 2 and 9 periods of the sequence $(\gamma)$ of period 1.
Let $t, p$ denote the lengths of the transient- and periodic parts respectively, then it can easily be seen that:

\[ t + mp = l; \quad m, p, l \in \mathbb{N}, \]
\[ t \in \mathbb{N} \cup \{0\}. \]

Next consider the case that $A = |A| < l$ and let $S_{(t,p)}$ denote the set of valid sequences with transient length $t$ and period $p$. Clearly, $t + p \leq A$ and we have the following sets of valid sequences:

\[
\begin{align*}
S_{(0,1)}, & \quad S_{(0,2)}, & \ldots & \ldots & \quad S_{(0,A)}, \\
S_{(1,1)}, & \quad S_{(1,2)}, & \ldots & \ldots & \quad S_{(1,A-1)}, \\
\vdots & \ldots & \ldots & \ldots & \\
S_{(A-2,1)}, & \quad S_{(A-2,2)}, & \quad S_{(A-1,1)}. \\
\end{align*}
\]

From the definition of this type of sequence and the fact that $A < l$, it can be seen that all these sets are mutually non-overlapping. The cardinality of the sequence sets is easily determined by simple counting arguments, yielding:

\[ |S_{(t,p)}| = \prod_{i=0}^{t+p-1} (A - i). \]

The cardinality only depends on the sum $t + p$. Let $T(A)$ denote the total number of valid sequences, then this number is:

\[ T(A) = \sum_{j=1}^{A} \prod_{i=0}^{j-1} (A - i). \]

Next consider the case that $A \geq l$. Then we have the following sets of valid sequences:

\[
\begin{align*}
S_{(0,1)}, & \quad S_{(0,2)}, & \ldots & \ldots & \quad S_{(0,l)}, \\
S_{(1,1)}, & \quad S_{(1,2)}, & \ldots & \ldots & \quad S_{(1,l-1)}, \\
\vdots & \ldots & \ldots & \ldots & \\
S_{(l-2,1)}, & \quad S_{(l-2,2)}, & \quad S_{(l-1,1)}. \\
\end{align*}
\]

Note that in this case the sets $S_{(0,l)}, S_{(1,l-1)}, \ldots, S_{(l-1,1)}$ all contain the same sequences, as there is only one period in the sequence, i.e. we are back at the situation of Section 3.4.2.2. So the total number of valid sequences in this case is:
The Shortest FSR Equivalent

\[ T(A) = \prod_{i=0}^{l-1} (A - i) + \sum_{j=1}^{l-1} j \prod_{i=0}^{j-1} (A - i). \]

After this long introduction we are now able to determine the probability \( p_n \) of having a valid sequence with characters from some alphabet \( A \) with \( |A| = A = a^n \) and \( n \geq 1 \), i.e.:

\[
p_n = \frac{T(a^n)}{a^{nl}} = \begin{cases} 
\frac{1}{a^{nl}} \left( \sum_{j=1}^{a^n} j^{l-1} \prod_{i=0}^{j-1} (a^n - i) \right); & n < \log_a l, \\
\frac{1}{a^{nl}} \left( \prod_{i=0}^{l-1} (a^n - i) + \sum_{j=1}^{l-1} j \prod_{i=0}^{j-1} (a^n - i) \right); & n \geq \log_a l.
\end{cases}
\]

(3.36)

In order to determine an upperbound on the expected exponent of the alphabet cardinality, we make use of the results of Section 3.4.2.2 for the generalised geometric distribution, i.e.:

\[ \mathcal{E}(n) = 1 + \sum_{n=1}^{\infty} \prod_{j=1}^{n} q_j. \]

We first concentrate on the first \( \lfloor \log_a l \rfloor \) terms. Then \( p_n \) can be written as:

\[ p_n = \frac{1}{A!} \sum_{j=1}^{A} j \binom{A}{j} j! < \frac{A!2^A}{A^l}. \]

It can be seen that \( p_n \) increases monotonically with \( A \). So considering the fact that \( A = l - 1 \) is the maximum for this case and using \( A! \leq A^{A+\frac{1}{2}} e^{-A} \sqrt{2\pi} \) we have:

\[ p_n \leq \sqrt{2\pi(l-1)} \left( \frac{2}{e} \right)^{l-1}, \]

(3.37)

which shows that \( p_n \) goes to zero for \( l \to \infty \).

Now consider only the first \( N = \lfloor \log_a l \rfloor - 1 \) terms in the summation, then we have:

\[ 1 + \sum_{n=1}^{N} \prod_{j=1}^{n} q_j \geq 1 + \sum_{n=1}^{N} \prod_{j=1}^{N} q_j. \]
However, using (3.37) we find that:

\[ \prod_{j=1}^{N} q_j \geq \prod_{j=1}^{N} \left\{ 1 - \sqrt{2\pi(l-1)} \left( \frac{2}{e} \right)^{l-1} \right\} \]

\[ \geq 1 - (\lceil \log_a l \rceil - 1) \sqrt{2\pi(l-1)} \left( \frac{2}{e} \right)^{l-1}. \]  

(3.38)

It can be seen from (3.38) that the product goes to 1 for \( l \rightarrow \infty \). As the upperbound for this product is also 1, we are in fact back at the situation of Section 3.4.2.2 if \( l \) is large enough. It then immediately follows that:

\[ \mathcal{E}(n) \leq \lceil \log_a l \rceil + \sum_{n=N+1}^{\infty} \prod_{j=N+1}^{n} q_j. \] 

(3.39)

From the expression for \( p_n \) (3.36) the coefficients of \( A^{-1}, A^{-2} \) can be determined, i.e.:

\[ p_n = 1 - \frac{(l-2)(l-1)}{2A} + \frac{(l-2)(3l^3 - 16l^2 + 25l + 12)}{24A^2}. \] 

(3.40)

The above equation (3.40) immediately results in the following upperbound for \( q_j \):

\[ q_j \leq \left( \frac{l-1}{2} \right) a^{-j}. \] 

(3.41)

Using (3.39) and (3.41) and following the same steps as in Section 3.4.2.2 we finally arrive at:

\[ \mathcal{E}(n) \leq \lceil \log_a (l-1) \rceil + \left\lfloor \log_a \frac{l-2}{2} \right\rfloor + \frac{a}{a - 1} \]

\[ \leq 2 \log_a (l-1) - \log_a 2 + \frac{a}{a - 1} + 2. \] 

(3.42)

Comparing the upperbound (3.42) with (3.35) from Section 3.4.2.2 shows that, asymptotically, there is only a negligible difference between the two situations.

### 3.4.2.4 Mutually Excluding m-tuples

As we have seen before, complexity \( c \) implies that there are at least two subsequences in a sequence of characters from some finite alphabet, which
both have a length of $c - 1$ and have different successor characters. Equivalently, a sequence of complexity of $c$ cannot have two or more identical subsequences of length $c$ with different successor characters.

The statistical model described in this section, models subsequences as elements from a set, where the elements are grouped as m-tuples. From this set a string is built in which only one member from each m-tuple (regardless which one) may occur. A string with this property is called a valid string.

For ease of understanding we start with a set $P$ of pairs, modelling the situation of binary sequences.

Let $P$ be a set of elements, where each element is paired with one other element. So $|P| = 2M$ and pairs are denoted by $(\alpha, \bar{\alpha})$. In order to determine the number of valid strings of certain length, we first determine the number of valid strings of length $l$, containing exactly $k$ different elements. Let $N_M(l, k)$ denote this number; of course $k$ must satisfy: $1 \leq k \leq M$. We then have:

$$N_M(l, k) = 2^k \binom{M}{k} K(l, k), \quad (3.43)$$

with:

$$K(l, k) = \sum_{i=1}^{k} \binom{k}{i} i^l (-1)^{k-i}. \quad (3.44)$$

A result explained as follows:

- $\binom{M}{k}$ ways to choose $k$ pairs out of $M$,
- $2^k$ ways to choose a member from each of the $k$ pairs,
- $K(l, k)$ ways of assigning exactly $k$ elements to $l$ places.

The number $K(l, k)$ can be determined using the inclusion-exclusion principle; it also satisfies:

$$\sum_{i=1}^{k} K(l, k) \binom{k}{i} = k^l.$$

Now let $T_2(l, M)$ denote the total number of valid sequences of length $l$, using a set of $M$ pairs, then we have:

$$T_2(l, M) = \sum_{k=1}^{M} N_M(l, k) = \sum_{k=1}^{M} \sum_{i=1}^{k} 2^k \binom{M}{k} \binom{k}{i} i^l (-1)^{k-i}.$$
The extension of (3.46) to the general (a-ary) case follows immediately from equations (3.43), (3.44) and (3.45). In this case there is a set $A$ of elements which is divided into disjoint a-tuples and every element is said to have $a - 1$ *tuple companions*, i.e. other elements belonging to the same a-tuple. So in general $|A| = a^n = aM$, with $M = a^{n-1}$ and $1 \leq n < \infty$. Hence, the general expression for the total number of valid sequences $T(l,M)$, using a set of $M$ a-tuples is:

$$T(l,M) = \sum_{i=1}^{M} a^i \binom{M}{i} i^l(1-a)^{M-i}. \tag{3.47}$$

Now the same experiment as described in the previous two sections is carried out. Again we start with an alphabet $A$ with $|A| = a > 1$ characters. If we draw a character which has a tuple companion already present in the sequence, we start all over again, but now with an alphabet of $a^2$ characters, then $a^3$ characters, etc. until a sequence has been drawn containing $l$ characters from distinct a-tuples. In this case the question also is what the expected exponent of $a$ is we end with.

The probability $p_n$ of having a valid string of length $l$ with characters from some alphabet $A$, with $|A| = aM = a^n$ is:

$$p_n = \frac{T(l,M)}{(aM)^l} = \frac{1}{a^{nl}} \sum_{i=1}^{a^{n-1}} a^i \binom{a^{n-1}}{i} i^l(1-a)^{a^{n-1}-i}. \tag{3.48}$$

Note that (3.47) may be written as:

$$T(l,M) = \left( x \frac{d}{dx} \right)^l [(x + 1 - a)^M - (1 - a)^M] \bigg|_{x=a}. \tag{3.49}$$

Using equation (3.49) the coefficients in the expansion of $p_n$ to $A = aM$ can be determined. The result is:

$$p_n = 1 - (a - 1) \frac{l(l-1)}{2A} + (a - 1) \frac{l(l-1)(l-2)}{24A^2} \left[ 3(a - 1)l - a + 5 \right] - \ldots. \tag{3.50}$$
Comparing (3.50) with equation (3.40) of the previous section, we find that the coefficients of $A^{-1}$ differ only by a factor $(a - 1)$ for large $l$, hence it is easy to see that the expected exponent has an upperbound analogous to that of Section 3.4.2.2, i.e.:

$$E(n) \leq 2 \log_a l + \log_a (a - 1) - \log_a 2 + \frac{a}{a - 1} + 2.$$ 

It is interesting to note that, although the starting-point for this model is quite different from that of Section 3.4.2.2, both result in similar upperbounds for the expected exponent of the alphabet cardinality. In particular for $a = 2$ both upperbounds are identical.

The model described in this section has a number of interesting mathematical properties, two of which are:

**I.** The probability $p_n$ can be written as follows:

$$p_n = a^{M-l} + a^{M-l-1}M \left(1 - \frac{1}{M}\right)^l (1 - a) + a^{M-l-2} \binom{M}{2} \left(1 - \frac{2}{M}\right)^l (1 - a)^2 + \cdots.$$ 

If we choose $n = 1 + \log_a l$, so that $M = l$, we have:

$$p_n = 1 + \frac{1}{a} \binom{l}{1} \left(1 - \frac{1}{l}\right)^l (1 - a) + \frac{1}{a^2} \binom{l}{2} \left(1 - \frac{2}{l}\right)^l (1 - a)^2 + \cdots.$$ 

For $l \to \infty$ this can be approximated by:

$$p_n \leq 1 + \binom{l}{1} \left(1 - \frac{1}{a}\right) e^{-1} + \binom{l}{2} \left(1 - \frac{1}{a}\right)^2 e^{-2} + \cdots$$

$$= \sum_{j=0}^{l-1} \binom{l}{j} \left(\frac{1 - a}{ae}\right)^j$$

$$= \left(1 + \frac{1 - a}{ae}\right)^l - \left(\frac{1 - a}{ae}\right)^l.$$ 

It can be seen that this goes to zero and the same situation arises as with equation (3.37) of the previous section.

**II.** Recursion is possible on $T(l, M)$, which is very convenient for numerical evaluation purposes.
The binomial coefficient in (3.47) is split up as follows:
\[
\binom{M}{i} = \left(\frac{M+1}{i} - 1\right) \binom{M}{i-1},
\]
yielding:
\[
T(l, M) = (M + 1) \sum_{i=1}^{M} a^i \binom{M}{i-1} i^{l-1}(1-a)^{M-i}
- \sum_{i=1}^{M} a^i \binom{M}{i-1} i^l(1-a)^{M-i}.
\]

Now define:
\[
U(l, M) := \sum_{i=1}^{M} a^i \binom{M}{i-1} i^l(1-a)^{M-i},
\]
then we have:
\[
T(l, M) = (M + 1)U(l - 1, M) - U(l, M). \tag{3.51}
\]

Next, consider \(T(l, M + 1)\):
\[
T(l, M + 1) = \sum_{i=1}^{M} a^i \binom{M+1}{i} i^{l}(1-a)^{M+1-i} + a^{M+1}(M + 1)^l
= \sum_{i=1}^{M} a^i \binom{M}{i} i^{l}(1-a)^{M+1-i}
+ \sum_{i=1}^{M} a^i \binom{M}{i-1} i^{l}(1-a)^{M+1-i} + a^{M+1}(M + 1)^l
= (1-a) \{T(l, M) + U(l, M)\} + a^{M+1}(M + 1)^l. \tag{3.52}
\]

Combining (3.51) and (3.52) gives us:
\[
T(l, M + 1) = (1-a)(M + 1)U(l - 1, M) + a^{M+1}(M + 1)^l. \tag{3.53}
\]

Next, consider \(U(l, M)\):
\[
U(l, M) = \sum_{i=1}^{M} a^i \binom{M}{i-1} i^l(1-a)^{M-i}
= \sum_{j=1}^{M} a^{j+1} \binom{M}{j} (j + 1)^l(1-a)^{M-j-1}
\]
\[ + a(1 - a)^{M-1} - a^{M+1}(M + 1)^l(1 - a)^{-1} \]
\[ = \frac{a}{1 - a} \sum_{j=1}^{M} a^j \binom{M}{j} \left( \sum_{k=0}^{l} \binom{l}{k} j^k \right) (1 - a)^{M-j} \]
\[ + \frac{a}{1 - a} \left\{ (1 - a)^M - a^M(M + 1)^l \right\} \]
\[ = \frac{a}{1 - a} \left\{ \sum_{k=0}^{l} \binom{l}{k} T(k, M) + (1 - a)^M - a^M(M + 1)^l \right\}. \] (3.54)

Combining (3.53) and (3.54) finally gives us the desired result:

\[ T(l, M + 1) = a(M + 1) \sum_{k=0}^{l-1} \binom{l-1}{k} T(k, M) + a(1 - a)^M(M + 1). \]

### 3.4.2.5 Conclusions on Heuristic Statistical Models

The conclusion to be drawn from the previously described heuristic statistical models is that we have to expect \( \sim 2 \log_a l \) as complexity of random sequences of length \( l \), \( l \) very large, over an \( a \)-ary alphabet. This is in itself not surprising, considering the square-root law of the birthday statistics.

Literature investigations have revealed a number of recent publications on pattern matching, arising from molecular biology. Here, one describes the observations that seemingly unrelated biological organisms possess long contiguous subsequences of DNA molecules which are practically identical. The key paper is [Arra 86], but also [Arra 85a] and [Arra 85b] are instructive. The results of [Arra 86] are summarized as follows:

Let \( X_1, \ldots, X_m \) and \( Y_1, \ldots, Y_n \) be two sequences of \( m \), resp. \( n \) independent identically distributed random variables and let \( p = Pr[X_i = Y_i] \), the probability that the outcomes of the \( i \)th \( X \) and \( Y \) are equal. The longest contiguous run of matches, \( R \), between the \( X \)'s and \( Y \)'s has an approximated expectation and variance of resp.:

\[ \mathcal{E}(R) = \log_p(qmn) + \gamma \log_p e - 1/2, \]
\[ VAR(R) = \left( \pi \log_p e \right)^2 /6 + 1/12, \]

where \( q = 1 - p \) and \( \gamma \) denotes Euler's constant.
Translated to one and the same random sequence of length \( l \), we arrive at \( 2 \log l \) approximately, as for \( l \to \infty \) the matching subsequences will not overlap with a probability going to 1.

### 3.4.3 Experimental Results

In order to verify the theoretic results of this chapter, a statistical experiment was set up. For this purpose a binary random number generator was used to generate random sequences for two experiments, in order to verify the progress of:

1. the expected complexity versus the sequence length,
2. the expected complexity versus the alphabet cardinality.

For the second experiment 2 or more bits were grouped together and converted to (hexadecimal) integers. If the value of the integer exceeded the maximum value \( A - 1 \), where \( A = |\mathcal{A}| \) denotes the alphabet cardinality, the group of bits was discarded and a new one was drawn.

#### 3.4.3.1 Complexity versus Sequence Length

For several sequence lengths a large number of binary sequences were generated independently and at random. The results are given in Table 3.10.

As can be seen from this table the results are in perfect agreement with the theory. For sequence length 24 the experiment yields estimates of 7.30 and 2.13 for expectation and variance respectively, whereas the exact values, as given by Table 3.1 are 7.418 and 2.188 respectively.
The Shortest FSR Equivalent

<table>
<thead>
<tr>
<th></th>
<th>( \hat{\mu} )</th>
<th>( \hat{\sigma} )</th>
<th></th>
<th>( \hat{\mu} )</th>
<th>( \hat{\sigma} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>12.21</td>
<td>1.70</td>
<td>9</td>
<td>4.54</td>
<td>0.65</td>
</tr>
<tr>
<td>3</td>
<td>8.27</td>
<td>1.16</td>
<td>10</td>
<td>4.35</td>
<td>0.59</td>
</tr>
<tr>
<td>4</td>
<td>6.76</td>
<td>0.93</td>
<td>11</td>
<td>4.22</td>
<td>0.58</td>
</tr>
<tr>
<td>5</td>
<td>5.90</td>
<td>0.84</td>
<td>12</td>
<td>4.10</td>
<td>0.55</td>
</tr>
<tr>
<td>6</td>
<td>5.40</td>
<td>0.75</td>
<td>13</td>
<td>4.01</td>
<td>0.58</td>
</tr>
<tr>
<td>7</td>
<td>5.01</td>
<td>0.67</td>
<td>14</td>
<td>3.92</td>
<td>0.55</td>
</tr>
<tr>
<td>8</td>
<td>4.77</td>
<td>0.70</td>
<td>15</td>
<td>3.80</td>
<td>0.59</td>
</tr>
</tbody>
</table>

Table 3.11: Estimates of expectation and deviation of complexity for various alphabet cardinalities.

![Figure 3.8: Estimates of expectation and deviation of complexity versus alphabet cardinality.](image)

The results also agree very well with the predicted values, given by the statistical models of Section 3.4.2. In particular the exact values as listed in Table 3.9 agree within \( \hat{\sigma}/2 \).

### 3.4.3.2 Complexity versus Alphabet Cardinality

For alphabet cardinalities \( A \) of 2 until 16, one thousand sequences of length 120 were drawn independently and at random. The results are listed in Table 3.11 and depicted in Figure 3.8.

Using the least squares method, the following estimate for the expected complexity as a function of alphabet cardinality is obtained:

\[
\hat{\mu} = e_1/ \ln A + e_2, \tag{3.55}
\]
\[ \hat{e}_1 = 7.8878 \quad \text{and} \quad \Delta e_1 = 0.065, \]
\[ \hat{e}_2 = 0.9525 \quad \text{and} \quad \Delta e_2 = 0.041. \]

Comparing (3.56) with the upperbound (3.35), results in a remarkably good estimate of 7.888 against a theoretical model value of 8.189. The results of this experiment therefore strongly support the assumption that the relation between expected complexity and alphabet cardinality indeed behaves like \(1/\ln A\).

### 3.5 Conclusions

This chapter highlighted the problem of finding the shortest feedback shift register which generates a given sequence with characters from some finite alphabet. We focussed here on the absolutely shortest FSR, regardless of its feedback function, which could be highly nonlinear. Moreover, the character alphabet need not necessarily be a finite field. To this end, a new complexity measure was introduced, which was called the maximum order complexity, as opposed to the first order, or linear complexity, the second order, or quadratic complexity, etc. The basic properties of maximum order complexity were shown and, in fact, it was demonstrated that the maximum order complexity is strongly connected with nonlinear feedback shift registers. The results on transposed and reciprocal sequences, sequences considered over power alphabets, as well as on the feedback functions of the maximum order FSR equivalent and their properties provide a new contribution to the theory of nonlinear feedback shift registers and a better understanding of their functioning.

The practical import of maximum order complexity was enhanced by the identification of an efficient algorithm for obtaining the maximum order complexity profile of arbitrary sequences. It was shown that the maximum order complexity profile can be determined in linear time and memory, using the Directed Acyclic Word Graph, which is a new application of Blumer's algorithm.

By considering the complexity of random sequences, a deep analysis of the typical behaviour of the maximum order complexity profile was obtained. The strong relation between counting incomplete paths in a De-Bruijn graph and counting the number of sequences of given length and maximum order complexity was proven. Significant bounds on these numbers were derived.
The expected maximum order complexity profile was considered from a statistical point of view, using several interesting heuristic models, and confirmed by experimental results. These results support the correctness of the models, stating that the expected maximum order complexity profile grows with twice the logarithm of the sequence length, where the alphabet cardinality is taken as the base of the logarithm. The consequence for the analysis of pseudo-random sequences and the resynthesis with feedback shift registers is that the total effort to determine the shortest FSR equivalent is quite high. If this total effort to be expected is denoted by $E_i$ and $\bar{c}$ denotes the expected complexity, we have for the binary case:

$$E_i = \bar{c}2^\bar{c}, \quad \bar{c} = 2 \log_2 l, \quad E_i = 2l^2 \log_2 l, \text{ including ANFT (Ch. 4).}$$

This makes the feasibility of general FSR resynthesis limited to moderate length sequences. If for example $\bar{c} = \log_2 l$, the total effort $E_i$ drops to a value of $l \log_2 l$, which is of course much more practical.

Concluding, we can say that the maximum order complexity profile is a useful new tool for judging the randomness of sequences. For example, it declares DeBruijn sequences as non-random, whereas these sequences are considered highly complex according to Lempel and Ziv and some of these sequences are also considered complex according to Rueppel's linear complexity profile.
Chapter 4

The Algebraic Normal Form of Arbitrary Functions over Finite Fields

Many functions over finite fields used in areas such as switching theory, digital optical computing, cryptography, etc. are mappings from $GF(q)^n$ onto $GF(q)$, i.e. functions of $n$ variables, each taking on values from the finite field of $q$ elements. One of the properties one may be interested in is their Algebraic Normal Form, i.e. their sum of products form e.g. as a measure of complexity (or nonlinearity). In this chapter we discuss a transform, called ANF Transform, which is a fast transform like the Fast Hadamard Transform, Fast Fourier Transform, etc. This ANFT is based on a matrix structure, which is shown to hold for every finite field. Using the ANFT we derive exact expressions for the probability distribution of the complexity of random functions over $GF(2)$.

4.1 Introduction

In this chapter we consider functions $f(x_1, x_2, \ldots, x_n)$, where $x_1, x_2, \ldots, x_n$ all are variables taking on values from $GF(q)$ and $f$ is a mapping of $GF(q)^n$ onto $GF(q)$. Such a function may be represented by some formula, expression or a table of argument values with corresponding function values. In the field of two elements the latter representation is often referred to as a truth table.

The goal is to devise an efficient tool for obtaining the algebraic normal form of these functions. The algebraic normal form is useful to judge the
complexity of a function, e.g. the number of product terms or the maximum nonlinear order (the largest product term). The ANF is used in areas such as switching theory, cryptography [Sieg 86, Ruep 84] and digital optical computing [Gayl 86]. The ANF over \( GF(q) \) is defined as:

\[
f(x_1, x_2, \ldots, x_n) = a_0 + a_1 x_1 + a_2 x_2 + \cdots + a_n x_n \\
+ a_{11} x_1^2 + a_{12} x_1 x_2 + \cdots + a_{nn} x_n^2 \\
+ a_{111} x_1^3 + a_{112} x_1^2 x_2 + a_{123} x_1 x_2 x_3 + \cdots + a_{nnn} x_n^3 \\
+ \cdots \\
+ a_{1\ldots 12\ldots 2\ldots \ldots n\ldots n} x_1^{q-1} x_2^{q-1} \cdots x_n^{q-1}.
\]

The order of a term in the ANF is defined as the sum of the exponents of the variables in a term. As can be seen, the highest order term has order \( n(q - 1) \) and there are \( q^n \) different terms corresponding to all the different products of all powers of all variables. The function \( f \) is therefore completely determined by the \( q^n \) coefficients \( a_{\ldots \ldots} \), which take on values from \( GF(q) \). Usually, these coefficients are specified so that the function \( f \) may be evaluated or the logical circuitry determined. The problem that will be treated in the sequel is that of determining the ANF coefficients, given the function values. This problem has been studied previously in [Benj 76] in the context of generalised boolean differences. The solution presented in this chapter focuses on the algebraic structure of a matrix transform, which is applied to determine the complexity of random functions over finite fields. In Section 4.2 a method for determining the ANF coefficients, called ANF transform, is given for the binary case, which is implemented as a fast transform in Section 4.3. Section 4.4 discusses an application of the theory of Section 4.2, viz. the complexity of random binary functions. Finally in Section 4.5 the method for the binary case is extended to \( GF(q) \).

### 4.2 The ANF Transform over \( GF(2) \)

The \( q^n \) product terms \( x_1^{e_1} x_2^{e_2} \cdots x_n^{e_n} \), with \( e_i \in \{0, 1, 2, \ldots, q - 1\} \) for \( i = 1, 2, \ldots, n \), form a basis for all functions of \( n \) variables over \( GF(q) \), as there are exactly \( q^n \) such functions and all \( q^n \) product terms are linearly independent. For the case \( q = 2 \) these products, written as \( 2^n \)-dimensional vectors over \( GF(2) \), form the well-known basis for the binary Reed-Muller codes of length \( 2^n \) and order \( n \), see [MacW 78]. These basis vectors are usually written down in ascending product order: \( 1, x_1, x_2, \ldots, x_n, x_1 x_2, \ldots, x_1 x_2 \cdots x_n \).
or in matrix form where the basis vectors are used as the rows of the matrix. For example, for \( n = 4 \) we have:

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

By this matrix any 16 bit ANF coefficient vector will be transformed into a unique 16 bit vector representing all function values.

There is, however, a more natural ordering of the products in the ANF, as given by the following definition:

**Definition 4.1** The product terms in the Algebraic Normal Form of a binary function of the \( n \) variables \( x_1, x_2, \ldots, x_n \) are defined to have a natural ordering iff the product terms are ordered as follows:

\[
1, x_1, x_2, x_1x_2, x_3, x_1x_3, x_2x_3, x_1x_2x_3, \ldots, x_1x_2x_3 \cdots x_n.
\]

This is a recursive ordering where the series of products is obtained by copying the already existing products, multiplied by the next variable. This ordering gives rise to a special matrix structure, which can also be obtained recursively.

**Proposition 4.1** Let the products in the Algebraic Normal Form have a natural ordering as defined above, then the linear transform which transforms a binary coefficient vector into a binary vector of function values is given by the binary matrix \( A_n \), which satisfies the recursion:

\[
A_0 = [1],
\]

\[
A_j = \begin{bmatrix} A_{j-1} & A_{j-1} \\ O_{j-1} & A_{j-1} \end{bmatrix}, \quad j = 1, 2, \ldots,
\]

where \( O_j \) denotes the \( 2^j \times 2^j \) all-zero matrix.

**Proof.** The proof follows immediately from the natural ordering of the product terms and the fact that the row vectors of the transform matrix are obtained by evaluating the corresponding product terms for all combinations of values of the \( n \) variables.
Our interest, however, lies in the inverse transform, i.e. the way to obtain the ANF coefficient vector when the vector representing the function values is given.

**Proposition 4.2** The inverse of the transform matrix $A_n$, as defined in Proposition 4.1, is $A_n$ itself.

**Proof.** Obviously, the inverse matrix exists, as all row vectors form a basis of $GF(2)^{2^n}$. So it is necessary and sufficient to prove that $A^2_n = I_n$, with $I_n$ the $2^n \times 2^n$ identity matrix. This can be proven by induction:

1) $A_0 = [1] = I_0$.

2) Suppose $A^2_{n-1} = I_{n-1}$, then

$$
A^2_n = \begin{bmatrix}
A^2_{n-1} & A^2_{n-1} + A^2_{n-1} \\
O_{n-1} & A^2_{n-1}
\end{bmatrix} = \begin{bmatrix}
I_{n-1} & O_{n-1} \\
O_{n-1} & I_{n-1}
\end{bmatrix} = I_n.
$$

From Proposition 4.2 we see that it is very easy to obtain the ANF coefficients, given the truth table of the function, using the transformation defined in Proposition 4.1. In the sequel this transform will be called the ANF transform or ANFT. Note that the position of the coefficients is now based on the natural ordering of the products. E.g. the coefficients of the first order ‘products’ (the linear terms) occur at positions $2^i$, $i = 0, 1, \ldots, n - 1$, the first position being position 0.

From the structure of the ANFT matrix one can already deduce interesting properties of a function $f$ of $n$ binary variables. E.g. if the truth table of $f$ has even weight, i.e. an even number of non-zero values, then its maximum nonlinear order is $n - 1$, else it has exactly nonlinear order $n$.

The recursion satisfied by the matrix $A_n$ as shown in Proposition 4.1 can also be written as

$$
A_0 = [1],
A_n = \begin{bmatrix}
1 & 1 \\
0 & 1
\end{bmatrix} \otimes A_{n-1},
$$

(4.1)
Table 4.1: Group table of all binary non-singular $2 \times 2$ matrices.

<table>
<thead>
<tr>
<th>$M_1 \backslash M_2$</th>
<th>$I$</th>
<th>$A$</th>
<th>$C$</th>
<th>$V$</th>
<th>$H$</th>
<th>$G$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$I$</td>
<td>$I$</td>
<td>$A$</td>
<td>$C$</td>
<td>$V$</td>
<td>$H$</td>
<td>$G$</td>
</tr>
<tr>
<td>$A$</td>
<td>$A$</td>
<td>$I$</td>
<td>$G$</td>
<td>$H$</td>
<td>$V$</td>
<td>$C$</td>
</tr>
<tr>
<td>$C$</td>
<td>$C$</td>
<td>$H$</td>
<td>$I$</td>
<td>$G$</td>
<td>$A$</td>
<td>$V$</td>
</tr>
<tr>
<td>$V$</td>
<td>$V$</td>
<td>$G$</td>
<td>$H$</td>
<td>$I$</td>
<td>$C$</td>
<td>$A$</td>
</tr>
<tr>
<td>$H$</td>
<td>$H$</td>
<td>$C$</td>
<td>$V$</td>
<td>$A$</td>
<td>$G$</td>
<td>$I$</td>
</tr>
<tr>
<td>$G$</td>
<td>$G$</td>
<td>$V$</td>
<td>$A$</td>
<td>$C$</td>
<td>$I$</td>
<td>$H$</td>
</tr>
</tbody>
</table>

where $\otimes$ denotes the usual Kronecker product of matrices. From equation (4.1) it can be seen that the $2 \times 2$ binary matrix $\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$ plays a fundamental role in the ANFT over $GF(2)$.

There are only 6 non-singular $2 \times 2$ matrices over $GF(2)$; a number which can easily be counted. So it looks interesting to determine the relationship of these matrices. The 6 matrices are:

$$
\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = I, \quad \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} = A, \quad \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} = C
$$

$$
\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = V, \quad \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} = H, \quad \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} = G
$$

These 6 matrices form a group under normal matrix multiplication as shown by Table 4.1. From this table it can be seen that besides the identity matrix $I$, the matrix $V$ which exchanges coordinates and the ANFT matrix $A$, only matrix $C$ which is the transpose of $A$, is self-inverse.

### 4.3 Fast Transform Implementation

The ANFT can be implemented very efficiently due to the structure of the transform matrix. This is illustrated as follows.

Let $\phi_n = (\phi_0, \phi_1, \ldots, \phi_{2^n-1})$ denote the vector of function values, i.e. the truth table in vector notation. Also let any vector $v$ of function values be written as $v = v^1 \| v^2$, i.e. the concatenation of the first half and the second half of $v$. Then it can easily be seen that an ANFT of order $n$ reduces to two ANFT's of order $n - 1$ and a vector modulo 2 addition:

$$
\phi_n A_n = \phi_n^1 A_{n-1} \| (\phi_n^1 + \phi_n^2) A_{n-1}.
$$
This reduction is illustrated by Figure 4.1. This reduction implies that an ANFT of order \( n \) reduces to order 0 in \( n \) steps, thereby having to carry out modulo 2 additions only. It can also be seen that the total number of modulo 2 additions equals \( \frac{m}{2} \log_2 m \), where \( m = 2^n \) is the length of the vector \( \phi \).

In Figure 4.2 the ANFT for \( n = 3 \) is shown in recursive form and Figure 4.3 illustrates the binary 'butterfly' form which is an analogon to the Fast Fourier Transform [Rabi 75], including the permuted output order. Note that due to its simplicity this algorithm can be implemented in software or hardware very efficiently.

### 4.4 Random Binary Functions

In this section the results of the second section are applied to random binary functions. Such functions assume the values 0 and 1 with probability one half, for every argument value, i.e. every bit in their truth table is chosen independently at random. It turns out to be easy to establish the statistics of the ANF coefficients.

**Lemma 4.3** For random binary functions the ANF coefficients are independent, identically distributed (i.i.d.) binary random variables (r.v.) with probabilities \( Pr[0] = Pr[1] = \frac{1}{2} \).

**Proof.** The transform matrix \( A_n \) is invertible. \( \square \)
Figure 4.2: Binary ANFT for $n = 3$ in recursive form.

Figure 4.3: Butterfly form of binary ANFT.
It now is relatively easy to say something about how complex a random binary function is on the average, if it is clear what is meant by the word complexity. It is clear that complexity of functions is not only related to the number of product terms in the ANF (which is binomially distributed due to the previous lemma), but also to the order of the product terms. To this end, a complexity definition will be given, but first the statistics of the (nonlinear) order of a function, which is defined to be equal to the order of highest order product term in the ANF, is determined in particular.

**Proposition 4.4** For random binary functions of \( n \) variables the expectation and variance of the nonlinear order \( O \) are given by:

\[
\mathcal{E}(O) = n - \sum_{k=0}^{n-1} 2^{-S_n(k)},
\]

\[
\text{VAR}(O) = \sum_{k=0}^{n-1} (2k + 1)2^{-S_n(k)} - \left( \sum_{k=0}^{n-1} 2^{-S_n(k)} \right)^2,
\]

with \( S_n(k) = \sum_{i=0}^{k} \binom{n}{i} \).

**Proof.** There are \( 2^n \) coefficients in the ANF, each having probability one half to occur. There are exactly \( \binom{n}{i} \) coefficients for all \( i \)th order products. Using the fact that the order of a function is determined by its highest order product, the probability distribution of the function order is readily determined:

\[
Pr[O = n - j] = \left\{ \prod_{k=0}^{j-1} 2 \binom{n}{k} \right\} \binom{n}{j} \left( 1 - 2 \binom{n}{j} \right)
\]

\[
= 2^{-S_n(j-1)} - 2^{-S_n(j)}, \quad 0 \leq j < n,
\]

\[
Pr[O = 0] = 2^{-(2^n-1)}.
\]

By straightforward calculation the expectation and variance of the order \( O \) can be determined. \( \square \)

The expectation and variance can very well be approximated by:

\[
\mathcal{E}(O) \approx n - \frac{1}{2} - 2^{-(n+1)},
\]

\[
\text{VAR}(O) \approx \frac{1}{4} + 2^{-n}.
\]
These approximations show that a random binary function almost always has nonlinear order close to the maximum value.

As was already noticed neither the number of products in the ANF, nor the nonlinear function order alone are sufficient to describe the degree of difficulty or complexity of a function. It is clear that the number of products as well as their individual orders determine this complexity. Therefore we propose a weighted real sum of all coefficients as a complexity measure.

**Definition 4.2** The complexity of a binary function $f$ of $n$ variables is defined as the weighted real sum of all the $2^n$ coefficients in the Algebraic Normal Form of $f$. The weights assigned to each coefficient are chosen in such a way that they express the difficulty of building a function in hardware or software.

This complexity measure may be viewed as a mapping $c$ which maps all binary vectors of length $2^n$ e.g. to the positive integers, i.e. $c : GF(2)^{2^n} \rightarrow \mathbb{N}$.

The mapping may consist of taking the inproduct of the coefficient vector and a weight vector, which expresses the difficulty of multiplications and additions.

Let $\gamma := c(a) = a^T \omega$, where $a$ denotes the ANF coefficient vector and $\omega$ the weight vector. Let us assume $\omega$ to be $(1,2,2,3,2,3,4,2,3,3,4,\ldots)$, which expresses the fact that addition is slightly more difficult than multiplication, as is the case with digital electronic circuitry. Then, by straightforward calculation, we find for the expected complexity $\bar{\gamma}$ of a random binary function:

$$\bar{\gamma} = \left(\frac{n}{2} + 1\right) 2^{n-1}.$$  

This is exactly one half of the maximum complexity that can be achieved with the aforementioned weight vector. It is clear that many other weight vectors might be used that express in some way the particular difficulty of building a function in hardware or software.

### 4.5 The ANF Transform over $GF(q)$

In the first section the algebraic normal form of functions over $GF(q)$ was introduced. As could be seen there are $q^n$ product terms and the maximum power of a variable that occurs is $q - 1$, as $x^q = x$ in the finite field $GF(q)$. In the second section a natural ordering of the products was introduced which gave rise to a special (recursive) transform matrix structure. This
matrix structure then led us to the inverse matrix. In this section we will follow the same approach.

The natural ordering of products of variables in the ANF over $GF(q)$ is defined as follows:

**Definition 4.3** The product terms in the Algebraic Normal Form of a function of the $n$ variables $x_1, x_2, \ldots, x_n$ over $GF(q)$ are defined to have a natural ordering iff the product terms are ordered as follows:

$$1, x_1, x_2^2, \ldots, x_1^{q-1}, x_2, x_1 x_2, x_2^2 x_2, \ldots, x_1^{q-1} x_2^{q-1}, \ldots, x_1^{q-1} x_2^{q-1} x_3^{q-1} \ldots x_n^{q-1}.$$  

This ordering again gives rise to a special matrix structure, which can also be obtained recursively.

**Proposition 4.5** Let the products in the Algebraic Normal Form have a natural ordering as shown above and let $\alpha$ be a primitive element of $GF(q)$. Then the linear transform which transforms a $GF(q)$ coefficient vector into a vector of function values is given by the matrix $A_n$ over $GF(q)$, which satisfies the recursion:

$$A_0 = [1],$$

$$A_j = \begin{bmatrix}
1 & 1 & 1 & 1 & \cdots & 1 \\
0 & 1 & \alpha & \alpha^2 & \cdots & \alpha^{q-2} \\
0 & 1 & \alpha^2 & \alpha^4 & \cdots & \alpha^{2(q-2)} \\
\vdots \\
0 & 1 & \alpha^{q-2} & \alpha^{2(q-2)} & \cdots & \alpha^{(q-2)^2} \\
0 & 1 & 1 & 1 & \cdots & 1
\end{bmatrix} \otimes A_{j-1}.$$  

**Proof.** The proof is the same as for Proposition 4.1. The fact that $\alpha$ is a primitive element of $GF(q)$ permits us to write all the non-zero elements of $GF(q)$ as a power of $\alpha$. 

The interest again lies in the inverse transform in order to obtain the ANF coefficient vector when the vector representing the function values is given.

**Proposition 4.6** The inverse of the transform matrix $A_n$ defined in Proposition 4.5 is the matrix $A_n^{-1}$ which satisfies the recursion:

$$A_0^{-1} = [1],$$
If we write $B_n$ for $A_n^{-1}$, the matrix recursions of Propositions 4.5 and 4.6 in fact read:

$$A_n = A_1 \otimes A_{n-1} \quad \text{and} \quad B_n = B_1 \otimes B_{n-1}, \quad n > 1,$$

where $A_1$ and $B_1$ are the lefthand matrices in the Kronecker products of both propositions. The proof of Proposition 4.6 is accomplished by making use of elementary properties of finite fields. But first we introduce some convenient notation.

Let $A_1$ and $B_1$ be written as follows:

$$A_1 = \begin{bmatrix} \varepsilon_0^T & M \\ 0 & 1 \end{bmatrix}, \quad B_1 = \begin{bmatrix} 1 \\ 0^T \end{bmatrix},$$

Here, $M$ is the $(q - 1) \times (q - 1)$ symmetric matrix of all powers of all non-zero elements of $GF(q)$, $M^{rot}$ is the right rotated version of $M$ (i.e. all row vectors are reversed because of symmetry), and:

$$\varepsilon_0^T = (1, 0, 0, \ldots, 0)^T, \quad \varepsilon_{q-2} = (0, 0, \ldots, 0, 1),$$

$$0^T = (0, 0, \ldots, 0, 0)^T, \quad 1 = (1, 1, \ldots, 1, 1),$$

are $(q - 1)$ dimensional vectors. It then follows by straightforward calculation that:

$$A_1 B_1 = \begin{bmatrix} \varepsilon_0^T & N \\ 0 & m \end{bmatrix},$$

with $m = -1 M^{rot}$ and $N = -M M^{rot} - D^{(0,q-2)}$, where $D^{(0,q-2)}$ is a $(q - 1) \times (q - 1)$ matrix with all zeroes except for the entry in the first row and last column where it has a 1.

In order to evaluate the products $M M^{rot}$ and $1 M^{rot}$, we use the fundamental property of finite fields that $\beta^{q-1} = 1$ for every non-zero element $\beta$ from $GF(q)$ in the following lemma.

**Lemma 4.7** Let $M$ and $M^{rot}$ be as defined before, then the matrix $P := -M M^{rot}$ is given by its elements $p_{ij}$ to be:

$$p_{ij} = \begin{cases} 1; & j = i + 1, \ 0 \leq i < q - 2, \\
1; & i = 0, \ j = q - 2, \\
0; & \text{else}. \end{cases}$$
Proof. From the definition of the matrices $M$ and $M^\text{rot}$ it follows that:

$$-p_{ij} = \sum_{k=0}^{q-2} (\alpha^{q-2-j+i})^k = \sum_{k=0}^{q-2} \beta^k$$

$$= \begin{cases} 
0; & \beta \neq 1, \\
-1; & \beta = 1. 
\end{cases}$$

Corollary 4.8 Let $1$, $\varepsilon_{q-2}$ and $M^\text{rot}$ be as defined above, then the vector $m = -1M^\text{rot} = \varepsilon_{q-2}$.

Proof. As the first row of $M$ is equal to $m$ the result follows immediately from Lemma 4.7.

Note also that $D^{(0,q-2)}$ eliminates exactly the only non-zero element off the diagonal of the matrix $P$.

We are now ready to prove Proposition 4.6.

Proof of Proposition 4.6 We prove that $A_n B_n = I_n$, where $I_n$ is the $q^n \times q^n$ identity matrix. This can easily be achieved by induction:

i) $A_0 B_0 = [1] = I_0$.

ii) Suppose $A_{n-1} B_{n-1} = I_{n-1}$, then

$$A_n B_n = (A_1 \otimes A_{n-1})(B_1 \otimes B_{n-1})$$

$$= A_1 B_1 \otimes A_{n-1} B_{n-1}$$

$$= A_1 B_1 \otimes I_{n-1}.$$}

However, from Lemma 4.7 and Corollary 4.8 it follows that $A_1 B_1 = I_1$, so that $A_n B_n = I_n$.

For fields of characteristic 2 the matrix $B_1$ is particularly simple as all minus signs vanish; in particular for $GF(2)$ it reduces to the matrix $A_1$ of Section 4.2.

Example 4.1 As an example the $B_1$ matrices over $GF(3)$, $GF(4)$ and $GF(5)$ are given below.

$$GF(3) : \begin{bmatrix} 1 & 0 & 2 \\
0 & 2 & 2 \\
0 & 1 & 2 \end{bmatrix} \quad GF(5) : \begin{bmatrix} 1 & 0 & 0 & 4 \\
0 & 4 & 4 & 4 \\
0 & 2 & 1 & 3 \end{bmatrix}$$
ANF Transform over $GF(q)$

$GF(4) : \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & \alpha^2 & \alpha & 1 \\ 0 & \alpha & \alpha^2 & 1 \end{bmatrix}$ with $\alpha^2 + \alpha + 1 = 0$

Example 4.2 $GF(2)^2$ versus $GF(4)$.

Consider two binary variables $x_0$ and $x_1$ which form a vector $\underline{x}$ of length two over $GF(2)$. Suppose this vector $\underline{x}$ is transformed into the vector $\underline{y} = (y_0, y_1)$ by a simple wire crossing as shown in Figure 4.4. The complexity of this transformation, denoted with $2\gamma_2$, is equal to 8 if the weight vector $\omega_2$ is taken to be $(1, 2, 2, 4)$.

<table>
<thead>
<tr>
<th>$x_0$</th>
<th>$x_1$</th>
<th>$y_0$</th>
<th>$y_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Figure 4.4: Wire crossing as a transform in $GF(2)^2$.

Now consider the binary length two vectors as variables $X$ and $Y$ which take on values from $GF(4)$. Then $Y$ is a quadratic function of $X$, i.e. $Y = \alpha X^2$, with $\alpha^2 + \alpha + 1 = 0$. The circuit which implements this transform over $GF(4)$ now consists of two $GF(4)$ multipliers, as shown in Figure 4.5, which is clearly more complex than a wire crossing. The complexity of this transformation can be obtained in a similar way as for the binary case. However, a $GF(q)$ complexity measure should also take into account the
multiplication by a constant. Hence, the complexity, $\gamma_4$, is equal to 12 if the weight vector $w_4$ is taken to be $(2, 4, 6, 8)$ and the multiplication by $\alpha$ is taken into account.

### 4.6 Conclusions

In this chapter we have introduced a fast transform method for obtaining the Algebraic Normal Form of functions over finite fields. This transform is based on a matrix structure and shows resemblance with many other fast transforms like the Hadamard, Walsh and Fourier transforms. We have also discussed an application of this ANFT to random binary functions and their complexity. We feel that our approach provides insight in the theory of finite field transforms and that the ANFT as we have described it can successfully be applied in the areas mentioned in the introduction.
Chapter 5

Generating Binary DeBruijn Sequences

This chapter discusses an algorithm for the generation of binary DeBruijn sequences using feedback shift registers (FSR's).

The motivation for investigating this problem emerges from Chapter 3, where DeBruijn sequences occur at several places as having interesting properties. Also from Chapter 4 we have learned about the expected complexity of random functions and it therefore seems interesting to look at the complexity of the feedback function of the FSR that generates a DeBruijn sequence.

The algorithm presented here is based on the principle of cycle joining, through which all the cycles in the cycle structure of a FSR can be joined together to one complete cycle, thereby producing a DeBruijn sequence of period $2^n$, where $n$ is the length of the FSR.

By a proper choice of the feedback function of the FSR it turns out to be possible to generate $O(2^{2n}/\log^{2} n)$ DeBruijn sequences of period $2^n$, requiring only $3n$ bits of storage and at most $4n$ FSR shifts for the generation of the next bit in the sequence.

5.1 Introduction

A binary DeBruijn sequence is a periodic binary sequence with period $2^n$ in which every binary $n$-tuple occurs exactly once and therefore has maximum order complexity equal to $n$. It is well-known ([DeBr 46]) that the number of DeBruijn sequences with period $2^n$ is equal to $2^{2^n-1}-n$. A comprehensive survey of the construction problem of DeBruijn sequences can
be found in [Fred 82]. Many algorithms make use of a shift register producing short cycles which may be joined together to produce one DeBruijn sequence in several ways. The common approach is to apply the pure cycling register (PCR, [Fred 75]) or the pure summing register (PSR, [Etzi 84]) for producing the cycles. In Section 5.3 a universal algorithm is described for joining cycles in an arbitrary cycle structure, i.e. arising from a FSR with some arbitrary feedback function. This algorithm may be considered a generalization of Fredricksen's method.

Although the algorithm only needs 3n bits of storage, it is shown that the computational complexity to produce a sequence bit directly depends on the lengths of the various cycles produced by the FSR. The conclusion is that it is desirable to use feedback functions, which have a cycle structure consisting of many short cycles. By applying the theory of linear feedback shift registers very large sets of appropriate feedback functions can be obtained which satisfy this property. By this method, as proposed in Section 5.4, it is shown that under certain conditions approximately $\sqrt{\log 2n/\pi n} 2^{2n/\log 2n}$ DeBruijn sequences can be generated, for $n \to \infty$, requiring at most $4n$ FSR shifts for the generation of the next bit in the sequence.

### 5.2 Basic Concepts

For the description of a feedback shift register (FSR) we refer to Chapter 3, Figure 3.1. As we restrict ourselves to binary DeBruijn sequences, the FSR consists of $n$ binary memory cells and a feedback function $F$ which is a mapping of $GF(2)^n$ onto $GF(2)$.

A DeBruijn sequence of period $2^n$ is denoted by $s = (s_0, s_1, \ldots, s_{2^n-1})^\infty$, with all $s_i \in GF(2)$. As in Section 3.3 the state of the FSR is denoted by $S^i = (s_i, s_{i+1}, \ldots, s_{i+n})$, the $i$th state vector, which is an element of $GF(2)^n$. However, for ease of notation sometimes the superscript $i$ will be omitted. By clocking the FSR the state $S^i$ is succeeded by the next-state $S^{i+1} = (s_{i+1}, \ldots, s_{i+n})$, where $s_{i+n} = F(s_i, s_{i+1}, \ldots, s_{i+n-1})$. The feedback function $F$ determines the next-state operator $\mathcal{F} : GF(2)^n \to GF(2)^n$, under which $S^i \mathcal{F} = S^{i+1}$. Following [Etzi 84] the conjugate $\hat{S}$ and the companion $S'$ of state $S$ are defined by:

\[
\hat{S} = (s_i + 1, s_{i+1}, \ldots, s_{i+n-1}),
\]

\[
S' = (s_i, s_{i+1}, \ldots, s_{i+n-1} + 1).
\]
Due to the way the binary FSR operates, each state has two possible predecessors which are each others conjugates, and two companion states as possible successors. The transitions between these four states determine an adjacency quadruple belonging to the binary \((n - 1)\)-tuple they have in common. The superposition of all adjacency quadruples yields a DeBruijn graph \(G_n\) of degree \(n\), which is a directed graph with \(2^n\) nodes, labeled by the elements of \(GF(2)^n\). Figure 5.1 shows the DeBruijn graphs for \(n = 3\) and \(n = 4\).

Analogous to Definition 3.3 a cyclic sequence of states is defined as a cycle.

**Definition 5.1** A cycle \(C\) of length \(k\) in \(G_n\) is defined as the cyclic sequence of \(k\) distinct states, i.e. \(C = (S^i, S^{i+1}, \ldots, S^{i+k-1})\), such that \(S^i = S^{i+k-1}F\) and \(S^{i+j} = S^{i+j-1}F\), for \(j = 1, 2, \ldots, k - 1\).

The feedback function \(F\) is said to be non-singular if the operator \(F\) is one-to-one. If \(F\) is non-singular then each state has a unique predecessor and a unique successor. Consequently, the DeBruijn graph \(G_n\) decomposes under \(F\) into a set of branchless and disjoint cycles, which is called a factor of the graph. As was already mentioned in Section 3.1.3, we know from Golomb's book, [Golo 67], that a necessary and sufficient condition
for a feedback function $F$ to be non-singular is that $F$ can be written as $F(x_0, x_1, \ldots, x_{n-1}) = x_0 + G(x_1, \ldots, x_{n-1})$, where $G$ is a function of $n - 1$ variables over $GF(2)$. In the sequel our attention will be restricted to non-singular feedback functions $F$ only.

Finally a word about joining two cycles together to one. Two cycles $C_1$ and $C_2$ are said to be adjacent if they are state disjoint and there exists a state $S$ on $C_1$ and a state $S'$ on $C_2$, which are each other's companions. Cycle $C_1$ can be joined to cycle $C_2$ by interchanging the predecessors of $S$ and $S'$ in the corresponding adjacency quadruple, a process illustrated in Figure 5.2.

**5.3 A Universal Algorithm for Joining Cycles in the DeBruijn Graph**

Consider an arbitrary non-singular feedback function $F$ by which the DeBruijn graph decomposes into a set of $m$ distinct cycles $C_1, C_2, \ldots, C_m$. We will now show that it is always possible to find adjacency quadruples for joining these cycles together to one DeBruijn cycle, i.e. a complete cycle containing all $2^n$ different states.

**Proposition 5.1** Let $C$ be a cycle in a factor of the DeBruijn graph $G_n$. Then $C$ is a DeBruijn cycle if and only if the existence of state $S$ on $C$ also implies the existence of its companion $S'$ on $C$.

**Proof.** First assume that the cycle $C$ satisfies the property that if $S$ is on $C$ then also its companion $S'$ is on $C$. Without loss of generality assume that $C$ contains the all-zero state $(0, 0, \ldots, 0)$. Then this cycle also contains the companion state $(0, 0, \ldots, 1)$, which has two possible successors. Since these
successors are each other's companions, they both are on $C$. By repeating this argument all $2^n$ states appear to be on $C$, hence $C$ is a DeBruijn cycle.

Next assume state $S$ is on $C$, but its companion $S'$ is not on $C$. Then, by definition, $C$ is not a DeBruijn cycle. □

**Corollary 5.2** Let the DeBruijn graph $G_n$ be decomposed as a factor containing two or more cycles. Then each individual cycle is adjacent to some other cycle in the factor.

**Proof.** Since none of the cycles is a DeBruijn cycle, it directly follows from Proposition 5.1 that each cycle contains at least one state $S$ whose companion $S'$ is on some other cycle. □

The problem of joining cycles in a factor now is reduced to finding adjacency quadruples between adjacent cycles. In constructing an algorithm to solve this problem we have made use of the decimal representation of FSR states.

**Definition 5.2** The decimal representation $D_i$ of a FSR state $S^i$ of length $n$ is the integer number defined by:

$$D_i = \sum_{j=0}^{n-1} s_{i+j} 2^{n-j-1},$$

where the binary digits $s_{i+j}$ are treated as integers.

The numerically least state plays an important role in the sequel and therefore we use the following definition:

**Definition 5.3** The cycle representative $R(C)$ of a cycle $C$ is defined as the state which has the least decimal value of all states on that cycle, i.e.:

$$R(C) = \min_{S^i \in C} D^i.$$

Note that since there is no state with decimal value less than zero, the all-zero state always is the cycle representative of the cycle it lies on.

A cycle representative has the following important property.

**Proposition 5.3** If a factor contains two or more cycles then each non-zero cycle representative and its companion occur on different cycles.
Proof. Assume that \( R = (s_i, s_{i+1}, \ldots, s_{i+n-1}) \neq 0 \) is the cycle representative of a cycle \( C \) and that the companion of \( R, R' \) is on \( C \). Then, due to the non-singularity of the feedback function used, the state \((0, s_i, \ldots, s_{i+n-2})\) must occur on \( C \) as predecessor of \( R \) or \( R' \). This, however, is impossible since \((0, s_i, \ldots, s_{i+n-2})\) has a smaller decimal value than \( R \), contradicting the assumption that \( R \) is cycle representative. □

The following algorithm uses the fact that non-zero cycle representatives and their companions are positioned on different cycles, to join cycles in a very structured and simple way.

**Cycle Joining Algorithm**

Given is a FSR of length \( n \) with non-singular feedback function \( F \). From state \( S_i = (s_i, s_{i+1}, \ldots, s_{i+n-1}) \) of the FSR the successor state \( S_{i+1} = (s_{i+1}, s_{i+2}, \ldots, s_{i+n}) \) is obtained as follows:

If \( S_i = (s_i, 0, \ldots, 0) \)
then \( S_{i+1} = (0, 0, \ldots, s_i + 1) \)
else Begin
If either \((s_i+1, s_{i+2}, \ldots, 0)\) or \((s_i+1, s_{i+2}, \ldots, 1)\) is a cycle representative
then \( S_{i+1} = (s_{i+1}, s_{i+2}, \ldots, F(s_i, \ldots, s_{i+n-1}) + 1) \)
else \( S_{i+1} = (s_{i+1}, s_{i+2}, \ldots, F(s_i, \ldots, s_{i+n-1})) \)
End

It can be seen that the algorithm complements the value of the feedback function only if there is a cycle representative amongst the two possible successor states. The first step of the algorithm is only strictly necessary if the feedback function \( F \) gives rise to a next-state operator \( \mathcal{F} \) which does not map the all-zero state to itself.

As the algorithm is memoryless, the combination of the FSR and the cycle joining algorithm can be seen as a new FSR with a feedback function which differs from \( F \), as depicted in Figure 5.3. This new feedback function \( \tilde{F} \) is completely determined by the original function \( F \). To verify the algorithm, it is first shown that if \( F \) is non-singular then \( \tilde{F} \) is also non-singular.

**Proposition 5.4** Let \( F \) be an arbitrary non-singular feedback function. If the cycle joining algorithm is applied on the FSR with feedback function \( F \), then the feedback function \( \tilde{F} \) that corresponds to the mapping in-
duced by the algorithm is also non-singular, i.e. \( \tilde{F}(x_0, x_1, \ldots, x_{n-1}) = x_0 + \tilde{G}(x_1, \ldots, x_{n-1}) \).

**Proof.** The effect of the cycle joining algorithm is that it complements an entry in the truth table of \( F \) every time it has encountered a cycle representative or a state with \( n - 1 \) trailing zeroes. Both cases are treated separately:

i) Suppose \( S^i \) has successor \( S^{i+1} \) in the FSR, and either \( S^{i+1} \) or its companion \( \hat{S}^{i+1} \) are cycle representatives. Then the conjugate state \( \hat{S}^i \) has \( S^{i+1} \) as its successor in the FSR, whose companion is \( \hat{S}^{i+1} \). Therefore always two entries in the truth table are complemented, viz. for \( S^i \) and for \( \hat{S}^i \), leaving the truth table of \( F \) symmetric in its variable \( x_0 \) and hence the feedback function non-singular.

ii) If the original feedback function \( F \) maps state \((s_i, 0, \ldots, 0)\) to \((0, 0, \ldots, s_i + 1)\) then the algorithm does not contribute. If however \((s_i, 0, \ldots, 0)\) is mapped to \((0, 0, \ldots, s_i)\) by \( F \), the algorithm complements the entries addressed by \((0, 0, \ldots, 0)\) and \((1, 0, \ldots, 0)\) in the truth table of \( F \), thereby leaving \( F \) non-singular.

Now that it has been demonstrated that the algorithm decomposes the DeBruijn graph into a set of disjoint cycles, it remains to show that this factor consists of only one DeBruijn cycle of length \( 2^n \).
Proposition 5.5 Let $F$ be an arbitrary non-singular feedback function. If the cycle joining algorithm is applied on a FSR with feedback function $F$, then the obtained cycle has length $2^n$.

Proof. Assume that two or more cycles of length less than $2^n$ are obtained. At least one of these does not contain the all-zero state. This cycle contains the two successive states $(1, s_i, \ldots, s_{i+n-2})$ and $(s_i, s_{i+1}, \ldots, s_{i+n-1})$, where the latter is assumed to be the cycle representative. According to Proposition 5.3 the companion of this cycle representative is on another state and its predecessor is the conjugate of the predecessor of the cycle representative, i.e.: $(0, s_i, \ldots, s_{i+n-2}), (s_i, s_{i+1}, \ldots, s_{i+n-1} + 1)$. This, however, contradicts with the way the cycle joining algorithm works, because $(1, s_i, \ldots, s_{i+n-2})$ would be mapped to $(s_i, s_{i+1}, \ldots, s_{i+n-1} + 1)$. Therefore, all $2^n - 1$ non-zero states are contained in one cycle. As the algorithm also maps $(s_i, 0, \ldots, 0)$ to $(0, 0, \ldots, s_i + 1)$ the result must be one cycle of length $2^n$. □

It should be clear that the algorithm itself is capable of generating all possible DeBruijn sequences by choosing all different non-singular feedback functions $F$ of which there are $2^{2^n-1}$. So there will be many feedback functions which give rise to the same DeBruijn sequence as there are only $2^{2^n-1-n}$ of these sequences.

Example 5.1 Consider a FSR of length 4 and feedback function

$$F(x_0, x_1, x_2, x_3) = x_0 + x_1x_2 + x_2x_3 + x_1x_2x_3.$$ 

The cycle structure consists of the five cycles:

- $(0)$
- $(1, 2, 4, 8)$
- $(3, 7, 15, 14, 12, 9)$
- $(5, 10)$
- $(6, 13, 11)$,

where each cycle is led by its representative. By applying the cycle joining algorithm to this FSR the five five cycles are joined together to one cycle. In this way the full cycle $(0, 1, 3, 6, 13, 11, 7, 15, 14, 12, 9, 2, 5, 10, 4, 8)$ is obtained.

The following table illustrates the operation of the cycle joining algorithm.
5.4 Efficient Use of the Cycle Joining Algorithm

The cycle joining algorithm requires storage of \( n \) bits for state \( S^i \), \( n \) bits for cycling either \( S^{i+1} \) or \( S^{n+1} \) and \( n \) bits to store the current least value on a cycle, accounting for a total of \( 3n \) bits.

It is more complicated to determine the amount of time needed to generate the next state on a cycle. The most time consuming part of the algorithm is the search for the cycle representatives of the cycles to which \( S^{i+1} \) and \( S^{n+1} \) belong. The worst case situation arises if one of these states is a cycle representative, in which case the entire cycle is traversed. This means that the maximum time required is proportional to the lengths of the cycles in the decomposition of the DeBruijn graph \( G_n \) induced by the feedback function \( F \). Evidently, functions \( F \) are required which give rise to relatively short cycles. For arbitrary non-singular feedback functions \( F \) the

<table>
<thead>
<tr>
<th>( S^i )</th>
<th>( S^{i+1} )</th>
<th>( S^{n+1} )</th>
<th>cmpl.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>13</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>11</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>6</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>15</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>14</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>12</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>9</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
DeBruijn Sequences

lengths of the longest cycles can be expected to be very large, according to Golomb [Golo 67, pg. 127]. However, the theory on linear feedback shift registers offers a solution to the problem of constructing feedback functions which result in arbitrary short cycles.

If the feedback function $F$ is linear, i.e. $F(x_0, x_1, \ldots, x_{n-1}) = b_0 x_0 + b_1 x_1 + \ldots + b_{n-1} x_{n-1}$ with all $b_i \in GF(2)$, then a characteristic polynomial $f(x)$ of degree $n$ over $GF(2)$ is associated with $F$, defined by:

$$f(x) = b_0 + b_1 x + \ldots + b_{n-1} x^{n-1} + x^n.$$ 

The order (sometimes called 'exponent') of a polynomial $g(x)$, denoted $\text{ord } g(x)$, is defined as the least integer $e$ for which $g(x)$ divides $x^e - 1$. If $C$ is a cycle associated with a characteristic polynomial $g(x)$, then the length of this cycle is a proper divisor of $\text{ord } g(x)$. The following results can be found in Golomb's book [Golo 67, pp. 37-42]:

- If $g(x)$ is an irreducible polynomial of degree $m$ over $GF(2)$, then $\text{ord } g(x)$ divides $2^m - 1$.

- Let $g_1(x), \ldots, g_k(x)$ be $k$ pairwise relatively prime, non-zero polynomials and let $f(x) = g_1(x) \cdots g_k(x)$. Then $\text{ord } f(x)$ is equal to the least common multiple of $\text{ord } g_1(x), \ldots, \text{ord } g_k(x)$.

Now consider a polynomial $f(x)$ which can be factored into $r$ irreducible polynomials of the same degree $m$. It then follows that the cycles induced by the feedback function associated with $f(x)$ all have lengths dividing $2^m - 1$.

The number of irreducible polynomials of degree $m$ over $GF(2)$ is given by:

$$\Psi_2(m) = \frac{1}{m} \sum_{d|m} 2^d \mu\left(\frac{m}{d}\right),$$

in which $\mu(\cdot)$ is the Möbius function. Hence, the total number $N(m, r)$ of different polynomials $f(x)$ of degree $n = mr$, obtained by multiplying $r$ different irreducible polynomials of degree $m$, is:

$$N(m, r) = \left(\Psi_2(m)\right)^r. \quad (5.1)$$

From equation (5.1) it is seen that $N(m, r)$ is maximal for $r = \lfloor \Psi_2(m)/2 \rfloor$, resulting in a set of polynomials of degree $n = m \lfloor \Psi_2(m)/2 \rfloor \approx 2^{m-1}$. In
this case the number of distinct polynomials, all of order $2^m - 1$, can be approximated with Stirling's formula as follows:

$$N(m, \lfloor \Psi_2(m)/2 \rfloor) \approx \sqrt{\frac{m}{\pi 2^{m-1}}} \cdot 2^{2m/m}. \quad (5.2)$$

In Table 5.1 the exact value of $N(m, \lfloor \Psi_2(m)/2 \rfloor)$ is given for values of $m$ up to 15.

By substituting $m = \log_2 n$, where the log is to the base 2, in equation (5.2), the following total number $\hat{N}(n)$ of distinct polynomials $f(x)$ of degree $n$ and order $2^m - 1 \approx 2n$ is obtained:

$$\hat{N}(n) \approx \sqrt{\frac{\log 2n}{\pi n}} \cdot 2^{2n/\log 2n}. \quad (5.3)$$

All linear feedback functions associated with the polynomials in the large set of cardinality given by (5.3) can be applied to the cycle joining algorithm. By this approach $\mathcal{O}(2^{2n/\log 2n})$ DeBruijn sequences of period $2^n$ can be generated, using only $3n$ bits of storage and at most $4n$ FSR shifts to generate every bit of the sequence.

As with most of the algorithms for the generation of DeBruijn sequences the described algorithm can generate only a small fraction of all possible
DeBruijn sequences, when using linear feedback functions. However any algorithm which generates all such sequences would require $2^{n-1} - n$ bits to indicate which sequence with period $2^n$ is to be generated; this seems hardly practical.

5.5 Synthesis of the FSR equivalent

As was already mentioned in Section 5.3 the cycle joining algorithm has the effect of changing pairs of entries in the truth table representation of the feedback function $F$ of the FSR. Consequently, one can adapt $F$ while generating the particular DeBruijn sequence and simultaneously obtain the nonlinear feedback function $F_D$ which applied to a FSR generates the same DeBruijn sequence. In this way the equivalent FSR can be synthesized which generates that particular (part of the) sequence most efficiently. This demonstrates that the algorithm is also a useful tool for the synthesis of nonlinear FSR's.

The cycle joining algorithm may also be used to generate sequences, other than DeBruijn sequences, of periods less than $2^n$. This can be achieved by simply suppressing the cycle joining operation at a certain pair of adjacency quadruples. In this way classes of sequences can be generated having interesting properties, such as particular periods and, hence, linear complexities.

For general feedback functions the cycle structure may consist of only a moderate amount of relatively long cycles. Using the cycle joining algorithm in this case will result in less efficiency, i.e. a longer generation time per sequence bit. It would be advantageous here to have at one's disposal some indicator function $F_I$ which identifies the cycle representatives, i.e. a mapping of $GF(2)^n$ onto $GF(2)$ which produces a 1 output, say, for every state that is a cycle representative. Such a function can in principle always be determined for some given feedback function $F$, for example by precomputations.

**Example 5.2** For the feedback function of Example 5.1, $F(x_0, x_1, x_2, x_3) = x_0 + x_1x_2 + x_2x_3 + x_1x_2x_3$, the set of cycle representatives is \{0, 1, 3, 5, 6\} and the set of states which complement $F$ is \{0, 1, 3, 11, 9, 2, 10, 8\}.

Using the cycle joining algorithm, the equivalent feedback function $F_D$ and the cycle representative indicator function $F_I$ appear to be:

\[
F_D(x_0, x_1, x_2, x_3) = 1 + x_0 + x_1 + x_1x_2 + x_2x_3 + x_1x_2x_3,
\]
\[
F_I(x_0, x_1, x_2, x_3) = (1 + x_0)(1 + x_1 + x_2 + x_1x_3 + x_2x_3 + x_1x_2x_3).
\]
Conclusions

It can be seen that the difference between the original feedback function $F$ and $F_D$ is $1 + x_1$.

**Example 5.3** The FSR of Example 5.1 can be used to generate a sequence of period 13 by disregarding state 6 as a cycle representative.

For many purposes it may be useful to have a table of all DeBruijn sequence generating feedback functions at one’s disposal. In the appendix the feedback functions of all DeBruijn sequences of periods 16 and 32 are listed, together with the weights of the functions, their complexities according to Section 4.4 and the average deviations from the perfect linear complexity profile $\delta$ (see [Ruep 84]). The average deviation $\delta$ of a sequence $s = (s_0, s_1, \ldots, s_{2^n-1})^\infty$ is defined as:

$$\delta = 2^{-n} \sum_{i=0}^{2^n-1} \sum_{j=0}^{2^{n+1}-1} |\Lambda(s_i, s_{i+1}, \ldots, s_{i+j-1}) - [(j + 1)/2]|,$$  \hspace{1cm} (5.4)

where $\Lambda(s_i, s_{i+1}, \ldots, s_{i+j-1})$ denotes the linear complexity of the sequence $(s_i, s_{i+1}, \ldots, s_{i+j-1})$.

5.6 Conclusions

This chapter covered the introduction of an algorithm for the generation of binary DeBruijn sequences. The algorithm is based on the principle of joining cycles generated by an arbitrary feedback shift register. To this end, the notion of cycle representative was introduced and it was shown that a cycle representative and its companion are positioned on different cycles. In its operation the cycle joining algorithm can be regarded as an additional feedback function to the FSR it acts on, with the effect of complementing an entry in the truth table of the original feedback function of the FSR, every time it has encountered a cycle representative.

The result of the algorithm always is a full cycle and, consequently, all DeBruijn sequences of given order can be generated by the algorithm, depending on the feedback function of the FSR. It was shown that for linear FSR’s, taking the feedback polynomial to be reducible, containing many low degree irreducible polynomials, the algorithm is very efficient, viz. it requires only $3n$ bits of storage and at most $4n$ FSR shifts to generate every bit of the sequence.

It was also shown that the cycle joining algorithm can be adapted to generate sequences with periods less than $2^n$, by leaving out one or more
cycles. Moreover, it was demonstrated that the algorithm is useful in obtaining the feedback function of the FSR which generates a particular (part of a) sequence most efficiently.
Chapter 6

Binary Sequence Generators Based on Source Coding Algorithms

6.1 Introduction

In Chapter 3 a theory was developed concerning the maximum order complexity of sequences. It was demonstrated that random sequences have an expected complexity close to twice the logarithm of their length. Consequently, it seems natural to look for classes of sequences which approach the complexity of real random sequences. Instead of considering all possible classes of sequences that have been published in the literature, in this chapter two methods for constructing sequences are investigated, which are based on source coding algorithms.

The motivation to look for new construction methods based on source coding algorithms, comes to a large extent from the paper of Lempel and Ziv [Lemp 76]. In this paper a new complexity measure for finite sequences is defined and it is shown that in particular DeBruijn sequences [DeBr 46] have the highest Ziv-Lempel complexity. Also the famous Berlekamp-Massey algorithm [Mass 69], used for BCH decoding and linear feedback shift register synthesis, can be viewed at as a source coding algorithm for a particular class of sequences. In the latter case the decoding algorithm constitutes the generation of a linear FSR sequence.

In Section 6.2 sequences generated by the Ziv-Lempel decoding algorithm are discussed. As a result, a classification of some known construction methods for sequences is given. Section 6.3 describes how to construct large
classes of binary sequences, which satisfy Golomb's first and second randomness postulates ([Golo 67]). The construction method is based mainly on enumerative coding principles.

The construction methods presented in this chapter are already fairly complex and seem difficult to analyze, even for the binary case. For this reason we restrict ourselves to binary sequences only, although the construction methods can easily be generalized.

6.2 Sequences Generated by a Switch Controlled Feedback Shift Register

6.2.1 The Ziv-Lempel Data Compression Algorithm

In [Ziv 77] a universal algorithm for sequential data compression has been presented. The algorithm is based on the complexity considerations of Lempel and Ziv as described in [Lemp 76] and uses maximum length copying from a buffer containing a fixed number of most recent output characters. The idea is that a sequence can be parsed into successive components (subsequences). Each component is a copy of some subsequence occurring earlier in the sequence, except for its last character which is called the innovation. So each component is fully determined by a pointer to the starting position of the subsequence to be copied, the length of this subsequence and the innovation character. The compression is generally obtained by only transmitting this pointer value, the length and the innovation character.

In the coding algorithm a buffer of length $L_b$ is employed, which contains the last $L_b$ characters of some sequence $z$. Initially the buffer is filled with some fixed sequence. Successive subsequences of $z$ are encoded by searching for the longest subsequence in the buffer which can be copied to obtain the sequence $z$. The length of the subsequence to be copied from the buffer is also limited to some prescribed value $L_s$. Hence, the codewords consist of a starting point in the buffer (called the pointer), the copy length and the innovation character.

The decoding algorithm works in a similar way. It also uses a buffer of length $L_b$, which contains the last $L_b$ characters of the sequence $z$ to copy from. The decoding algorithm can very elegantly be implemented with a shift register as depicted in Figure 6.1. If $W_i = w_i^1 w_i^2$ denotes the $i$th codeword, then $w_i^1$ denotes the position of the tap of the shift register which is fed back to the input, $w_i^2$ denotes the number of shifts the register
performs in this feedback mode, and $w_i^3$ denotes the innovation character which is shifted into the register without feedback.

**Example 6.1** Let $L_b = 2$, $L_s = 4$ and suppose we consider binary sequences only. Clearly we have codewords of 4 bits. The sequence $(1,1,0,1,0,0)$ is encoded as the following sequence of codewords: $(001,020,120)$. The sequence of codewords $(031,121)$ is decoded as the sequence $(0,0,0,1,0,1,1)$.

### 6.2.2 Generating Sequences with the Ziv-Lempel Decoding Algorithm

The Ziv-Lempel decoding algorithm is simplified if we restrict the character alphabet to be $GF(2)$. In this case the innovation character does not have to be encoded, as it is the complement of the character which follows the subsequence that has been copied from the register. So now $w_i^2$ shifts are performed in the feedback mode, followed by one shift with the feedback complemented.

Our approach is to use the binary Ziv-Lempel decoding algorithm as a sequence generator. To achieve this, the codewords comprising $w_i^1$ and $w_i^2$ are generated by linear feedback shift registers. In doing so, we aim at obtaining an output sequence which resembles a real random sequence very well. In particular the maximum order complexity of the sequence is of interest, as this method of copying subsequences guarantees a minimum value of the complexity. It seems, however, that this sequence generator is rather complex to analyze in great detail and therefore we restrict ourselves to the special case where $w_i^2 = 0$ for all $i$, as depicted in Figure 6.2. We call this FSR sequence generator a *Switch Controlled FSR*. It should be emphasized...
that, as a result of the aforementioned restriction, the relationship with the Ziv-Lempel decoding algorithm is strongly weakened and we may expect to obtain less significant results.

6.2.2.1 The Period of the Generated Sequence

Let \( s_0, s_1, \ldots, s_{r-1} \) denote periodic sequences with periods \( p_0, p_1, \ldots, p_{r-1} \), i.e. \( s_i = (s_{0,i}, s_{1,i}, \ldots, s_{p_i-1,i})^{\infty} \) with all \( s_{j,i} \in GF(2) \). Also, let \( W_j \) denote the binary \( r \)-tuple \( (s_{j,0}, s_{j,1}, \ldots, s_{j,r-1}) \) which controls the position of the feedback tap, hence, the sequence of \( r \)-tuples \( W_j \) is periodic with period \( p = \text{lcm}(p_0, p_1, \ldots, p_{r-1}) \). In the sequel it is assumed that the register length \( L_b = 2^r \). Moreover it is assumed that if \( W_j = (0, \ldots, 0) \), the feedback tap is at \( z_{j-1} \), if \( W_j = (0, \ldots, 0, 1) \) the feedback tap is at \( z_{j-1} - 2 \) and if \( W_j = (1, \ldots, 1) \), the feedback tap is at \( z_{j-2^r} \).

The output sequence \( z \) satisfies the recursion:

\[
\begin{align*}
z_j &= z_{j-I(W_j)} + 1 \\
&= F(W_j, z_{j-1}, \ldots, z_{j-2^r}) \\
&= 1 + \sum_{i=0}^{2^r-1} F_i(W_j) z_{j-i-1},
\end{align*}
\]

where \( I(W_j) \) denotes the integer representation of \( W_j \) according to Definition 5.2 and the switching functions \( F_i \), \( i = 0, 1, \ldots, 2^r - 1 \) are mappings from \( GF(2)^r \) onto \( GF(2) \), defined as:

\[
F_i(X) = \begin{cases} 
1, & \text{if } I(X) = i, \\
0, & \text{else.}
\end{cases}
\]
Equation (6.1) shows that our sequence generator can be regarded as a FSR with a nonlinear feedback function containing memory.

The generated sequence \( z \) will ultimately be periodic with some period \( p_z \). To determine the period \( p_z \) we first determine the period \( p_d \) of the sequence \( d \), which is obtained from \( z \) by addition of a one times delayed version, i.e. \( d_j = z_j + z_{j-1} \) for all \( j \). The recursion satisfied by \( d \), can be derived by applying equation (6.1) twice. Hence, we obtain:

\[
d_j = \sum_{i=0}^{2^r-1} \left\{ F_i(W_j)d_{j-i-1} + \left[ F_i(W_j) + F_i(W_{j-1}) \right] z_{j-i-2} \right\}.
\] (6.2)

By adding \( z_{j-1} \) to both sides of equation (6.1) a recursion for \( d \) is obtained which resembles that of \( z \), i.e.:

\[
d_j = 1 + \sum_{i=0}^{2^r-1} G_i(W_j)d_{j-i-1}
\] (6.3)

\[
= 1 + \sum_{i=1}^r d_{j-i},
\] (6.4)

where the switching functions \( G_i, i = 0, 1, \ldots, 2^r - 1 \) are mappings from \( GF(2)^r \) onto \( GF(2) \), defined as:

\[
G_i(X) = 1 + \sum_{k=0}^i F_k(X) = \begin{cases} 0, & \text{if } I(X) \leq i, \\ 1, & \text{else}. \end{cases}
\]

The following result is an immediate consequence of the recursion relation (6.4).

**Lemma 6.1** Let \( W_j \) and \( I(\cdot) \) be as defined before. If for some \( i > 0 \), we have that \( I(W_{i+k}) \leq k \), for \( k = 0, 1, \ldots, 2^r-1 \), then \( d_j = d_{j+p} \) for all \( j \geq i \).

**Proof.** Equation (6.4) implies that by the condition \( I(W_{i+k}) \leq k \), for \( k = 0, 1, \ldots, 2^r-1 \) all \( d_j, j \geq i \), are completely determined by the sequence of \( W_j \)'s and not by the initial \( z \)-register contents. But because of the periodicity of the sequence of \( r \)-tuples the same is true for the \( d_{j+p} \). The result now follows by induction on the \( d_j \).

Note that Lemma 6.1 states a sufficient condition for the period of \( d, p_d \), to divide \( p \), but this condition is not strictly necessary.

If the switch driving sequence is generated by a linear FSR, the period of \( d \) is exactly equal to \( p \), as expressed by the next proposition.
Proposition 6.2 If the \( r \) sequences \( s_0, s_1, \ldots, s_{r-1} \) are taken from \( r \) different stages of a linear feedback shift register with irreducible connection polynomial having period \( p \), and the condition \( W_i = W_{i-1} = \cdots = W_{i+2-2r} = (0, \ldots, 0) \) holds, then the period \( p_d \) of \( d \) is equal to \( p \).

Proof. From Lemma 6.1 it follows that \( p_d \) divides \( p \). Due to equation (6.4) the condition \( W_i = W_{i-1} = \cdots = W_{i+2-2r} = (0, \ldots, 0) \) implies that a run of \( 2^r - 1 \) or more ones occurs at least once in \( d \). This run, however, can only be terminated if the switch, pointing to an even tap position, is set to point to an odd position. This odd position can only be obtained by having \( s_{i,0} = 1 \) for some \( i > 0 \). Consequently, as \( d_j = d_{j+p_d} \) for all \( j \), there must at least exist one \( i \) such that \( s_{i,0} = s_{i+p_d,0} = s_{i+2p_d,0} = \cdots = 1 \). As the sequence \( s_0 \) is periodic with period \( p \), it follows from the decimation principle (see e.g. [Ruep 84, Ch. 6]) that \( p_d = p \). \( \square \)

In a similar way one can prove that Proposition 6.2 also holds if the \( r \) sequences, which constitute \( W_j \), are obtained from \( r \) separate linear FSR's with primitive connection polynomials which are pairwise relatively prime.

From the relation between \( d \) and \( z \) it is easy to see that the period \( p_z \) of \( z \) is equal to \( p_d \) if the number of ones in one period of \( d \) is even. If, however, the number of ones in one period of \( d \) is odd, \( p_z = 2p_d \) and the second half of \( z \) is the complement of the first half.

Example 6.2 Let \( r = 1 \) and let the sequence \( s_0 \) be generated by a linear FSR of length 4, with irreducible connection polynomial \( x^4 + x^3 + x^2 + x + 1 \). Hence, the switch driving sequence is periodic with \( p = 5 \). Depending on the initial state of the LFSR one of the three sequences

\[
\begin{align*}
  s_a &= (1,1,1,1,0)^\infty, \\
  s_b &= (1,1,0,0,0)^\infty, \\
  s_c &= (1,0,1,0,0)^\infty,
\end{align*}
\]

is obtained as \( s_0 \). It appears that \( s_b \) and \( s_c \) give rise to one and essentially the same \( d \) sequence. So we have:

\[
\begin{align*}
  d_a &= (1,1,0,1,0)^\infty, \\
  d_b &= (0,1,1,1,1)^\infty.
\end{align*}
\]

The corresponding \( z \) sequences are:

\[
\begin{align*}
  z_a &= (0,1,0,0,1,1,0,1,1,0)^\infty, \\
  z_b &= (0,0,1,0,1)^\infty,
\end{align*}
\]

and their complements, depending on the initial contents of the \( z \)-register.
Connection polynomial of driving LFSR & Period & $\Lambda(d)$
\hline
$x^3 + x + 1$ & 7 & 6 \\
$x^4 + x + 1$ & 15 & 12 & 10 & 14 \\
$x^5 + x^2 + 1$ & 31 & 30 & 31 & 31 \\
$x^6 + x + 1$ & 63 & 48 & 56 & 60 \\
$x^7 + x^3 + 1$ & 127 & 126 & 127 & 120 \\
$x^8 + x^4 + x^3 + x^2 + 1$ & 255 & 250 & 255 & 254 \\
$x^9 + x^4 + 1$ & 511 & 510 & 511 & 510 \\
$x^{10} + x^3 + 1$ & 1023 & 1007 & 1013 & 1022 \\
$x^{11} + x^2 + 1$ & 2047 & 2046 & 2047 & 2047 \\
\hline

Table 6.1: Linear complexity of $d$. 

Connection polynomial of driving LFSR & $c(z^1)$
\hline
$x^3 + x + 1$ & 2 \\
$x^4 + x + 1$ & 6 & 5 & 8 \\
$x^5 + x^2 + 1$ & 12 & 7* & 12* \\
$x^6 + x + 1$ & 11 & 10 & 12 \\
$x^7 + x^3 + 1$ & 21 & 17* & 15* & 12 \\
$x^8 + x^4 + x^3 + x^2 + 1$ & 16 & 17* & 15 & 13 \\
$x^9 + x^4 + 1$ & 28 & 23* & 16 & 19* \\
$x^{10} + x^3 + 1$ & 41 & 26* & 20 & 20* \\
$x^{11} + x^2 + 1$ & 45 & 31* & 33* & 20* \\
\hline

Table 6.2: Maximum order complexity of a single period of $z$. 

6.2.2.2 The Complexity of $z$

The main purpose of the nonlinear switching function is to generate an output sequence with complexity close to that of a real random sequence, whereas the switch driving sequences have low complexities. Also, the good statistical properties should be maintained. The linear complexity of $d$ as well as the maximum order complexity of $z$ have been determined experimentally for several values of $r$ and the length $L$ of the linear FSR driving the feedback switch. In the experiment it was ensured that $d$ has the same period as the switch driving sequences. The results of these experiments are listed in Tables 6.1 and 6.2. Table 6.1 shows that the linear complexities have values very close to the corresponding periods, which is
obviously what one would expect for periodic random sequences [Ruep 84, pg. 51]. It also appears, however, that for some combinations of linear FSR and \( r \) degeneracies occur.

The maximum order complexities as listed in Table 6.2 were determined using one period of \( z \) if \( p_z \) was equal to \( p_d \), else only half a period of \( z \) was used. In the latter case the table entries are marked with a * and have \( p_z = 2p_d \). From the table it can be seen that with every value of \( r \) there tends to be some region in the LFSR order \( L \), for which \( c(z^1) \approx 2L \). For higher and lower values of \( L \) the complexities can deviate substantially from the expected value of \( 2L \).

For the case \( r = 1 \) the sequence generator of Figure 6.2 allows a nice alternative representation. In this case there is only one switch driving sequence, which determines whether \( z_j \) or \( z_{j-2} \) is fed back. We find the following recursion relations:

\[
\begin{align*}
  z_j &= (s_j + 1)z_{j-1} + s_jz_{j-2} + 1, \quad (6.5) \\
  d_j &= s_jd_{j-1} + 1. \quad (6.6)
\end{align*}
\]

The corresponding \( d \) sequence generator is depicted in Figure 6.3. The recurrence relation (6.6) can be written as:

\[
  d_j = 1 + s_j(1 + s_{j-1}(1 + s_{j-2}(1 + \cdots))). \quad (6.7)
\]

However, since \( s \) is a linear FSR sequence, the product terms in (6.7) of degree exceeding the FSR length \( L \) will all be zero. Hence, we obtain:

\[
\begin{align*}
  d_j &= 1 + s_j(1 + s_{j-1}(1 + \cdots (1 + s_{j-L+1}))) \quad (6.8) \\
  &= 1 + s_j + s_js_{j-1} + \cdots + s_js_{j-1} \cdots s_{j-L+1}. \quad (6.9)
\end{align*}
\]

So for \( r = 1 \) the sequence \( d \) can be obtained directly from the LFSR by a nonlinear feedforward function as given by expression (6.9). For nonlinear

\[
\begin{align*}
  d_j &= 1 + s_j(1 + s_{j-1}(1 + s_{j-2}(1 + \cdots))). \\
  &= 1 + s_j + s_js_{j-1} + \cdots + s_js_{j-1} \cdots s_{j-L+1}.
\end{align*}
\]
Figure 6.4: Equivalent JK-flipflop generator for \( r = 1 \).

Feedforward functions, acting on LFSR sequences, Rueppel [Ruep 84] has given a lowerbound on the linear complexity. However, because the highest order product term in (6.9) has degree \( L \), this lowerbound does not explain the experimental results as listed in Table 6.1. In connection with this table we conjecture that for \( r = 1 \) and \( L \) prime the linear complexity of \( d \) is equal to \( 2^L - 2 \).

### 6.2.2.3 Statistical Properties

The recurrence relation (6.1) provides for a balanced number of ones and zeroes to be stored in the register. This may be regarded as a form of 'negative feedback', i.e. an unbalance in the distribution of ones and zeroes will eventually be corrected.

If we assume that the switch is driven by real random sequences, then the expectation of an output character is fully determined by the distribution of the previous characters stored in the register. Consequently, there exists a correlation between consecutive characters of \( z \). This correlation depends on the register length \( L_b \). If \( L_b \) is small the correlation is rather strong, which is demonstrated by the case \( r = 1 \). Then equation (6.5) shows that, if \( z_{j-2} = z_{j-1} \), \( z_j \) becomes the complement of \( z_{j-1} \), independent of \( s_j \). As a result the maximum run-length of runs in \( z \) will be 2. In general the maximum run-length of runs in \( z \) will be limited to \( L_b \). We therefore conclude that the period \( p \) of the switch driving sequence should be close to \( 2^{L_b} \) and \( L_b \) sufficiently large.

There is obviously also correlation between the sequences \( s \) and \( z \). The recurrence relation (6.9) can be obtained by a simplification of the 'flipflop' sequence generator, proposed by Pless [Ples 77]. From equation (2.1) of Chapter 2 it can be seen that the \( d \) sequence can be obtained from the \( s \) sequence through one JK-flipflop. Generating \( z \) requires an additional JK-flipflop as depicted in Figure 6.4. In [Rubi 79] it was shown that it is computationally feasible to break Pless' cipher by using the correlation...
between input and output sequences. A similar attack is possible on the generator proposed in this section for larger values of $L_b$. However, the computational complexity will be highly increased.

### 6.2.3 Improvement of the Generator

The main principle of the sequence generator proposed in this section, is to generate consecutive characters by copying from the finite vocabulary of a fixed length register. The vocabulary of the register is recursively updated with newly generated characters. Despite the high complexity of the generated sequences, their statistical properties appear to be imperfect as a result of this recursion relation. The finite vocabulary is clearly a drawback and therefore it may be useful to combine the switching function with an 'infinitely' large vocabulary, guaranteeing both large complexity and good statistical properties.

Such large vocabularies can be obtained with similar techniques as used in Rueppel's random sequence generator, [Ruep 84, pg. 131], or the multiplex scheme of Beker and Piper [Beke 82, pp. 240–245]. In fact these two sequence generators belong to, what we propose to call, the category of the general multiplexed FSR's, denoted $\Omega_M$. The general structure of this generator is depicted in Figure 6.5. Among the many other categories of sequence generators one can devise, the other one of interest here is the category of the feedforward filtered FSR's, denoted $\Omega_F$, as depicted in Figure 6.6. Our switch controlled FSR clearly belongs to $\Omega_F$.

We now give a classification according to both categories of some known sequence generators and possible extensions.

1. General Multiplexed FSR's:
Switch Controlled FSR

Figure 6.6: Structure of feedforward filtered FSR’s.

a. The degenerate case with only one LFSR is the nonlinear (memoryless) feedforward function generator.

b. Beker and Piper’s multiplex generator with two LFSR’s.

c. Rueppel’s “in-product generator”, which can be regarded as an extension of Beker and Piper’s generator, by allowing all sums of FSR taps.

d. Both 1b and 1c can be extended by taking multiple LFSR’s to generate the switch driving sequences.

e. A special case of 1d is the generator consisting of multiple LFSR’s and a nonlinear combining function as analyzed by Siegenthaler [Sieg 83].

2. Feedforward Filtered FSR’s:

a. The degenerate case with no driving FSR is the autonomous feedback shift register.

b. A generator with a combiner comprising one bit of memory to frustrate a correlation attack as described by Rueppel [Ruep 84, pp. 182-197].

c. Pless’ JK-flipflop generator.

d. Bernasconi and Günther’s sequence generator [Bern 85].

One might come to the conclusion that the difference between both categories is the presence of memory in the combining function. However, our switch controlled FSR with \( r = 1 \) falls under both categories, and hence \( \Omega_M \cap \Omega_F \neq \emptyset \).
6.3 Run Permuted Sequences

This section is concerned with the construction of classes of binary sequences, which are obtained by permuting the runs of zeroes and ones of some given periodic binary sequence \( s = (s_0, s_1, \ldots, s_{p-1})^\infty, s_i \in GF(2) \). To this end, we will first introduce the run-length notation of a periodic sequence. Then a large class of sequences will be constructed by permuting the runs of zeroes and ones of a DeBruijn sequence of given order, and discuss the properties of the sequences in this class with respect to their maximum order complexities. It is shown that in this way all DeBruijn sequences of this order are obtained, but also many more sequences with higher complexities. Finally we discuss how to generate the sequences in this class with the use of enumerative coding techniques.

6.3.1 Run-Length Notation of Periodic Sequences

Run-length coding (see e.g. [Golo 66]) is a source coding technique often used in practice for the purpose of data compression. The technique is very simple: a sequence is parsed into runs of ones and zeroes, and the length of each run is used as the codeword. A run of \( n \) characters is a subsequence of \( n \) consecutive identical characters. For example, the binary sequence \( (1,1,0,1,0,0,1)^\infty \) is run-length coded as the integer sequence \( (2,1,1,2,1,1)^\infty \), where the index 1 denotes the fact that the binary sequence starts with a 1.

Let \( \mathcal{R} \) denote the run-length coding operator acting on binary sequences. Clearly, \( \mathcal{R} \) maps \( GF(2)^* \) onto \( N^* \), the set of all sequences of integers. In the sequel we will only consider periodic sequences. Periodic sequences require some extra attention with respect to their run-length representations. Consider the two sequences \( (1,1,0,1,0,0,1)^\infty \) and \( (1,1,1,0,1,0,0)^\infty \). Obviously, these two sequences are essentially the same as only their initial phases differ. However, the first sequence would be encoded into 5 runs, whereas the second sequence comprises only 4 runs. Not mentioning the decoding problems of the first sequence, this ambiguity should be avoided. This can easily be achieved by requiring the sequence to be shifted in such a phase that the characters it begins and ends with are unequal. As there are usually many possibilities to do this, we choose for some standard form, i.e. we always let the sequence start with its longest run of ones. Consequently, we have the following definition:
Definition 6.1 The run-length representation $r = R_s$ of the binary periodic sequence $s$ is defined as the periodic sequence of positive integers, denoting the consecutive runs of ones and zeroes of $T_s$, where $T_s$ is a shifted version of $s$ such that it begins with its longest run of ones.

It should be noted that in this form of run-length representation the convention to begin with ones has as a consequence that $r$ cannot be shifted over some arbitrary phase, but rather over even shifts only.

It can be seen that if the binary sequence $s$ has period $p > 1$, then the run-length sequence $r$ can have any even period $q$, with $2 \leq q \leq p - 1$.

6.3.2 Classes of Run Permuted Sequences

The main objective of this section is to construct classes of sequences by permuting the runs of ones and zeroes of some given periodic binary sequence. The procedure considered here is the following: the given sequence is written in its run-length representation according to Definition 6.1, then the integers representing the runs of ones and the integers representing the runs of zeroes are permuted independently. The new sequence of integers obtained in this way is then transformed back into a binary sequence. The reason for doing this is twofold:

- The described procedure preserves the R1 and R2 properties of the original sequence ([Golo 67]), i.e. the number of ones and zeroes as well as the distribution of the runs remain unaltered.

- We expect that the sequences obtained in this way have interesting properties such as a good maximum order complexity.

In the remainder of this section we restrict ourselves to sequences with the so-called run property ([Golo 80, Chen 83], i.e. sequences which satisfy Golomb's second postulate perfectly, in particular DeBruijn sequences, as we want to obtain random looking sequences. Obviously, it is also possible to apply the described procedure to sequences not having the run property, hence obtaining classes of sequences with different properties.

Most often one considers sequences with period $p = 2^n - 1$ with the run property, however, we will consider sequences with the run property which are periodic with period $p = 2^n$. The number of runs of ones and zeroes of any length of these sequences are exactly the same. This number of runs of either zeroes or ones of length $l$, $R_l^n$, satisfies analogous to the definition
in [Golo 67, pp. 43-44]:

\[ R_i^n = \begin{cases} 
2^{n-l-2}, & 1 \leq l < n-1, \\
0, & l = n - 1, \\
1, & l = n. 
\end{cases} \quad (6.10) \]

From (6.10) it can be seen that the total number of runs, \( R^n \), of either ones or zeroes satisfies:

\[ R^n = 1 + \sum_{i=1}^{n-2} R_i^n = 2^{n-2}. \quad (6.11) \]

Only by permuting the runs of ones and zeroes independently can we guarantee to obtain sequences which have the run property. Moreover, several permutations yield essentially the same sequence, i.e. phase shifted versions of one sequence. These two considerations allow us to count the number of essentially different sequences obtained in this way.

**Proposition 6.3** Let \( C_n \) denote the class of essentially different sequences obtained by independently permuting the runs of ones and zeroes of a De-Bruijn sequence of order \( n \). The number of sequences in the class is given by the following equation:

\[
|C_n| = 2^{-n+2} \left\{ \frac{(2^{n-2})!}{\prod_{i=1}^{n-2} (2^{n-l-2})!} \right\}^2. \quad (6.12)
\]

**Proof.** Let \( P_n \) denote the set of sequences of period \( 2^n \), obtained by permuting the runs of either ones or zeroes of a DeBruijn sequence of order \( n \). There are \( R^n \) runs in total, however, \( R_i^n \) runs of length \( l \). Hence, the number of permutations of runs satisfies:

\[
|P_n| = \frac{(R^n)!}{\prod_{i=1}^{n-2} (R_i^n)!} \cdot \frac{(2^{n-2})!}{\prod_{i=1}^{n-2} (2^{n-l-2})!}. \quad (6.13)
\]

The total number of sequences obtained by independently permuting the runs of ones and zeroes is obviously equal to \( |P_n|^2 \). Amongst these there are sequences of runs which are cyclic shifts of each other, resulting in essentially the same binary sequence. As \( R_n^n = 1 \), the sequences of runs are periodic with period \( R^n \) and hence we have:

\[
|C_n| = \frac{|P_n|^2}{R^n}.
\]

This is exactly what equation (6.12) states. \( \square \)
By straightforward application of \( \binom{n}{k} = \frac{n!}{k!(n-k)!} \), one can obtain the following expression for \(|P_n|\):

\[
|P_n| = \prod_{i=1}^{n-2} \left( \frac{2^l}{2^{l-1}} \right).
\]

Consequently, we have:

\[
|C_n| = 2^{-n+2} \prod_{i=1}^{n-2} \left( \frac{2^l}{2^{l-1}} \right)^2.
\] (6.14)

The binomial coefficients in (6.14) can be approximated using Stirling's approximation formula, see e.g. [Abra 70], resulting in:

\[
\binom{k}{k/2} \approx \frac{2^{k+1}}{\sqrt{2\pi k}}.
\] (6.15)

Using this approximation (6.15) yields:

\[
\left( \frac{2^l}{2^{l-1}} \right)^2 \approx \frac{2}{\pi} 2^{2l+1-l} = \frac{8}{\pi} G_{l+2},
\] (6.16)

where \(G_{l+2}\) denotes the number of binary DeBruijn sequences of order \(l+2\). Combining (6.14) and (6.16) and taking into account that \(G_1 = G_2 = 1\), finally results in:

\[
|C_n| \approx \left( \frac{4}{\pi} \right)^{n-2} \prod_{k=1}^{n} G_k.
\] (6.17)

The approximation (6.17) is not very good, as can be seen from a small example. If for example \(n = 5\), (6.17) yields \(|C_5| \approx 135272\), whereas (6.14) yields \(|C_5| = 88200\). The approximation error is due to the approximation errors of the binomial coefficients in equation (6.14), which accumulate because of the product in this equation. Therefore we propose to write (6.17) as:

\[
|C_n| = \rho_n \left( \frac{4}{\pi} \right)^{n-2} \prod_{k=1}^{n} G_k,
\] (6.18)

or equivalently, when the product is evaluated:

\[
|C_n| = \rho_n \left( \frac{4}{\pi} \right)^{n-2} 2^{2n-(n+1)n/2-1}.
\] (6.19)
Table 6.3: Correction factor $\rho_n$ of equation (6.18).

<table>
<thead>
<tr>
<th>$n$</th>
<th>$\rho_n$</th>
<th>$n$</th>
<th>$\rho_n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.7854</td>
<td>5</td>
<td>0.6222</td>
</tr>
<tr>
<td>2</td>
<td>0.6940</td>
<td>6</td>
<td>0.6173</td>
</tr>
<tr>
<td>3</td>
<td>0.6520</td>
<td>7</td>
<td>0.6149</td>
</tr>
<tr>
<td>4</td>
<td>0.6320</td>
<td>8</td>
<td>0.6137</td>
</tr>
</tbody>
</table>

In the above equation (6.18) the correction factor $\rho_n$ is less than 1 for all $n$ and computer analysis shows that it converges to 0.61 $\cdots$ for large $n$. Some values of $\rho_n$ are given in Table 6.3.

Equation (6.18) clearly shows that the fraction of DeBruijn sequences contained in $C_n$ goes to zero for large $n$. However, all DeBruijn sequences of order $n$ are contained in $C_n$, as DeBruijn sequences are by definition all those sequences in which all subsequences of length $n$ occur exactly once.

It can easily be shown that all sequences in $C_n$ constitute a small fraction of all binary (balanced) sequences with period $2^n$. To this end, let $U_n$ denote the set of all essentially different binary sequences of period $2^n$ and let $K_n \subset U_n$ denote the subset of all such sequences which are balanced, i.e. have an equal number of ones and zeroes. Clearly $C_n \subset K_n \subset U_n$. The cardinalities of both sets are:

$$|U_n| = \frac{2^{2n} - 2^{2n-1}}{2^n} \approx 2^n (G_n)^2,$$  \hspace{1cm} (6.20)

$$|K_n| = \frac{\left(\begin{array}{c} 2^n \\ 2^{n-1} \end{array}\right) - \left(\begin{array}{c} 2^{n-1} \\ 2^{n-2} \end{array}\right)}{2^n} \approx \frac{2^{2n-3n/2+1}}{\sqrt{2\pi}}.$$  \hspace{1cm} (6.21)

Using (6.19) together with (6.20) and (6.21), we obtain:

$$\frac{|C_n|}{|U_n|} \approx \rho_n \left(\frac{4}{\pi}\right)^{n-2} 2^{-1-n(n-1)/2},$$  \hspace{1cm} (6.22)

$$\frac{|C_n|}{|K_n|} \approx \rho_n \sqrt{2\pi} \left(\frac{4}{\pi}\right)^{n-2} 2^{-2-n(n-2)/2}.$$  \hspace{1cm} (6.23)

From (6.22) and (6.23) we see that both fractions go to zero asymptotically.

It should be noted here that the results discussed so far also hold for sequences of period $2^n - 1$, which have the run property, such as the maximum length linear feedback shift register sequences.
### 6.3.3 Properties of the Sequence Class $C_n$

As was mentioned before, $C_n$ contains all DeBruijn sequences of order $n$, but these sequences constitute only a small fraction of the entire class. The other sequences in $C_n$ must necessarily have maximum order complexities higher than $n$. Concerning this complexity we have the following results:

**Proposition 6.4** For all sequences $s \in C_n$, $n > 2$, the maximum order complexity $c(s)$ satisfies the inequality: $n \leq c(s) \leq 2^{n-1} - 1$.

**Proof.**

- **i)** The lowerbound is trivially satisfied by DeBruijn sequences and, as the sequences have period $2^n$, the complexity cannot be less than $n$.

- **ii)** For $n = 3$ the lowerbound coincides with the upperbound. For $n > 3$ the upperbound is obtained by sequences constructed as follows. We divide all runs into two sets; this is possible for all runs except for the two longest runs of both ones and zeroes, which are unique. With these two sets we construct two identical sequences. Then the longest runs of ones and zeroes are placed in front of the first sequence and the longest but one runs in front of the second sequence. The two sequences are now concatenated to one sequence. The longest subsequence which occurs twice in this sequence clearly has length $2^{n-1} - 2$ as depicted below:

\[
\begin{array}{cccccccccccc}
1 & 1 & 1 & 0 & 0 & 0 & \cdots & 0 & 1 & \cdots & 1 & 0 & 0 & \cdots & 0 & 1 & \cdots & 0 \\
n & n & 2^{n-1} - 2n + 2 & n-2 & n-2 & 2^{n-1} - 2n + 2 & \infty & \\
\end{array}
\]

Here, the subsequences start with bold printed 0's and end with bold printed 1's.

Clearly, it is impossible to construct a sequence with two subsequences of greater length than $2^{n-1} - 2$, as these subsequences have to be 'disjoint' due to the unequal lengths of the various runs.

\[\Box\]

**Corollary 6.5** For all sequences $s \in C_n$, $n > 2$, the maximum order complexity $c(s)$ cannot be equal to $2^{n-1} - 2$. 
Proof. This follows immediately from the proof of the previous proposition, viz. it is impossible to delete one binary character from the two longest subsequences by interchanging runs, as the number of runs is even. \qed

The number of sequences with maximum order complexity of $2^{n-1} - 1$ can easily be counted.

**Proposition 6.6** Let $M_n \subseteq C_n$ denote the subset of sequences in $C_n$ having maximum order complexity of $2^{n-1} - 1$. The number of sequences in this set satisfies $|M_n| = 2^{-n+5}|C_{n-1}|$, for all $n \geq 4$.

**Proof.**

i) All the runs of ones and zeroes, that occur at least twice and of which there are $2(2^{n-3} - 1)$ of each kind, can be permuted arbitrarily, thus accounting for

$$\left\{ \frac{(2^{n-3} - 1)!}{n^{-3} \prod_{i=1}^{n-3} (2^{n-1-i})!} \right\}^2$$

possibilities.

ii) There are exactly 2 ways of positioning the runs of lengths $n$ and $n - 2$ in the construction of the proof of Proposition 6.4, viz.:

1. $(n, n, \ldots, n - 2, n - 2, \ldots) \infty$
2. $(n, \ldots, n, n - 2, \ldots, n - 2) \infty$

All the other ways necessarily result in a lower complexity.

iii) The roles of the one and zero runs can be interchanged.

From these three arguments we conclude that

$$|M_n| = 4 \left\{ \frac{(2^{n-3} - 1)!}{n^{-3} \prod_{i=1}^{n-3} (2^{n-1-i})!} \right\}^2$$

Combining this with equation (6.12), the desired result follows. \qed
It appears not to be straightforward to find a general expression for the number of sequences in $C_n$ with given complexity, other than for the three values shown above. However, some small numerical examples can give an impression of the distribution of complexity.

Example 6.3

1. For $n = 4$ we have $C_4 = 36$, $G_4 = 16$ and $M_4 = 4$. The 16 remaining sequences all have $c = 5$. The average complexity $\bar{c}$, where the average is taken over $C_4$, is 4.7778.

2. For $n = 5$ we have $C_5 = 88200$, $G_5 = 2048$ and $M_5 = 36$. The distribution of complexity of the other sequences is as follows:

<table>
<thead>
<tr>
<th>$c$</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
</tr>
</thead>
<tbody>
<tr>
<td>#($)</td>
<td>17376</td>
<td>37824</td>
<td>19824</td>
<td>8048</td>
<td>1840</td>
<td>860</td>
<td>256</td>
<td>88</td>
</tr>
</tbody>
</table>

The average complexity $\bar{c}$ is equal to 7.2892.

6.3.4 Generation of Run Permuted Sequences

The sequences we have discussed so far in this section can also be generated efficiently. The generation process is based on enumerative coding techniques, applied to generate permutations.

Usually, enumerative coding is applied to compress individual sequences with special properties, such as constant weight sequences or sequences with a prescribed number of 0-0, 0-1, 1-0 and 1-1 transitions. The first method was described by Schalkwijk [Scha 72], the latter by Cover [Cove 73].

We will use an example to demonstrate the enumerative encoding and decoding of permutations. Consider all sequences which comprise one $a$, one $b$ and two $c$'s; clearly there are $4!/2! = 12$ of these sequences. Let us assume a lexicographic ordering $a < b < c$ and define the leftmost character in the sequence to be the most significant character. All 12 sequences are represented by the ternary tree of Figure 6.7. The numbers written with the nodes of the tree, denote the number of leaves that can be reached from that node, i.e. the number of sequences starting with $a$, $b$, $c$, $aa$, $ab$, .... These numbers can easily be determined. For example, if $N_x$ denotes the number of sequences starting with an $x$, where $x$ is a sequence of length less than 4 from $\{a, b, c\}$, we have $N_a = 3!/2!$, $N_b = 3!/2!$, $N_c = 3!/1!$, $N_{ab} = 2!/2!$, and so on. From the code tree it can be seen that $cabc$ is
coded into \((3 + 3) + 0 + 0 = 6\). The codewords are often called the indices, written as \(i(cabc) = 6\). The general expression for the indices is:

\[
i(\xi_0, \xi_1, \xi_2, \xi_3) = \sum_{n=0}^{2} \sum_{\nu_n < \xi_n} N_{\xi_n \cdots \xi_1 \nu_n},
\]

where \(\nu_n, \xi_n \in \{a, b, c\}\), for \(n = 0, 1, 2, 3\). Using the above expression we see that \(i(abcc) = 0\) and \(i(ccba) = 11\).

The decoding process uses again the node numbers of the code tree. As an example, consider the decoding of an index with value 5. Clearly, the sequence cannot start with an \(a\), as all such sequences have indices less than 3. Also, the sequence cannot start with a \(c\), as all of these sequences have indices \(> 3 + 3\). Hence, the sequence starts with a \(b\). Next, the index is decreased by \(N_a = 3\), yielding a new index with value 2. Applying the same procedure, \(c\) is obtained as the second character, yielding a new index with value 1. Continuing in this way we find that the sequence with index 5 is \(bcca\).

It is obvious that the enumerative decoding process can be applied twice, once for the runs of ones and once for the runs of zeroes. In this way, all different run permuted sequences can be generated in their run-length representations. To generate the corresponding binary sequences it suffices to apply the run-length decoding algorithm. The structure of such a sequence generator is depicted in Figure 6.8.
6.4 Conclusions

In this chapter two construction methods for binary sequences were described and their properties investigated.

The first construction method is based on the Ziv-Lempel source coding algorithm and uses a switch controlled FSR to generate sequences. This method is difficult to analyze in great detail. The described method uses fixed length copying only. However, it is claimed here that, if one allows variable copying lengths and chooses the various parameters involved appropriately, then the sequences generated are suitable for cryptographic purposes. It must be emphasized, however, that there are still many unanswered questions and a substantial research effort seems necessary to provide the answers.

We also concluded that many known construction methods can be regarded as improvements and extensions of our method. In view of this fact, one might draw the conclusion that feedback functions, as well as feedforward functions, containing memory, which are applied to simple sequence generators, greatly enhance their complexity performance.

Our second construction method is based on run-length coding and enumerative coding techniques. Starting with a DeBruijn sequence of given order we construct an entire class of sequences by permuting the runs of ones and zeroes. The class of sequences contains all DeBruijn sequences of the given order, but many other sequences as well, all satisfying Golomb's first and second randomness postulates.

It was also demonstrated that all the sequences in the class can be
generated by enumerative decoding techniques. In this way many more classes of sequences can be constructed, e.g. if the starting sequence is a *Maximum Length Linear FSR sequence* or a *Legendre sequence* ([Golo 67, pg. 47]). It should also be mentioned that if one agrees on having a smaller class of sequences, with sequences possibly occurring more than once, there may be other alternatives to generate the run permuted sequences.
Chapter 7

Information Theory of Shift Register Sequences

7.1 Introduction

In 1949 Claude Shannon, the founder of information theory, published his paper "Communication Theory of Secrecy Systems", [Shan 49]. This paper was a first inception to a mathematical theory of secrecy systems. Famous are Shannon's notions of key equivocation and unicity distance. It lasted many years before there was any substantial progress made in this area, see [Hell 77,Tilb 85].

The main problem in applying information theory to practical cipher systems was given by Shannon himself:

"...we have calculated the equivocation characteristic for a simple substitution applied to a two-letter language. This is about the simplest type of cipher and the simplest language structure possible, yet already the formulas are so involved as to be nearly useless. What are we to do with cases of practical interest, say the involved transformations of a fractional transposition system applied to English with its extremely complex statistical structure?"

This could be an explanation for the small number of papers in this particular area of information theory. Clearly, many practical cipher systems have been described, which are too complex to be analyzed in Shannon's way.
In this chapter an attempt is made to come to an information theoretic analysis of cipher systems based on sequences generated by means of feedback shiftregisters. We are concerned here with so-called uncertainty profiles, which are plots showing the uncertainty e.g. in the next bit of a fixed but in a random phase started sequence of characters from some alphabet. Three of these uncertainty profiles are defined and their properties shown.

In this entire chapter we concentrate on the amount of information in bits (in general a-ary information units) which is revealed by observing successive characters of some fixed finite sequence. An interpretation in error probabilities has been given, see [Tilb 85], and could also be applied here. It seems, however, that there is no real reason in favor of applying the latter method to the subject of this chapter, as it is merely a matter of interpretation and not of obtaining sharper bounds or more meaningful results.

In many streamcipher systems keystream generators are used to obtain very long random-looking sequences, called pseudo-random or pseudo-noise sequences, which are used to add to the message stream to obtain the cryptogram. Judging the quality of such sequences has been an interesting problem having received much attention in the past. Well-known in this respect are works of Golomb [Golo 67], Lempel and Ziv [Lemp 76] and Rueppel [Ruep 84]. Golombs randomness postulates, for instance, define almost exclusively sequences obtained from a maximum-length linear feedback shiftregister and his postulates may be relaxed quite a bit as was shown by Wanders [Wand 88]. Also, Chapter 3 of this thesis may be regarded as a contribution in this area.

### 7.2 The Character Uncertainty Profile

Consider the following model of a keystream generator (KG). A given sequence \( z = (a_0, a_1, \ldots, a_{p-1})^\infty \), \( a_i \in A \), where \( A \) is some finite alphabet, is started in an arbitrary unknown phase to emit successive characters. So the first character is \( a_{0+i} \), followed by \( a_{i+1} \), etc. where the indices are taken modulo \( p \). The initial phase \( i \) is a uniformly distributed random variable with probability \( Pr[i = i] = 1/p \), for all \( i \in [0, p-1] \). It can be seen that the KG forms a stationary and ergodic source, which we will denote by \( \langle z \rangle \). Following Gallager [Gall 68], the average amount of information which is contained in the first \( L \) characters of the keystream, i.e. the entropy of the keystream source \( \langle z \rangle \) is denoted by \( H(Z_1, Z_2, \ldots, Z_L) \). Using the chainrule
of entropy, [Gall 68, pg. 22], this can be written as:

\[ H(Z_1, \ldots, Z_L) = H(Z_1) + H(Z_2|Z_1) + H(Z_3|Z_1, Z_2) + \cdots + H(Z_L|Z_1, \ldots, Z_{L-1}). \]

(7.1)

In (7.1) the conditional entropy \( H(Z_i|Z_{i-1}, \ldots, Z_{i+1}) \) plays a fundamental role; it is in fact the uncertainty in the next character of the keystream, given all the previous characters. Consequently, we have the following definition:

**Definition 7.1** The sequence of successive conditional entropies 
\[ H(Z_i|Z_{i-1}, \ldots, Z_{i+1}), \quad N \geq 1, \]

of the stationary ergodic keystream source \( \langle z \rangle \) is called the Character Uncertainty Profile (CUP) of \( \langle z \rangle \), denoted by 
\[ \text{CUP}(\langle z \rangle). \]

The entropy of \( \langle z \rangle \) is of importance for the quality of a cipher system. Consider Figure 7.1, which shows a streamcipher system in which the cryptotext characters \( c_i \) are related to the plaintext characters \( p_i \) and the keystream characters \( z_i \) by: \( c_i = p_i \oplus z_i \) and \( p_i = c_i \oplus z_i \), where \( \oplus \) and \( \ominus \) denote some additive mixing function and its inverse respectively. The average amount of information about the plaintext a cryptanalyst is able to extract from the cryptotext is equal to the mutual information between \( P \) and \( C \), 
\[ I(P_1, \ldots, P_L; C_1, \ldots, C_L). \]

This mutual information, however, satisfies:

\[ I(P_1, \ldots, P_L; C_1, \ldots, C_L) = H(C_1, \ldots, C_L) - H(C_1, \ldots, C_L|P_1, \ldots, P_L) \]

\[ = H(C_1, \ldots, C_L) - H(Z_1, \ldots, Z_L). \]

(7.2)

This result follows from the relation between \( c_i \), \( p_i \) and \( z_i \), and the fact that the character alphabets all have the same cardinalities.
If the keystream characters are independent and identically distributed random variables with uniform probability distribution, so will the ciphertext characters be and hence \( I(P; C) = 0 \). In this case the CUP will show a "straight line", i.e. the CUP will be a sequence of constants (their values depending on the base of the logarithm in the entropy).

The CUP has some interesting properties:

**Proposition 7.1**

1. The CUP is monotonically non-increasing with increasing number of observed characters.

2. The CUP is zero after \( c(z) \) characters, where \( c(z) \) denotes the maximum order complexity of the sequence \( z \).

3. If \( z \) is a DeBruijn sequence of order \( n \), its CUP is constant for the first \( n - 1 \) values and zero thereafter.

**Proof.**

1. This follows from the fact that the CUP is a sequence of conditional entropies \( H(Z_N|Z_1, \ldots, Z_{N-1}) \) and conditioning on more variables cannot increase the entropy, see [Gall 68, pg. 25].

2. As every \( c(z) \) long subsequence occurs only once as a property of maximum order complexity, the succeeding character is uniquely determined and hence the uncertainty about this character is zero.

3. In a DeBruijn sequence ([DeBr 46]) of order \( n \) each \( n \)-tuple occurs exactly once. Therefore each \( (n-j) \)-tuple, \( 0 < j < n \), occurs exactly \( a^j \) times, where \( a \) is the cardinality of the character alphabet. Consequently, the succeeding characters are uniformly distributed, yielding an entropy \( H(Z_L|Z_1, \ldots, Z_{L-1}) = -\log_a \frac{1}{a} = 1 \), for \( L \leq n \). As in this case \( c(z) = n \), it follows from 2 that the uncertainty is zero after \( n \) characters.

From Proposition 7.1-3 and equation (7.2) it follows that DeBruijn sequences are optimal in the sense that they maintain \( I(P_L; C_L) = 0 \) until \( L = n + 1 \).
Example 7.1 Consider the binary DeBruijn sequence of order 4, $z = (111010110000100)\infty$, and the sequence $y = (111010100001100)\infty$ of period 16 and maximum order complexity $c(y) = 5$. By straightforward calculation one can obtain the CUP's of both sequences:

$$CUP(z) = (1, 1, 1, 1, 0, 0, \ldots, 0)$$

$$CUP(y) = (1, 1, 1, 1, 1, 0, \ldots, 0)$$

These sequences are used to 'encrypt' a binary symmetric memoryless source (i.e. a source that emits 0's and 1's with equal probability independent of the preceding bits) by means of modulo 2 addition. In this case the mutual information $I(P; C)$ for both sequences denoted by $I((z))$ and $I((y))$ increases with $L$ as follows:

$$I((z)) = 0, 0, 0, 0, 1, 2, 3, 4, \ldots$$

$$I((y)) = 0, 0, 0, 1, 2, 3, 4, \ldots$$

7.3 The Phase Uncertainty Profile

Consider again the keystream generator model introduced in the previous section. The random initial phase in which the sequence is started may be regarded as (part of) the secret key $K$ of the cipher system; a situation often described in literature, see e.g. [Sieg 83]. A well designed keystream generator will maintain as high as possible an uncertainty in this key $K$, even after having emitted many keystream characters. This is what Shannon calls the "acid test", see [Shan 49, pg. 710]. The latter situation arises in a known or chosen plaintext attack. Here, the interest lies in the equivocation of key, given successive keystream characters, $H(K|Z_1, \ldots, Z_L)$, or the average amount of information about the key a cryptanalyst is able to extract from the keystream, $I(K; Z_1, \ldots, Z_L) = H(K) - H(K|Z_1, \ldots, Z_L)$. Evidently, the ideal situation here is that $I(K; Z_1, \ldots, Z_L) = 0$ for $L$ as large as possible.

To cope with the situation described above, we introduce the following definition:

Definition 7.2 Let $K$ denote the initial random phase of the stationary ergodic keystream source $(z)$. The sequence of successive conditional entropies $H(K|Z_1, \ldots, Z_L)$, $L \geq 0$, is called the Phase Uncertainty Profile (PUP) of $(z)$, denoted by $PUP((z))$. 
The role of the PUP can be seen, loosely speaking, as a quality measure for the cryptanalysis of the secret key rather than of the plaintext as with the CUP.

The following properties apply to the PUP:

**Proposition 7.2**

1. The PUP decreases monotonically with increasing number of observed characters until \( c(z) \) characters have been observed, after which the PUP is zero.

2. The equality \( H(K|Z_1,\ldots,Z_{c-1}) = H(Z_c|Z_1,\ldots,Z_{c-1}) \) holds, where \( c = c(z) \).

3. The PUP of a DeBruijn sequence of order \( n \), decreases linearly from \( n \) to 0 for values of \( L \) from 1 to \( n \).

**Proof.**

1. Whenever \( c(z) \) characters have been observed, it follows from the definition of maximum order complexity that this \( c(z) \)-long subsequence is unique and hence the initial phase of the sequence is uniquely determined.

Now consider the entropy of the joint ensemble \( (K, Z_1,\ldots,Z_N) \). We have:

\[
H(K, Z_1,\ldots,Z_N) = H(K|Z_1,\ldots,Z_N) + H(Z_1,\ldots,Z_N)
\]

and also:

\[
H(K, Z_1,\ldots,Z_N) = H(K).
\]

Here, (7.4) follows from the fact that the initial phase uniquely determines the sequence of outcomes of \( Z_1,\ldots,Z_N \). Combining (7.3) and (7.4) yields:

\[
H(K|Z_1,\ldots,Z_N) = H(K) - H(Z_1,\ldots,Z_N)
\]

\[
= H(K) - \sum_{j=1}^{N} H(Z_j|Z_1,\ldots,Z_{j-1}).
\]

The monotonicity now is implied by the fact that the conditional entropies \( H(Z_j|Z_1,\ldots,Z_{j-1}) \) are positive for \( j < c(z) \), according to Proposition 7.1.
2. Clearly, $H(Z_1, \ldots, Z_c) = H(K)$; combining this with equation (7.5) we obtain:

$$H(K|Z_1, \ldots, Z_{c-1}) = H(K) - H(Z_1, \ldots, Z_{c-1})$$

$$= H(Z_1, \ldots, Z_c) - H(Z_1, \ldots, Z_{c-1})$$

$$= H(Z_c|Z_1, \ldots, Z_{c-1}).$$

3. This is an immediate consequence of Proposition 7.1-3, combined with equation (7.5).

Equation (7.5) shows that the CUP and the PUP are closely related. However, it is the behaviour of the PUP from which it can be concluded that it is not wise to use the initial phase of a keystream generator as a key in a streamcipher system.

### 7.4 Obtaining High Equivocation of Key

In this section we consider a new model of a keystream generator, different from the one used in the previous two sections. Given is an ensemble $Z$ of periodic sequences $z_i$, $1 \leq i \leq E$, with periods $p_i$ and maximum order complexities $c_i$. Governed by some secret key a sequence is chosen, for all time, at random with probability $1/E$ and then the sequence is started in a random phase with probability $1/p_i$, emitting successive characters.

Unlike the previous KG model the source described here is not ergodic (it is, however, stationary). This source is a typical example of a composite source as described e.g. by Viterbi and Omura [Vite 79, pp. 527–534]. For stationary non-ergodic sources the notion of entropy may be applied as well, providing that the averages taken are ensemble averages. So analogously to Definition 7.1 the CUP of an ensemble of sequences can be defined, which obviously does not have all the same properties as mentioned in Proposition 7.1. However, we will not do so, but rather focus on the equivocation of key.

Let $s$ denote the random variable indicating the sequence that is chosen; hence $s$ takes on values from $[1, E]$. Then the following elementary equations are evident:

$$H(S, Z_1, \ldots, Z_N) = H(S) + H(Z_1, \ldots, Z_N|S)$$

$$= H(Z_1, \ldots, Z_N) + H(S|Z_1, \ldots, Z_N).$$
Unlike equation (7.3) we have that $H(S, Z_1, \ldots, Z_N) \geq H(S)$, as the choice of sequence in general does not uniquely determine the sequence of outcomes $Z_1, \ldots, Z_N$.

Here, the conditional entropy $H(S|Z_1, \ldots, Z_N)$ plays the same role as $H(K|Z_1, \ldots, Z_N)$ in the previous section. Consequently the following definition is relevant:

**Definition 7.3** Let $S$ denote the initial random choice of the particular sequence from the ensemble $Z$. The sequence of successive conditional entropies $H(S|Z_1, \ldots, Z_N)$, $N \geq 0$, of the stationary non-ergodic keystream source $(z)$ is called the Sequence Uncertainty Profile (SUP) of $(z)$, denoted by SUP$(z)$.

Suppose SUP$(z)$ is zero after exactly $\ell$ characters. Consequently, there must be exactly one sequence in the ensemble $Z$ which has the $\ell$ characters as a subsequence, but for less than $\ell$ characters there exist two or more sequences having the same subsequence. This suggests a complexity definition analogous to that of Chapter 3, but for an ensemble of sequences rather than for a single sequence.

**Definition 7.4** The maximum order complexity of an ensemble $Z$ of sequences $z_i = (\alpha_{0,i}, \alpha_{1,i}, \ldots, \alpha_{p,i-1,i})^\infty$, denoted by $c_\ell(Z)$, is defined as the shortest length $\ell$, such that any subsequence of length $\ell$ uniquely identifies each of the sequences $z_i \in Z$.

There does not seem to exist a simple relation between the complexities of individual sequences and the complexity of an ensemble, as may be concluded from the next example.

**Example 7.2**

1. Let $z_1 = (01)^\infty$ and $z_2 = (1000000)^\infty$. Clearly $c(z_1) = 1$ and $c(z_2) = 6$, whereas $c_\ell(Z) = 4$.

2. Let $z_1 = (01111)^\infty$ and $z_2 = (10000)^\infty$. In this case $c(z_1) = 4$ and $c(z_2) = 4$, whereas $c_\ell(Z) = 3$.

The Sequence Uncertainty Profile has the following properties:

**Proposition 7.3**

1. The SUP is monotonically non-increasing with increasing number of observed characters.
2. The SUP is zero after \( c_\ell (Z) \) characters have been observed.

3. The SUP of an ensemble containing all DeBruijn sequences of order \( n \) is constant for the first \( n \) values, monotonically decreasing thereafter and zero after \( a^n - 2 \) characters at minimum, where \( a \) denotes the alphabet cardinality.

Proof.

1. Clearly \( H(S|Z_1, \ldots ,Z_N) \leq H(S|Z_1, \ldots ,Z_{N-1}) \). The equality can indeed occur, viz. if and only if all the sequences in the ensemble have the same number of all different subsequences of certain length.

2. By Definition 7.4.

3. The proof is in three parts:

   i) As was mentioned with the proof of Proposition 7.1-3, in every DeBruijn sequence of order \( n \) each \((n - j)-tuple, 0 \leq j < n\), occurs exactly \( a^j \) times. Consequently, \( H(S|Z_1, \ldots ,Z_N) = H(S), 0 < N \leq n\).

   ii) Also, as a direct consequence of the sequence order, a fraction \( a^{-j} \) of all possible \((n + j)\)-tuples occurs in each DeBruijn sequence. However, not all sequences can have the same distribution of \((n + j)\)-tuples, with increasing \( j \), as all the sequences in the ensemble are evidently different. This fact implies that \( H(S|Z_1, \ldots ,Z_N) < H(S|Z_1, \ldots ,Z_{N-1}) \), for \( n < N \leq c_\ell (Z) \).

   iii) It is shown that \( a^n - 2 \leq c_\ell (Z) \leq a^n - 1 \). The upper-bound is trivial: two DeBruijn sequences always differ in at least two characters, as all characters occur equally often in each sequence. The lower-bound is obtained by the two DeBruijn sequences, of order \( n > 2 \), having the following
states sequences:

\[
\begin{array}{c}
S_{i-1} \\
(\alpha, \beta, \ldots, \beta, \gamma) \quad (\alpha, \beta, \ldots, \beta, \beta) \\
\downarrow \\
(\beta, \ldots, \beta, \gamma, \beta) \quad (\beta, \ldots, \beta, \beta, \beta) \\
\downarrow \\
\vdots \\
(\gamma, \beta, \ldots, \beta, \beta) \quad (\beta, \ldots, \beta, \beta, \epsilon) \\
\downarrow \\
(\beta, \beta, \ldots, \beta, \beta) \quad (\beta, \epsilon, \beta, \ldots, \beta) \\
\downarrow \\
(\beta, \beta, \ldots, \beta, \delta) \quad (\epsilon, \beta, \ldots, \beta, \delta) \\
\downarrow \\
S_{i+n+1}
\end{array}
\]

These subsequences of states can indeed occur in a pair of DeBruijn sequences with characters from any alphabet, as all states are distinct and they are the shortest paths containing these states that yield DeBruijn sequences, i.e. they include the all-\(\beta\) state.

One can prove that for binary DeBruijn sequences Proposition 7.3-3 holds exactly, i.e. in this case \(c_\epsilon (Z) = 2^n - 2\). The proof can be accomplished by using the non-singular feedback function property of the FSR that generates the DeBruijn sequence, see also Chapter 5, and showing that this feedback function necessarily must be linear if \(c_\epsilon (Z) = 2^n - 1\).

From the behaviour of the SUP it may be concluded that the KG model presented in this section is much in favor of the model used in the previous sections. In fact it is demonstrated here that one should have the ability to choose from a large family of sequences by means of a secret key. Moreover, small differences between individual sequences within the family guarantee that the equivocation of key does not decrease rapidly with the number of successive keystream characters which have become known.

**Example 7.3** The ensemble of all 16 binary DeBruijn sequences of order
Figure 7.2: CUP and SUP of the ensemble of all DeBruijn sequences of order 4.

4 has CUP and SUP as depicted in Figure 7.2, with values:

CUP: 1 1 1 1 0.70 0.66 0.62 0.55 0.49 0.35 0.33 0.15 0.11 0.03 0 0
SUP: 4 4 4 4 4 3.30 2.63 2.02 1.47 0.97 0.62 0.29 0.14 0.03 0 0

7.5 Conclusions

In this chapter streamcipher systems were treated in an information theoretic manner. To this end, a probabilistic source model of a keystream generator was introduced. This keystream source uses a fixed periodic sequence \( z \) which is started in a random secret phase, the key of the cipher system, to emit successive characters. To judge the cryptographic quality of this keystream source, so-called uncertainty profiles were defined, viz. the character- and the phase uncertainty profiles, abbreviated CUP and PUP respectively. The CUP shows the behaviour of the uncertainty in the next keystream character, given all previous ones. The PUP, in fact, is the progression of the equivocation of key, given a number of past successive keystream characters. It was shown that both uncertainty profiles reach zero after exactly \( c(z) \) characters, the maximum order complexity of the sequence, which is definitely a drawback of this type of cipher system.

Alternatively a more complicated source model of a keystream generator was introduced. In this model the keystream source first chooses at random
one sequence from an ensemble of fixed periodic sequences, this random choice being the secret key of the cipher system. Then the chosen sequence is started in a random phase to emit successive characters. For this source model the sequence uncertainty profile (SUP) was introduced. This SUP has the same role as the PUP for the first model, i.e. key equivocation progression. Also, this SUP was shown to give rise to a new complexity definition for an ensemble of sequences rather than for individual sequences as in Chapter 3.

From the behaviour of the various uncertainty profiles it clearly follows that the second model is superior to the first one. The results of this chapter justify the use of a keystream generator as depicted in Figure 7.3, in which either $K_1$ or $K_2$ or both are (part of) the secret key, but the initial phase is not.

Figure 7.3: Key multiplicity with a FSR based keystream generator.
Bibliography


Bibliography


Appendix

Binary DeBruijn Sequences of Order 4 and 5

In this appendix all binary DeBruijn sequences of order 4 and 5, of which there are 16 and 2048 respectively, are listed. The sequences are listed in tables as follows:

\[ \text{NR, HEXSEQ, RECNR, COMNR, HEXFF, FWT, FCMP, AVGDL}\]

in which we have:

**NR:** The table entry of the sequence.

**HEXSEQ:** The DeBruijn sequence in hexadecimal notation, e.g. \(1111011001010000 = \text{F650} \).

**RECNR:** The table entry of the reciprocal sequence.

**COMNR:** The table entry of the complementary sequence.

**HEXFF:** Hexadecimal notation of the naturally ordered \(g\)-part of the feedback function \(f(x_0, x_1, \ldots, x_n) = x_0 + g(x_1, \ldots, x_n)\), e.g. \(1 + x_1 + x_1x_2 + x_2x_3 + x_1x_2x_3 \equiv 11010011 \equiv \text{D3} \).

**FWT:** The number of product terms in the algebraic normal form of the \(g\)-function.

**FCMP:** The feedback function complexity according to the weight vector \((122323342 \cdots)\), as explained in Chapter 4.

**AVGDLP:** The average deviation from the perfect linear complexity profile as given by equation (5.4).
If RECNR and COMNR are identical they are *italicized*, denoting that reciprocating and complementing yield essentially the same sequence.

For order 4 the results are:

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>F650</td>
<td>15</td>
<td>16</td>
<td>C1</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>F614</td>
<td>11</td>
<td>4</td>
<td>D3</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>F590</td>
<td>16</td>
<td>15</td>
<td>BF</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>F584</td>
<td>12</td>
<td>2</td>
<td>E5</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>F4C2</td>
<td>7</td>
<td>8</td>
<td>C7</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>F4B0</td>
<td>13</td>
<td>14</td>
<td>83</td>
<td>3</td>
</tr>
<tr>
<td>7</td>
<td>F432</td>
<td>5</td>
<td>9</td>
<td>9D</td>
<td>5</td>
</tr>
<tr>
<td>8</td>
<td>F42C</td>
<td>9</td>
<td>5</td>
<td>D9</td>
<td>5</td>
</tr>
<tr>
<td>9</td>
<td>F342</td>
<td>8</td>
<td>7</td>
<td>CB</td>
<td>5</td>
</tr>
<tr>
<td>10</td>
<td>F2D0</td>
<td>14</td>
<td>13</td>
<td>8F</td>
<td>5</td>
</tr>
<tr>
<td>11</td>
<td>F286</td>
<td>2</td>
<td>12</td>
<td>9B</td>
<td>5</td>
</tr>
<tr>
<td>12</td>
<td>F21A</td>
<td>4</td>
<td>11</td>
<td>AD</td>
<td>5</td>
</tr>
<tr>
<td>13</td>
<td>F0D2</td>
<td>6</td>
<td>10</td>
<td>91</td>
<td>3</td>
</tr>
<tr>
<td>14</td>
<td>F0B4</td>
<td>10</td>
<td>6</td>
<td>D5</td>
<td>5</td>
</tr>
<tr>
<td>15</td>
<td>F0A6</td>
<td>1</td>
<td>3</td>
<td>89</td>
<td>3</td>
</tr>
<tr>
<td>16</td>
<td>F09A</td>
<td>3</td>
<td>1</td>
<td>F7</td>
<td>7</td>
</tr>
</tbody>
</table>

From the table it can clearly be seen that the sequences 3 and 16, which are maximum-length linear FSR sequences extended with a zero at the correct position, are the worst in many respects.

The results for order 5 are given on the following pages. As can be seen the sequence with number 575 has one of the 'simplest' feedback functions, viz. \( f(x_0, x_1, x_2, x_3, x_4) = 1 + x_0 + x_2 + x_1x_2x_3x_4 \), however, the average deviation from the perfect linear complexity profile amounts to 155.81. In this respect sequence 17 is much better (AVGDLP = 46.81), whereas its feedback function has almost the same complexity.
DeBruijn Sequences of Order 4 & 5
Appendix
### DeBruijn Sequences of Order 4 & 5

<table>
<thead>
<tr>
<th>Year</th>
<th>Sequence</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>2017</td>
<td>FA8258E6</td>
<td>302 504 E7E3 12 34 40.06</td>
</tr>
<tr>
<td>2018</td>
<td>FA8259C6</td>
<td>294 576 F6FD 14 40 32.81</td>
</tr>
<tr>
<td>2019</td>
<td>FA825B8C</td>
<td>286 668 F4FF 14 41 56.94</td>
</tr>
<tr>
<td>2020</td>
<td>FA825B98</td>
<td>278 708 E5E1 10 27 45.50</td>
</tr>
<tr>
<td>2021</td>
<td>FA8960E6</td>
<td>298 760 B249 8 22 45.06</td>
</tr>
<tr>
<td>2022</td>
<td>FA896706</td>
<td>290 1819 A357 10 32 35.25</td>
</tr>
<tr>
<td>2023</td>
<td>FA896E0C</td>
<td>282 650 A155 8 25 44.25</td>
</tr>
<tr>
<td>2024</td>
<td>FA896E60</td>
<td>274 1514 B04B 8 23 38.63</td>
</tr>
<tr>
<td>2025</td>
<td>FA9058E6</td>
<td>301 968 D8DF 14 42 45.00</td>
</tr>
<tr>
<td>2026</td>
<td>FA9059C6</td>
<td>293 1186 CAC1 8 20 36.13</td>
</tr>
<tr>
<td>2027</td>
<td>FA905B8C</td>
<td>285 1524 C8C3 8 21 41.31</td>
</tr>
<tr>
<td>2028</td>
<td>FA905B98</td>
<td>277 1651 D9DD 12 35 46.38</td>
</tr>
<tr>
<td>2029</td>
<td>FA9160E6</td>
<td>297 818 EE75 10 30 46.13</td>
</tr>
<tr>
<td>2030</td>
<td>FA916706</td>
<td>289 1880 9F6B 12 36 40.00</td>
</tr>
<tr>
<td>2031</td>
<td>FA916E0C</td>
<td>281 702 9D69 10 29 35.63</td>
</tr>
<tr>
<td>2032</td>
<td>FA916E60</td>
<td>273 1612 8C77 10 31 44.56</td>
</tr>
<tr>
<td>2033</td>
<td>FA9058DC</td>
<td>16 696 F2C5 10 27 48.75</td>
</tr>
<tr>
<td>2034</td>
<td>FA9059B8</td>
<td>8 734 E3DB 12 35 38.44</td>
</tr>
<tr>
<td>2035</td>
<td>FA2560DC</td>
<td>12 1272 A76F 12 37 68.75</td>
</tr>
<tr>
<td>2036</td>
<td>FA2566E0</td>
<td>4 942 B671 10 29 51.63</td>
</tr>
<tr>
<td>2037</td>
<td>FA4158DC</td>
<td>15 1566 CEF9 12 33 44.00</td>
</tr>
<tr>
<td>2038</td>
<td>FA4159B8</td>
<td>7 1736 DFE7 14 41 45.19</td>
</tr>
<tr>
<td>2039</td>
<td>FA4560DC</td>
<td>11 1314 9B63 10 31 49.81</td>
</tr>
<tr>
<td>2040</td>
<td>FA4566E0</td>
<td>3 1114 8A4D 8 23 37.25</td>
</tr>
<tr>
<td>2041</td>
<td>FA8258DC</td>
<td>14 528 F7F3 14 41 37.81</td>
</tr>
<tr>
<td>2042</td>
<td>FA8259B8</td>
<td>6 600 E6ED 12 33 50.88</td>
</tr>
<tr>
<td>2043</td>
<td>FA8960DC</td>
<td>10 1369 A259 8 23 38.50</td>
</tr>
<tr>
<td>2044</td>
<td>FA8966E0</td>
<td>2 1494 B347 10 31 38.06</td>
</tr>
<tr>
<td>2045</td>
<td>FA9058DC</td>
<td>13 1016 CBCF 12 35 81.38</td>
</tr>
<tr>
<td>2046</td>
<td>FA9059B8</td>
<td>5 1296 DAD1 10 27 36.38</td>
</tr>
<tr>
<td>2047</td>
<td>FA9160DC</td>
<td>9 1398 9E65 10 29 38.94</td>
</tr>
<tr>
<td>2048</td>
<td>FA9166E0</td>
<td>1 1588 8F7B 12 37 39.13</td>
</tr>
</tbody>
</table>
Curriculum Vitae

Cees J.A. Jansen was born in Roosendaal, The Netherlands, on November 19, 1953. He received the Ingenieur degree in electrical engineering from Eindhoven University of Technology, Eindhoven, The Netherlands, in 1980. He joined Philips Usfa B.V. in 1980, where he has been engaged in research and development of cryptographic systems and equipment. His activities in this area have included the design of key management and streamcipher systems, as well as the application of source coding and digital signal processing techniques. He is a member of the IEEE Information Theory Group and the International Association for Cryptologic Research.
Samenvatting

Onderzoek Naar Niet-Lineaire Stroomvercijfer Systemen: Constructie en Evaluatie Methoden

Vercijfer systemen worden gebruikt voor het realiseren van een scala aan beveiligingsdoeleinden. Stroomvercijfer systemen vormen een belangrijke klasse van vercijfer systemen en worden zeer veel toegepast in de praktijk. Van oudsher echter, bestaat er slechts weinig theorie over deze praktische stroomvercijfer systemen, welke doorgaans berusten op niet-lineaire systeem aspecten. Dit proefschrift behandelt enkele van deze niet-lineaire aspecten. Nieuwe constructie methoden en nieuwe, krachtige evaluatie methoden worden voorgesteld.

Indien het gaat over de bescherming tegen de gevolgen van manipulatie van de cryptotext, blijkt de mixer het belangrijkste onderdeel van een stroomvercijfer systeem te zijn. Efficiënte niet-lineaire mixers gebaseerd op het gebruik van JK-flipflops worden voorgesteld. Verschillende nieuwe gebruiksmodes voor blokvercijfer systemen worden behandeld. De OFBNLF mode, welke de beste bescherming biedt, maakt gebruik van eenvoudige niet-lineaire bewerkingen en is daardoor zeer effectief en eenvoudig te implementeren.

Veel aandacht is besteed aan het probleem van het vinden van het aller kortste (mogelijk niet-lineair) teruggekoppelde schuifregister, dat een gegeven reeks van characters, afkomstig uit een willekeurig eindig alfabet, kan opwekken. Voor dit doel is er een nieuwe complexiteits maat geïntroduceerd, genaamd de maximale orde complexiteit van reeksen. Een nieuw stuk theory over het niet-lineair teruggekoppelde schuifregister is ontwikkeld, handelend over de elementaire complexiteits eigenschappen van getransponeerde en reciproke reeksen, reeksen beschouwd over machts-
Samenvatting

alfabetten, duale reeksen en terugkoppel functies van het maximale orde teruggekoppelde schuifregister equivalent. Het niet bestaan van zelf-complementaire en zelf-reciproke DeBruijn reeksen is aangetoond. Het algoritme van Blumer blijkt een zeer krachtig gereedschap voor het bepalen van zowel het maximale orde complexiteitsprofiel van reeksen, als ook hun periode, lineair in de benodigde tijd en geheugen. De complexiteit van toevalsreeksen is theoretisch benaderd. Hierbij is het spronggedrag van het maximale orde complexiteitsprofiel volledig geanalyseerd. Het blijkt dat, in tegenstelling tot het lineaire complexiteitsprofiel, wat altijd een sprong maakt naar een waarde van $l - c_1$, indien er een sprong optreedt bij uitbreiding van de reeks met een karakter, het maximale orde complexiteitsprofiel ook naar lagere waarden kan springen. Aangetoond is ook dat het probleem van het tellen van reeksen van zekere lengte en complexiteit gelijk is aan het moeilijke probleem van het tellen van onvolledige paden in een DeBruijn graaf van zekere orde. Statistische modellen van de maximale orde complexiteit worden voorgesteld en gebruikt om de verwachte complexiteit van toevalsreeksen te voorspellen. Het resultaat hiervan is dat de verwachte maximale orde complexiteit toeneemt met tweemaal de logaritme uit de reekslengte. Experimentele resultaten blijken de geldigheid van de modellen te ondersteunen.

Onder andere voor de analyse van niet-lineaire terugkoppel functies van teruggekoppelde schuifregisters wordt een nieuwe kijk op de algebraïsche normaal vorm gepresenteerd. Gebruik makend van de zogenaamde natuurlijke ordening van de produkttermen in deze normaal vorm wordt de algebraïsche normaal vorm transformatie (ANFT) gedefinieerd. Deze ANFT is een snelle transformatie, welke eenvoudig in hardware of software geïmplementeerd kan worden. De ANFT kan toegepast worden op binaire toevalsfuncties ten behoeve van de complexiteitsanalyse van deze functies.

Een algoritme voor het samenvoegen van cykels in de binaire DeBruijn graaf wordt gepresenteerd. Dit algoritme gedraagt zich feitelijk als een toegevoegde niet-lineaire terugkoppeling functie, welke bij elk niet-singulier teruggekoppeld schuifregister gebruikt kan worden. Door gebruik te maken van een teruggekoppelde schuifregister met een geschikte lineaire terugkoppel functie, kan het algoritme $O(2^{2n}/\log^2 n)$ binaire DeBruijn reeksen opwekken, waarbij slechts $3n$ bits opslag en maximaal $4n$ schuifoperaties benodigd zijn om elke bit uit de reeks op te wekken.

Het schuifregister met geschakelde terugkoppel functie wordt voorgesteld als een manier om binaire reeksen op te wekken, welke goede complexiteits eigenschappen bezitten. De relevante parameters kunnen zodanig gekozen
worden dat de opgewekte reeksen een maximale orde complexiteit hebben ongeveer gelijk aan het dubbele van de logaritme uit hun lengte. Statistische eigenschappen wijzen op de mogelijkheid van een correlatie aanval op deze generator, echter met een hoge berekeningscomplexiteit. Vele bekende constructie methoden voor reeks generatoren blijken te kunnen worden beschouwd als uitbreidingen van de beschreven methode.

Een tweede benadering voor het verkrijgen van reeksen met goede complexiteit eigenschappen bestaat uit het construeren van zogenaamde run gepermuteerde reeksen. Uitgaande van een DeBruijn reeks van gegeven orde wordt een klasse van reeksen geconstrueerd door de runs van enen en nullen te permuteren. Aangetoond is dat deze klasse alle DeBruijn reeksen van de gegeven orde bevat, alsmede een groot aantal andere reeksen van dezelfde lengte, welke allen voldoen aan de eerste twee randomness postulaten van Golomb. Alle reeksen in de klasse kunnen eenvoudig met behulp van enumeratieve koderings schema's opgewekt worden.

Stroomvercijfer systemen worden op een informatie theoretische manier beschouwd door de sleutelgenerator als een stochastische bron te modeleren. Deze sleutelbron maakt gebruik van een vaste periodieke reeks, welke in een geheime willekeurige fase gestart wordt, om opeenvolgende sleutelkarakters af te geven. Teneinde de cryptografische kwaliteit van deze sleutelbron te kunnen beoordelen, worden zogenaamde onzekerheidsprofielen geïntroduceerd. Deze onzekerheidsprofielen tonen duidelijk de zwakte van de sleutelgenerator aan.

Een ingewikkelder bron model, namelijk een samengestelde bron, van een sleutelgenerator wordt voorgesteld. Dit tweede model blijkt cryptografisch veel beter te zijn dan het eerste model. De conclusie is dat sleutelgeneratoren in staat dienen te zijn om een ensemble van reeksen op te wekken. Uit dit ensemble wordt dan een reeks bij toeval gekozen onder invloed van een geheime sleutel. De initiële fase van de reeks dient echter bij toeval gekozen te worden, maar moet niet als geheime sleutel informatie beschouwd worden.
X
De producent van cryptografische apparatuur besteedt doorgaans veel meer aandacht aan systeem-, hardware- en software aspecten dan aan cryptografische.
Bij de consument lijkt dit vaak juist tegenovergesteld te zijn.

XI
Het verschil tussen wat men research dan wel voorontwikkeling noemt bij grote ondernemingen is niet zozeer gelegen in het wetenschappelijk niveau van de ontplooide activiteiten, maar meer in de plaats binnen de organisatie waar deze uitgevoerd worden.

XII
De ruimte welke op voorgedrukte documenten, zoals belasting formulieren, gelaten wordt voor het invullen van gegevens nodigt niet uit tot volledigheid en duidelijkheid.

Stellingen
behorende bij het proefschrift van C.J.A. Jansen

Delft, 18 april 1989
Het is onmogelijk bij een ternair teruggekoppeld schuifregister dat de toestand (⋯22) opgevolgd of voorafgegaan wordt door de toestand (⋯11).


Het is onjuist te stellen dat het aantal binaire reeksen van periode $2^n$ met de normale run verdeling iets groter (“slightly larger”) is dan het aantal $2^{2^n-1-n}$ van DeBruijn reeksen van orde $n$, voor alle $n > 2$.


Het standaardiseren van de klassieke gebruiksmodes van blokvercijfer algoritmen, ECB, CBC, CFB en OFB, is een onnodige beperking voor de praktijk en getuigt niet van veel creativiteit.


De relatie tussen kenmerken van een reeks zoals complexiteit en statistiek en de reeks zelf lijkt niet op eenvoudige en consistente wijze te geven. Zo is de periodieke binaire reeks (1110100001101100)° de enige perfecte gebalanceerde binaire reeks van periode 19, zijn complement en reciproke niet meegerekend. De maximale orde complexiteit van deze reeks is 5 en derhalve bestaan er 8192 terugkoppel functies waarmee deze reeks opgewekt kan worden. Hieronder zijn er slechts 2 niet-singulier, waarvan de minst complexe functie 9 produkttermen heeft. Er bestaan ook slechts 2 functies welke aanleiding geven tot de meeste vertakkingspunten, waarvan de minst complexe 12 produkttermen heeft. De minst complexe functie heeft slechts 7 produkttermen. Deze reeks kent geen duale reeks.

Voor ieder priemgetal $p$ bestaan er precies $p(p-1)$ periodieke binaire reeksen met periode $p$ en maximale orde complexiteit $p - 1$.

Het is onjuist om eenvoudigweg te stellen dat DeBruijn reeksen op echte random reeksen lijken.


Bij het ontwerp van cryptografische algoritmen kan met succes gebruik gemaakt worden van bronkoderingstechnieken.

Vanuit informatie theoretisch oogpunt beschouwd zijn veel van de in de literatuur gepubliceerde stroomvercijfer algoritmen van weinig nut voor de cryptografie.

Uit resultaten op het gebied van de moleculaire biologie is af te leiden dat de DNA reeks van het λ virus opgevat kan worden als een reeks over $GF(4)$, opgewekt door een teruggekoppeld schuifregister van 16 secties.
