DNA Computing: Foundations and Implications *

Lila Kari   Shinnosuke Seki   Petr Sosík

1 Introduction

DNA computing is an area of natural computing based on the idea that molecular biology processes can be used to perform arithmetic and logic operations on information encoded as DNA strands. The aim of this review is two-fold. First, we introduce the fundamentals of DNA computing, including basics of DNA structure and bio-operations, and two historically important DNA computing experiments. Secondly, we describe some of the ways in which DNA computing research has impacted fields in theoretical computer science.

The first part of this review outlines basic molecular biology notions necessary for understanding DNA computing (Section 2), recounts the first experimental demonstration of DNA computing (Section 3), as well as the milestone wet laboratory experiment that first demonstrated the potential of DNA computing to outperform the computational ability of an unaided human (Section 4).

The second part of the review describes how the properties of DNA-based information, and in particular the Watson-Crick complementarity of DNA single strands, have influenced areas of theoretical computer science such as formal language theory, coding theory, automata theory and combinatorics on words. More precisely, Section 5 summarizes notions and results in formal language theory and coding theory arising from the problem of design of optimal encodings for DNA computing experiments: Section 5.1 describes the problem of DNA encodings design, Section 5.2 consists of an analysis of intramolecular bonds (bonds within a given DNA strand), Section 5.3 defines and characterizes languages that avoid certain undesirable intermolecular bonds (bonds between two or more DNA strands), and Section 5.4 investigates languages whose words avoid even imperfect bindings between their constituent strands.

Section 6 contains another, vectorial, representation of DNA strands and two computational models based on this representation: sticker systems and Watson-Crick automata. After a brief description of the representation of DNA partial double strands as two-line vectors, and of the sticking operation that combines them, Section 6.1 describes basic sticker systems, sticker systems

*This work was supported by The Natural Sciences and Engineering Council of Canada Discovery Grant and Canada Research Chair Award to L.K.

with complex structures (Section 6.1.1), and observable sticker systems (Section 6.1.2). Section 6.2 investigates the accepting counterpart of the generative sticker systems devices: Watson-Crick automata and their properties.

Section 7 describes the influence that properties of DNA-based information have had on research in combinatorics on words, by enumerating several natural generalizations of classical concepts of combinatorics of words: pseudo-palindromes, pseudo-periodicity, Watson-Crick conjugate and commutative words, involutively bordered words, pseudoknot bordered words. In addition, this section outlines natural extensions in this context of two of the most fundamental results in combinatorics of words, namely Fine and Wilf's theorem and Lyndon-Schützenberger result.

Section 8 presents general thoughts on DNA-based information, bioinformation and biocomputation.

2 A Computer Scientist's Guide to *DNA*

In this section we give a brief description of the basic molecular biology notions of DNA structure and DNA-based bio-operations used in DNA computing. For further details, the reader is referred to [107], [35, 12, 49, 78].

A **DNA (deoxyribonucleic acid)** molecule is a linear polymer. The monomer units of DNA are nucleotides (abbreviated *nt*), and the polymer is known as a "polynucleotide." There are four different kinds of nucleotides found in DNA, each consisting of a nitrogenous *base* (**A**denine, A, **C**ytosine, C, **G**uanine, G, or **T**hymine, T), and a sugar-phosphate unit. The abbreviation N stands for any nucleotide. The bases are relatively flat molecules and can be divided into purine bases (adenine and guanine) that have two carbon-containing rings in their structure, and smaller pyrimidine bases (cytosine and thymine) that have one carbon-containing ring in their structure.

A sugar-phosphate unit consists of a deoxyribose sugar and one to three phosphate groups. Together, the base and the sugar comprise a *nucleoside.* The sugar-phosphate units are linked together by strong *covalent bonds* to form the backbone of the DNA single strand (ssDNA). A DNA strand consisting of n nucleotides is sometimes called an n-mer. An *oligonucleotide* is a short DNA single strand, with twenty or fewer nucleotides. Since nucleotides may differ only by their bases, a DNA strand can be viewed as simply a word over the four-letter alphabet $\{A, C, G, T\}$.

A DNA single strand has an orientation, with one end known as the 5' end, and the other end known as the 3' end, based on their chemical properties. By convention, a word over the DNA alphabet represents the corresponding DNA single strand in the 5' to 3' orientation, that is, the word ACGTCGACTAC stands for the DNA single strand 5'-ACGTCGACTAC-3'. A crucial feature of DNA single strands is their **W***atson-Cric***k** *(WK) complementarity*: A (a purine) is complementary to T (a pyrimidine), and G (a purine) is complementary to C (a pyrimidine). Two complementary DNA single strands with opposite orientation bind to each other by weak *hydrogen bonds* between their individual bases as follows: A binds with T through two hydrogen bonds, while G binds with C through three hydrogen bonds. Thus, two Watson-Crick complementary single strands form a stable DNA double strand (dsDNA) resembling a helical ladder, with the two backbones at the outside, and the bases paired by hydrogen bonding and stacked on each other, on the inside. For example, the DNA single strand 5'-ACGTCGACTAC- 3' will bind to the DNA single strand 5'-GTAGTCGACGT-3' to form the 11 base-pair long (abbreviated as $11bp$) double strand

$$\begin{matrix} 5' - ACGTCGACTAC - 3' \\ 3' - TGCAGCTGATG - 5' \end{matrix}.$$

Figure 1 schematically illustrates this DNA double strand, omitting the double helix structure, for clarity. If v denotes a DNA single strand over the alphabet $\{A, C, G, T\}$, then by $\overrightarrow{v}$ we will denote its Watson-Crick complement.

Another molecule that can be used for computation is *RNA*, *ribonucleic acid*. RNA is similar to DNA, but differs from it in three main aspects: RNA is usually single-stranded while DNA is usually double-stranded, RNA nucleotides contain the sugar ribose, while DNA nucleotides contain the sugar deoxyribose, and in RNA the base Uracil, U, substitutes for thymine, T, which is present in DNA.

The *genome* of an organism is the totality of its genetic information encoded in DNA. It consists of *chromosomes*, which, in turn, consist of genes. A *gene* is a segment of DNA that holds the information encoding a coherent set of functions necessary to build and maintain cells, and pass genetic traits to offspring. A gene comprises *coding* subsequences (*exons*) that determine what the gene does, and *non-coding* subsequences (*introns*). When a gene is active, the coding and non-coding sequences are copied in a process called *transcription*, producing an RNA copy of the gene's information. This piece of RNA can then direct the *translation* of the catenation of the coding sequences of this gene into *proteins* via the *genetic code*. The genetic code maps each 3-letter RNA segment (called *codon*) into an amino acid. Several designated triplets, the start codon (AUG), and the stop (UAA, UAG, UGA) codons, signal the initiation, respectively the termination of a translation. There are twenty different standard amino acids. Some of them are encoded by one codon, while others are encoded by several "synonymous" codons. A protein is a sequence over the 20-letter alphabet of amino acids. Proteins are essential parts of organisms and participate in every process within cells having, e.g., catalytical, structural or mechanical functions.

To encode, for example, English text using DNA, one can choose an encoding scheme mapping the Latin alphabet onto strings over $\{A, C, G, T\}$, and proceed to synthesize the obtained information-encoding strings as DNA single strands. In a hypothetical example, one could encode the letters of the English alphabet as $\mathbf{A} \rightarrow ACA$, $\mathbf{B} \rightarrow ACCA$, $\mathbf{C} \rightarrow ACCCA$, $\mathbf{D} \rightarrow AC^4A$, etc., wherein the ith letter of the alphabet is represented by AC^iA, i.e., a single strand of DNA consisting of i repetitions of C flanked by one A at the beginning and another A at the end. Under this encoding, the text "To be or not to be" becomes the

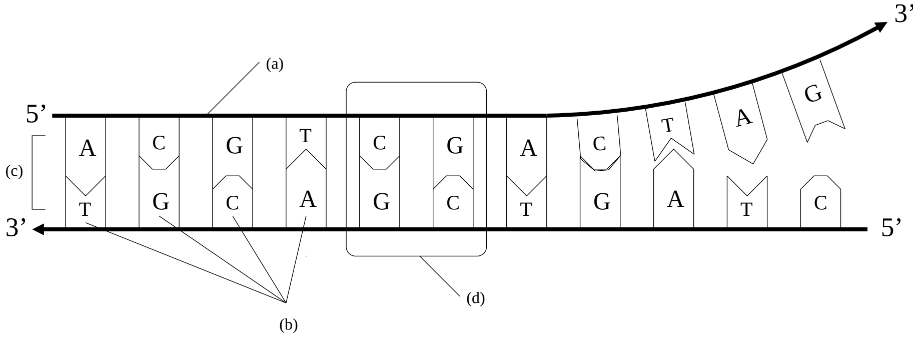

Figure 1: DNA structure: (a) DNA's sugar-phosphate backbone, (b) DNA bases, (c) Watson-Crick complementarity between bases A and T of two DNA single strands of opposite orientation, (d) Watson-Crick complementarity between bases C and G of two DNA single strands of opposite orientation.

DNA single strand represented by

$$\overbrace{AC^{20}A}^{T}\overbrace{AC^{15}A}^{O}\overbrace{AC^{2}A}^{B}\overbrace{AC^{5}A}^{E}\overbrace{AC^{15}A}^{O}\overbrace{AC^{18}A}^{R}\overbrace{AC^{14}A}^{N}\overbrace{AC^{15}A}^{O}\overbrace{AC^{20}A}^{T}\overbrace{AC^{20}A}^{T}\overbrace{AC^{15}A}^{O}\overbrace{AC^{2}A}^{B}\overbrace{AC^{5}A}^{E}$$

that can be readily synthesized.

Indeed, **DNA synthesis** is the most basic **bio-operation** used in DNA computing. DNA solid-state synthesis is based on a method by which the initial nucleotide is bound to a solid support, and successive nucleotides are added step-by-step, from the 3' to the 5' direction, in a reactant solution (Figure 2).

While the above encoding example is purely hypothetical, DNA strands of lengths of up to 100 nucleotides, can be readily synthesized using fully automated *DNA synthesizers*. The result is a small test tube containing a tiny, dry, white mass of indefinite shape containing a homogeneous population of DNA strands that may contain 10^{18} identical molecules of DNA. In bigger quantities, dry DNA resembles tangled, matted white thread.

Using this or other DNA synthesis methods, one can envisage encoding any kind of information as DNA strands. There are several reasons to consider such a DNA-based memory as an alternative to all the currently available implementations of memories. The first is the extraordinary information-encoding density that can be achieved by using DNA strands. According to [96], 1 gram of DNA, which contains 2.1×10^{21} DNA nucleotides, can store approximately 4.2×10^{21} bits. Thus, DNA has the potential of storing data on the order of 10^{10} more compactly than conventional storage technologies. In addition, the robustness of DNA data ensures the maintenance of the archived information over extensive periods of time [8, 19, 104].

For the purposes of DNA computing, after encoding the input data of a problem on DNA strands, DNA bio-operations can be utilized for computations, see [62, 94, 4]. The bio-operations most commonly used to control DNA computations and DNA robotic operations are described below.

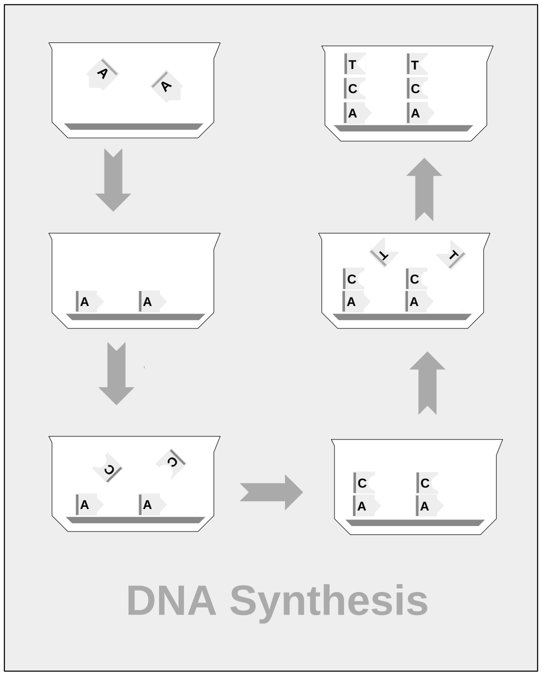

Figure 2: DNA solid-state synthesis. The initial nucleotide is bound to a solid support, and successive nucleotides are added step-by-step, from the 3' to the 5 direction, in a reactant solution. From [62].

DNA single strands with opposite orientation will join together to form a double helix in a process based on the Watson-Crick complementarity and called **base-pairing** (**annealing**, **hybridization**, **renaturation**), illustrated in Figure 3. The reverse process – a double-stranded helix coming apart to yield its two constituent single strands – is called **melting** or **denaturation**. As the name suggests, melting is achieved by raising the temperature, and annealing by lowering it. Each DNA double-strand denaturates at a specific temperature, called its *melting temperature*. The melting temperature is defined as the temperature at which half of the DNA strands are double-stranded, and half are single-stranded. It depends on both the length of the DNA sequence, and its specific base-composition, [101].

Cutting DNA double strands at specific sites can be accomplished with the aid of specific enzymes, called *restriction enzymes (restriction endonucleases)*. Each restriction enzyme recognizes a specific short sequence of DNA, known as a *restriction site*. Any double-stranded DNA that contains the restriction site within its sequence is cut by the enzyme at that location, according to a specific pattern. Depending on the enzyme, the cutting operation leaves either two "blunt-ended" DNA double strands or, more often, two DNA strands that are double-stranded but have single-stranded overhangs known as "sticky-ends", Figure 4. Hundreds of restriction enzymes are now known, and a large number are commercially available. They usually recognize sites ranging in size from

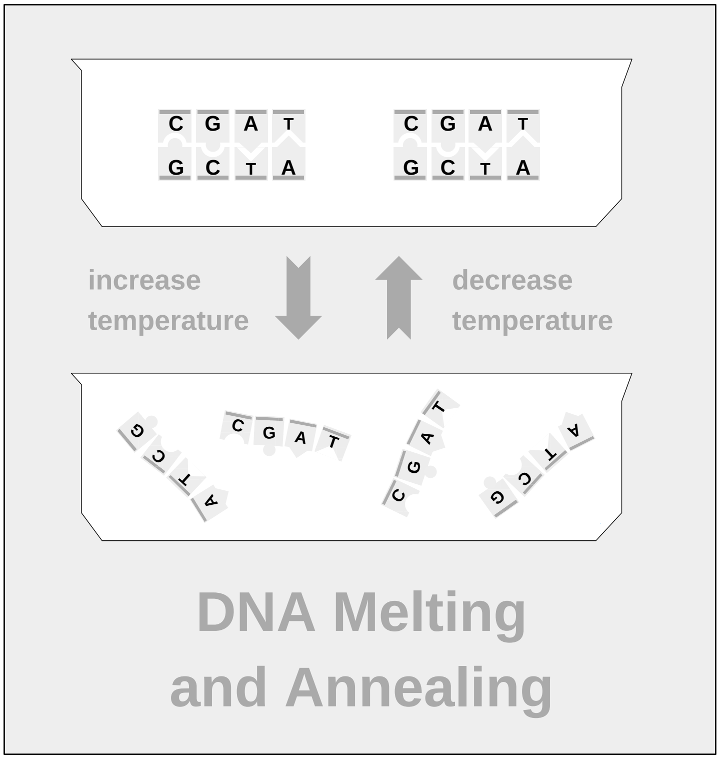

Figure 3: Melting (separating DNA double strands into their constituent single strands), and annealing (the reformation of double-stranded DNA from thermally denatured DNA). Raising the temperature causes a DNA double strand to "melt" into its constituent Watson-Crick complementary single strands. Decreasing the temperature has the opposite effect: two DNA single strands that are Watson-Crick complementary will bind to each other to form a DNA double strand. From [62].

4bp to 8bp, the recognition sites are often pseudopalindromic (see Section 7 for a definition of pseudopalindrome).

Another enzyme, called *DNA ligase*, can repair breaks in a double-stranded DNA backbone, and can covalently rejoin annealed complementary ends in the reverse of a restriction enzyme reaction, to create new DNA molecules. The process of thus pasting together compatible DNA strands is called **ligation**.

Separation of DNA strands by size is possible by using a technique called *gel electrophoresis*. The DNA molecules, which are negatively charged, are placed in "wells" situated at one side of an agarose or polyacrylamide gel. Then an electric current is applied to the gel, with the negative pole at the side with the wells, and the positive pole at the opposite side. The DNA molecules will be drawn towards the positive pole, with the larger molecules travelling more slowly through the gel. After a period, the molecules will spread out into distinct bands according to size, Figure 5. The gel method at constant electric field is capable of separating by size DNA molecules as long as 50,000 base pairs,

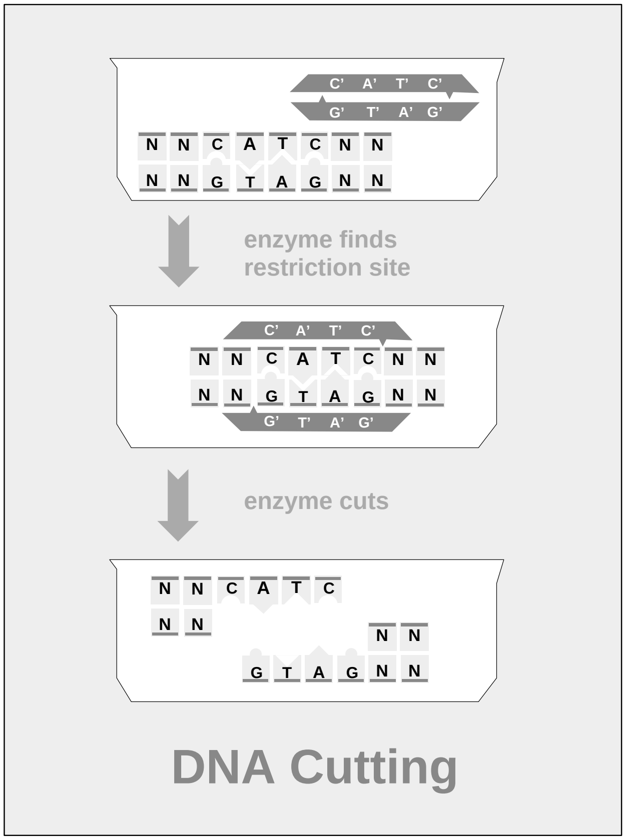

Figure 4: DNA cutting (digestion) by a restriction enzyme. A hypothetical restriction enzyme (dark grey) recognizes the restriction site $CATC$ on a DNA double-strand (light grey), and cuts the two backbones of the DNA strand, as shown. Under most conditions, Watson-Crick complementarity of four base-pairs is not sufficient to keep the strands together, and the DNA molecule separates into two fragments. The result of the digestion is thus a $CATC - 3'$ sticky end overhang, and a $3' - GTAG$ sticky end overhang. The reverse process of ligation can restore the original strand by bringing together strands with compatible sticky ends by means of Watson-Crick base-pairing, and using the enzyme ligase that repairs the backbone breaks (nicks) that had been introduced by the restriction endonuclease. From [62].

with a resolution of better than 1% the size of the DNA.

Extraction of DNA single strands that contain a target sequence, v, from a heterogeneous solution of DNA single strands, can be accomplished by *affinity purification*, Figure 6. A DNA probe is a single-stranded DNA molecule used in laboratory experiments to detect the presence of a complementary sequence among a mixture of other singled-stranded DNA molecules.

The extraction process begins by synthesizing probes, i.e., strands $\overrightarrow{v}$, Watson-Crick complementary to v, and attaching them to a solid support, e.g.,

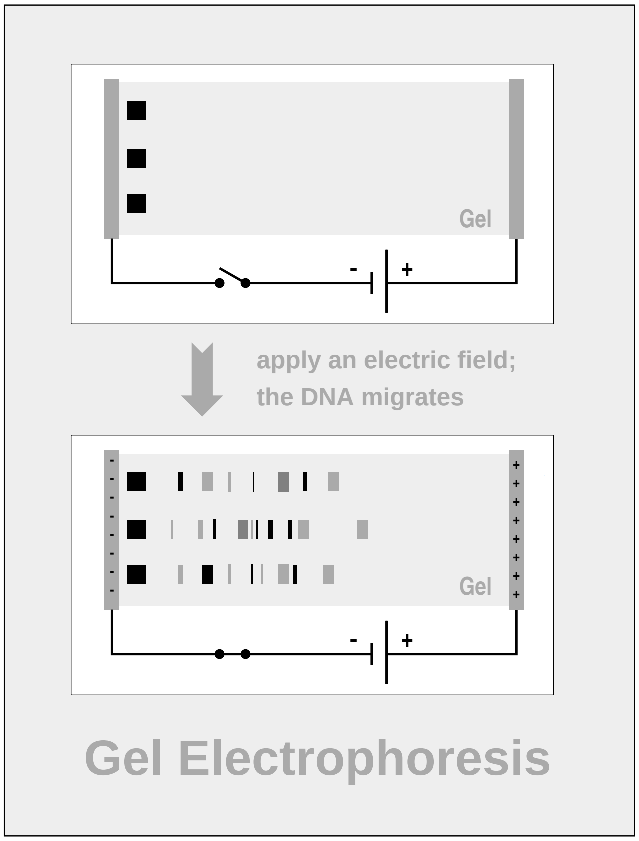

Figure 5: Separation of DNA strands by size (length) by using gel electrophoresis. The DNA samples are placed in *wells* (slots) near one end of the gel slab. The power supply is switched on and the DNA, which is highly negatively charged, is allowed to migrate towards the positive electrode (right side of the gel) in separate *lanes* or *tracks*. DNA fragments will move through the gel at a rate which is dependent on their size and shape. After a while, the DNA molecules spread out into distinct bands and the use of a control "ladder" that contains DNA strands of incremental lengths, allows the determination of the lengths of the DNA molecules in the samples. From [62].

magnetic beads. Then, the heterogeneous solution of DNA strands is passed over the beads. Those strands containing v are "detected" by becoming annealed to $\overleftarrow{v}$, and are thus retained. Strands not containing v pass through without being retained. The solid medium, e.g., magnetic beads, can then be removed from the mixture, washed and the target DNA molecules released from the entrapment.

DNA replication is accomplished by a process called *Polymerase Chain Reaction*, or *PCR*, that involves the *DNA polymerase* enzyme, Figure 7.

The PCR replication reaction requires a guiding DNA single strand called *template*, and an oligonucleotide called *primer*, that is annealed to the template. The primer is required to initiate the synthesis reaction of the polymerase en-

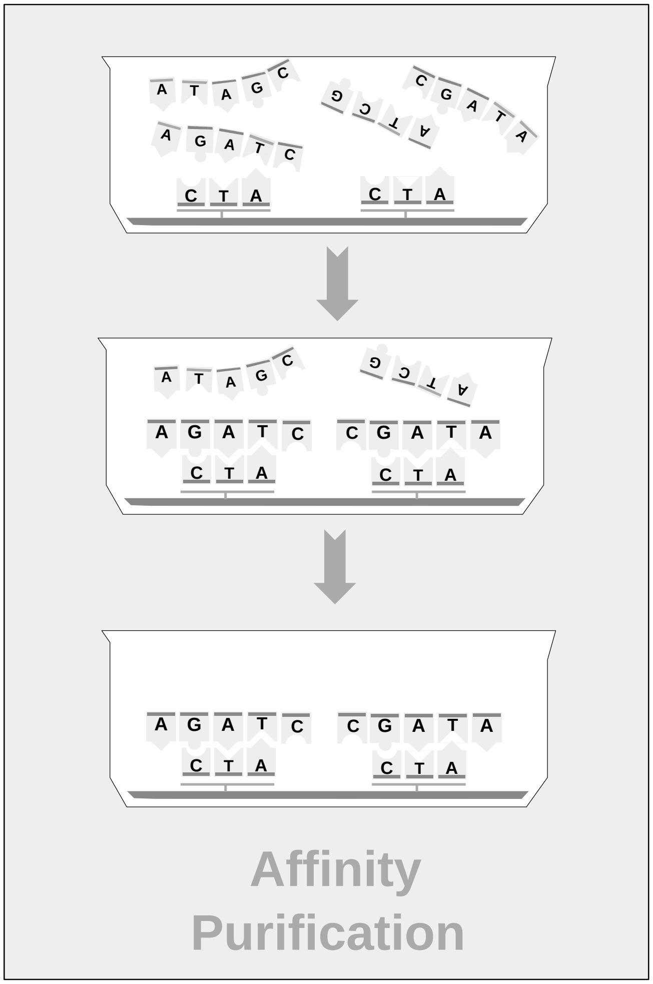

Figure 6: Extraction of strands that contain a target sequence, $v = 5' - GAT - 3'$, by affinity purification. (The 3nt pattern GAT is for illustration purposes only, in practice a longer DNA subsequence would be needed for the process of extraction to work.) The Watson-Crick complement of v, namely $3' - CTA - 5'$, is attached to a solid support, e.g., magnetic beads. The heterogeneous solution of DNA strands is poured over. The DNA strands that contain the target sequence v as a subsequence will attach to the complement of the target by virtue of Watson-Crick complementarity. Washing off the solution will result in retaining the beads with the attached DNA strands containing the target sequence. From [62].

zyme. The DNA polymerase enzyme catalyzes DNA synthesis by successively adding nucleotides to one end of the primer. The primer is thus extended at its 3' end, in the direction 5' to 3' only, until the desired strand is obtained that starts with the primer and is complementary to the template. (Note that DNA chemical synthesis and enzymatic DNA replication proceed in opposite directions, namely $3' - > 5'$ and $5' - > 3'$, respectively).

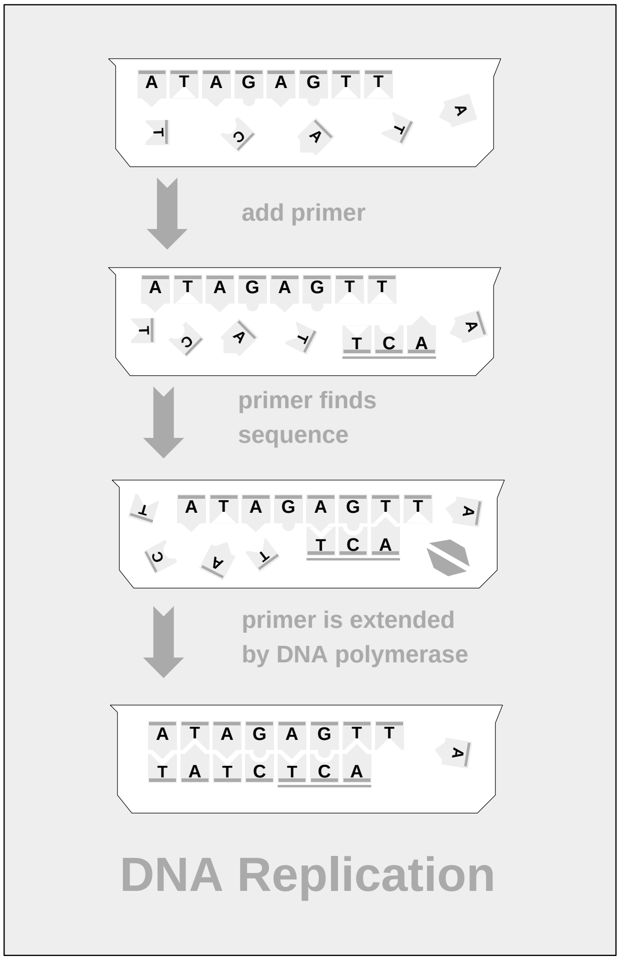

Figure 7: DNA strand replication using one primer and the enzyme DNA polymerase. Given a template DNA strand $5' - ATAGAGTT - 3'$ to replicate, first a primer $3' - TCA - 5'$ (a short DNA sequence, usually 10-15nt long, that is complementary to a portion of the template) is added to the solution. DNA polymerase (dark grey) extends the primer in the 5' to 3' direction, until the template becomes fully double stranded. Repeating the process by heating the solution to denaturate the double-strands and then cooling it to allow annealing of the primer, will thus lead to producing many copies of the portion of the complement of the template strand that starts with the primer. The idea is used in Polymerase Chain Reaction (PCR) which uses two primers, DNA polymerase, dNTPs and thermal cycling to produce exponentially many copies of the subsequence of a template strand that is flanked by the primers. dNTP, *deoxyribonucleoside triphosphate*, stands for any of the nucleotides d**A**TP, d**T**TP, d**C**TP, d**G**TP. Each nucleotide consists of a base, plus sugar (which together form a *nucleoside*), plus triphosphate. dNTPs are the building blocks from which the DNA polymerases synthesizes a new DNA strand. From [62].

If two primers are used, the result is the exponential multiplication of the subsequence of the template strand that is flanked by the two primers, in a process called *amplification*, schematically explained below. For the purpose of this explanation, if x is a string of letters over the DNA alphabet $\{A, C, G, T\}$, then $\overline{x}$ will denote its simple complement, e.g., $\overline{AACCTTGG} = TTGGAACC$.

Let us assume now that one desires to amplify the subsequence between x and y from the DNA double strand $\begin{array}{c} 5' - \alpha x \beta y \delta - 3' \\ 3' - \bar{\alpha}\bar{x}\bar{\beta}\bar{y}\bar{\delta} - 5' \end{array}$, where $\alpha, x, \beta, y, \delta$ are DNA segments. Then one uses as primers the strand x and the Watson-Crick complement $\overrightarrow{y}$ of y. After heating the solution and thus melting the double-stranded DNA into its two constituent strands, the solution is cooled and the Watson-Crick complement of y anneals to the "top" strand, while x anneals to the "bottom" strand. The polymerase enzyme extends the 3' ends of both primers into the 5' to 3' direction, producing partially double-stranded molecules $\begin{array}{l} 5' - \alpha x \beta y \delta - 3' \\ 3' - \bar{\alpha}\bar{x}\bar{\beta}\bar{y} \quad - 5' \end{array}$ and $\begin{array}{l} 5' - \quad x \beta y \delta - 3' \\ 3' - \bar{\alpha}\bar{x}\bar{\beta}\bar{y}\bar{\delta} - 5' \end{array}$. In a similar fashion, the next heating-cooling cycle will result in the production of the additional strands $5' - x\beta y - 3'$ and $3' - \bar{x}\bar{\beta}\bar{y} - 5'$. These strands are Watson-Crick complementary and will, from now on, be produced in excess of the other strands, since both are replicated during each cycle. At the end, an order of 2^n copies of the desired subsequences flanked by x and y will be present in solution, where n is the number of the heating-cooling cycles, typically 20 to 30.

A biocomputation consists of a succession of *bio-operations*, [28], such as the ones described in this section. The DNA strands representing the output of the biocomputation can then be **sequenced** (**read out**) using an automated sequencer. One sequencing method uses special chemically modified nucleotides (dideoxyribonucleoside triphosphates - ddNTPs), that act as "chain terminators" during PCR, as follows. A sequencing primer is annealed to the DNA template that we wish to read. A DNA polymerase then extends the primer. The extension reaction is split into four tubes, each containing a different chain terminator nucleotide, mixed with standard nucleotides. For example, tube C would contain chemically modified C (dd**C**TP), as well as the standard nucleotides (d**A**TP, d**G**TP, d**C**TP and d**T**TP). Extension of the primer by the polymerase then produces all prefixes ending in G of the complement of the original strand. A separation of these strands by length using gel electrophoresis allows the determination of the position of all Gs (complements of Cs). Combining the results obtained in this way for all four nucleotides allows the reconstruction of the original sequence.

Some of the novel features of DNA-encoded information and bio-operations have been used for the first time for computational purposes in the breakthrough proof-of-principle experiment in DNA computing reported by Adleman in 1994.

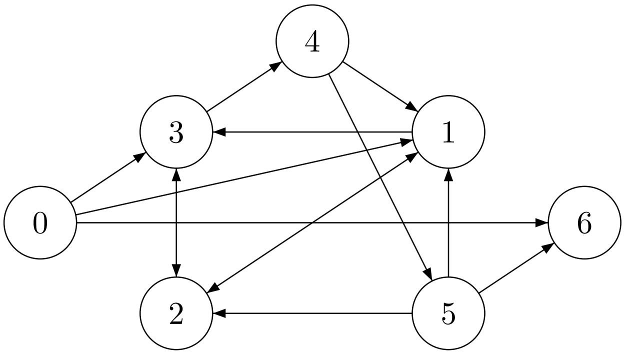

Figure 8: The seven-vertex instance of the Hamiltonian Path Problem solved by Adleman, who used solely molecular biology tools to obtain the answer. This was the first ever experimental evidence that DNA computing is possible [1].

3 The First DNA Computing Experiment

The practical possibilities of encoding information in a DNA sequence and of performing simple bio-operations were used by Leonard Adleman in 1994, [1], to perform the first experimental DNA computation that solved a 7-vertex instance of an NP-complete problem, namely the directed Hamiltonian Path Problem (HPP).

A directed graph G with designated vertices v_{start} and v_{end} is said to have a Hamiltonian path if and only if there exists a sequence of compatible directed edges $e_1, e_2, \ldots, e_z$ (that is, a directed path) that begins at v_{start}, ends at v_{end} and enters every other vertex exactly once.

The following (nondeterministic) algorithm solves the problem:

Input. A directed graph G with n vertices and designated vertices v_{start} and v_{end}.
Step 1. Generate random paths through the graph.
Step 2. Keep only those paths that begin with v_{start} and end with v_{end}.
Step 3. Keep only those paths that enter exactly n vertices.
Step 4. Keep only those paths that enter all of the vertices of the graph at least once.
Output. If any paths remain, output "YES"; otherwise output "NO".

Below we describe Adleman's bio-algorithm that solves the 7-vertex instance of the Hamiltonian Path Problem illustrated in Figure 8, where $v_{start} = 0$ and $v_{end} = 6$.

To encode the *input* to the problem, that is the vertices and directed edges of the graph, each vertex of the graph was encoded into a carefully chosen 20-mer single strand of DNA. Then, for each oriented edge of the graph, a DNA sequence was designed and synthesized, consisting of the second half of the sequence encoding the source vertex and the first half of the sequence encoding the target vertex. Exceptions were the edges that started with v_{start}, and the ones that ended in v_{end}, for which the DNA encoding consisted of the full sequence of the source vertex followed by the first half of the target vertex, respectively the second half of the source vertex followed by the full sequence of the target vertex.

For example, the DNA sequences for the vertex 3 and the oriented edges 2 → 3 and 3 → 4 were encoded respectively as:
$O_3 \quad = 5' - GCTATTCGAGCTTAAAGCTA - 3'$
$O_{2\to 3} = 5' - GTATATCCGAGCTATTCGAG - 3'$
$O_{3\to 4} = 5' - CTTAAAGCTAGGCTAGGTAC - 3'.$

To implement *Step 1*, one mixed together in a test tube multiple copies of each of the encodings of the Watson-Crick complements $\overrightarrow{O_i}$ of all the vertices, together with the encodings for all the directed edges, for a ligation reaction. The complements of the vertices served as splints and brought together sequences associated to compatible edges. For example, the edges $O_{2\to 3}$ and $O_{3\to 4}$ were brought together by the Watson-Crick complement $\overrightarrow{O_3}$ as follows:

$$\begin{array}{c} 5' - \overbrace{GTATATCCGAGCTATTCGAG}^{\text{edge } 2\to 3}\overbrace{CTTAAAGCTAGGCTAGGTAC}^{\text{edge } 3\to 4} - 3' \\ {\scriptstyle 3'-}\underbrace{CGATAAGCTCGAATTTCGAT}_{\text{complement of vertex } 3}{\scriptstyle -5'} \end{array}$$

Hence, the Watson-Crick complementarity and the combined ligation reaction resulted in the formation of DNA molecules encoding random paths through the graph. Out of these, the next steps had to find and discard the paths that were not Hamiltonian.

To implement *Step 2* (keep only paths that start with v_{start} and end with v_{end}) the product of *Step 1* was amplified by PCR with primers O_0 and $\overrightarrow{O_6}$. Thus, only those molecules encoding paths that begin with the start vertex 0 and end with the end vertex 6 were amplified.

For implementing *Step 3* (keep only paths of the correct length) gel electrophoresis was used, allowing separation of DNA strands by length. Since any Hamiltonian path, if it exists, has to pass through all the 7 vertices of the graph, only DNA double strands of length $7 \times 20 = 140$ were retained.

Step 4 (keep only paths that pass through each vertex at least once) was accomplished by iteratively using affinity purification. After generating single-stranded DNA from the double-stranded product of the preceding step, one attached a sequence $\overrightarrow{O_i}$ to magnetic beads, and the heterogeneous solution of "candidate paths" was passed over the beads. Those strands containing O_i annealed to the complementary sequence and were hence retained. These strands represented paths that pass through the vertex i. This process was repeated successively with $\overrightarrow{O_1}$, $\overrightarrow{O_2}$, $\overrightarrow{O_3}$, $\overrightarrow{O_4}$ and $\overrightarrow{O_5}$.

To obtain the *Output* (are there any paths left?) the presence of a molecule encoding a Hamiltonian path was checked. This was done by amplifying the result of *Step 4* by PCR with primers O_0 and $\overrightarrow{O_6}$, and then reading out the DNA sequence of the amplified molecules. Note that the final PCR was not required for computational purposes: It was employed firstly to make sure that, after the several filtering steps, the amount of DNA was above the detection threshold, and secondly to verify the correctness of the answer.

The entire computation required approximately seven days of wet lab work, and was carried out in approximately one fiftieth of a teaspoon of solution [2]. It was the first proof-of-concept experiment that DNA computation was possible. From a practical point of view, Adleman's approach had both advantages and disadvantages. On one hand, *Step 1*, which is the cause of the time-complexity exponential blow-up in the classical electronic implementation of the algorithm, took only one time-step in Adleman's bio-algorithm. On the other hand, the amount of space needed for the generation of the full solution space grows exponentially relative to the problem size. Indeed, a scaled-up input of size $n = 200$ for the Hamiltonian Path Problem would require, in this brute force approach, an amount of DNA whose weight would be greater than that of the Earth [53].

The construction of a DNA pool that contains the full-solution space has been avoided in subsequent bio-algorithms proposed for solving HPP. For example, in [88] a bio-algorithm was proposed that stepwise generated only the possible paths. In this approach, all paths were stepwise extended from v_{start} to v_{end} as follows. After letting many molecules representing v_{start} attach to a surface, $n+1$ repetitions of the following step found the Hamiltonian Path: "Extend each path by one adjacent vertex; Remove paths which contain the same vertex twice". In addition to being space and time efficient (each extension step could be as quick as 30 minutes), this bio-algorithm avoided the laborious step of separation of strands by length.

4 Beyond Unaided Human Computation

We present another significant milestone in DNA computing research, the first experiment that demonstrated that DNA computing devices can exceed the computational power of an unaided human. Indeed, in 2002 an experiment was reported, [11], that solved a 20-variable instance of the NP-complete 3-SAT problem, wherein the answer to the problem was found after an exhaustive search of more than 1 million (2^{20}) possible solution candidates.

The input to a 3-SAT problem is a Boolean formula in three-conjunctive-normal-form (3-CNF), i.e., a Boolean formula that consists of disjunctions of conjunctive clauses, where each conjunctive clause is the conjunction of at most three literals (a literal is either a Boolean variable or its negation). This formula is called *satisfiable* if there exists a truth value assignment to its variables that satisfies it, i.e., that makes the whole formula true. Thus, the output to the 3-SAT problem is "yes" if such a satisfying truth value assignment exists, and "no" otherwise.

The input formula for this experiment was the 20-variable, 24-clause, 3-CNF formula:

$\Phi = (\overline{x_3} \vee \overline{x_{16}} \vee x_{18}) \wedge (x_5 \vee x_{12} \vee \overline{x_9}) \wedge (\overline{x_{13}} \vee \overline{x_2} \vee x_{20}) \wedge (x_{12} \vee \overline{x_9} \vee \overline{x_5}) \wedge (x_{19} \vee \overline{x_4} \vee x_6) \wedge (x_9 \vee x_{12} \vee \overline{x_5}) \wedge (\overline{x_1} \vee x_4 \vee \overline{x_{11}}) \wedge (x_{13} \vee \overline{x_2} \vee \overline{x_{19}}) \wedge (x_5 \vee x_{17} \vee x_9) \wedge (x_{15} \vee x_9 \vee \overline{x_{17}}) \wedge (\overline{x_5} \vee \overline{x_9} \vee \overline{x_{12}}) \wedge (x_6 \vee x_{11} \vee x_4) \wedge (\overline{x_{15}} \vee \overline{x_{17}} \vee x_7) \wedge (\overline{x_6} \vee x_{19} \vee x_{13}) \wedge (\overline{x_{12}} \vee \overline{x_9} \vee x_5) \wedge (x_{12} \vee x_1 \vee x_{14}) \wedge (x_{20} \vee x_3 \vee x_2) \wedge (x_{10} \vee \overline{x_7} \vee \overline{x_8}) \wedge (\overline{x_5} \vee x_9 \vee \overline{x_{12}}) \wedge (x_{18} \vee \overline{x_{20}} \vee x_3) \wedge (\overline{x_{10}} \vee \overline{x_{18}} \vee \overline{x_{16}}) \wedge (x_1 \vee \overline{x_{11}} \vee \overline{x_{14}}) \wedge (x_8 \vee \overline{x_7} \vee \overline{x_{15}}) \wedge (\overline{x_8} \vee x_{16} \vee \overline{x_{10}})$, where, for a Boolean variable x_i, $\overline{x_i}$ denotes the negation of x_i, $1 \leq i \leq 20$. The formula Φ was designed so as to have a unique satisfying truth assignment, namely $x_1 = F$, $x_2 = T$, $x_3 = F$, $x_4 = F$, $x_5 = F$, $x_6 = F$, $x_7 = T$, $x_8 = T$, $x_9 = F$, $x_{10} = T$, $x_{11} = T$, $x_{12} = T$, $x_{13} = F$, $x_{14} = F$, $x_{15} = T$, $x_{16} = T$, $x_{17} = T$, $x_{18} = F$, $x_{19} = F$, $x_{20} = F$.

The DNA computing experiment that solved the problem, [11], was based on the following non-deterministic algorithm.

Input: A Boolean formula Φ in 3-CNF.
Step 1: Generate the set of all possible truth value assignments.
Step 2: Remove the set of truth value assignments that make the first clause false.
Step 3: Repeat Step 2 for all the clauses of the input formula.
Output: The remaining (if any) truth value assignments.

To implement this algorithm, the input data was encoded as follows. Every variable x_k, $k = 1, \ldots, 20$, was associated with two distinct 15-mer DNA single strands. One of them, denoted by X_k^T, represented true (T), while the second, denoted by X_k^F, represented false (F).

Below are some examples of the particular 15-mer sequences - none of which contained the nucleotide G - synthesized and used in the experiment:

$$X_2^T = ATT\ TCC\ AAC\ ATA\ CTC, \quad X_2^F = AAA\ CCT\ AAT\ ACT\ CCT,$$
$$X_3^T = TCA\ TCC\ TCT\ AAC\ ATA, \quad X_3^F = CCC\ TAT\ TAA\ TCA\ ATC.$$

Using these 15-mer encodings for the two truth values of all the 20 variables, the library consisting of all possible 2^{20} truth assignments was assembled using the mix-and-match combinatorial synthesis technique of [39]. In brief, oligonucleotides for X_{20}^T and X_{20}^F were synthesized separately, then mixed together. The mixture was divided in half and the result put in two separate test tubes. The synthesis was restarted separately, with sequences X_{19}^T and X_{19}^F respectively. In principle, the process can be repeated until the desired library is obtained. In practice, two half-length libraries were created separately, and then linked together using a polymerase-chain extension similar to that in [105]. Each *library strand* encoding a truth assignment was thus represented by a 300-mer DNA strand consisting of the ordered catenation of twenty 15-mer value sequence, one for each variable, as follows:

$$X_1 X_2 \ldots X_{20}, \text{ where } \alpha_i \in \{X_i^T, X_i^F\}, 1 \leq i \leq 20.$$

The biocomputation *wet-ware* essentially consisted of a glass module filled with a gel containing the library, as well as twenty four glass *clause modules*, one for each of the 24 clauses of the formula. Each clause module was filled with gel containing probes (immobilized DNA single strands) designed to bind only library strands encoding truth assignments satisfying that clause.

The strands were moved between the modules with the aid of gel electrophoresis, i.e., by applying an electric current that resulted in the migration of the negatively charged DNA strands through the gel.

The protocol started with the library passing through the first clause module, wherein library strands containing the truth assignments satisfying the first clause (i.e. library strands containing sequences X_3^F, or X_{16}^F, or X_{18}^T) were captured by the immobilized probes, while library strands that did not satisfy the first clause (i.e. library strands containing sequences X_3^T, and X_{16}^T, and X_{18}^F) continued into a buffer reservoir. The captured strands were then released by raising the temperature, and used as input to the second clause module, etc. At the end, only the strand representing the truth assignment that satisfied all 24 clauses remained.

The output strand was PCR amplified with primer pairs corresponding to all four possible true-false combinations of assignments for the first and last variable x_1 and x_{20}. None except the primer pair $(X_1^F, \overrightarrow{X_{20}^F})$ showed any bands, indicating thus two truth values of the satisfying assignment, namely $x_1 = F$ and $x_{20} = F$. The process was repeated for each of the variable pairs (x_1, x_k), $2 \leq k \leq 19$, and, based on the lengths of the bands observed, value assignments were given to the variables. These experimentally derived values corresponded to the unique satisfying assignment for the formula Φ, concluding thus the experiment.

One of the remarkable features of this benchmark DNA computing experiment was that the sole bio-operation that was used (except during input and output) was Watson-Crick complementarity based annealing and melting.

Generally, it is believed that DNA computers that use a brute-force search algorithm for SAT are limited to 60 to 70 variables [79]. Several other algorithms that do not use brute force, such as the breadth-first search algorithm [110], and random walk algorithms [80, 32] have been proposed. With the breadth-first search algorithm, the capacity of a DNA computer can be theoretically increased to about 120 variables [110]. A recent example of this approach that avoids the generation of the full solution space is a solution to the SAT problem using a DNA computing algorithm based on ligase chain reaction, [108]. This bio-algorithm can solve an n-variable m-clause SAT problem in m steps, and the computation time required is $O(3m + n)$. Instead of generating the full-solution DNA library, this bio-algorithm starts with an empty test tube and then generates solutions that partially satisfy the SAT formula. These partial solutions are then extended step by step by the ligation of new variables using DNA ligase. Correct strands are amplified and false strands are pruned by a ligase chain reaction (LCR) as soon as they fail to satisfy the conditions.

The two DNA computing experiments described in Section 3 and Section 4 are historically significant instances belonging to a vast and impressive array of often astonishing DNA computing experiments, with potential applications to, e.g., nanorobotics, nanocomputing, bioengineering, bio-nanotechnology, and micromedicine. Several of these significant experiments are described in Chapters **??**, **??**, **??**, **??**, and **??** of this handbook.

5 DNA Complementarity and Formal Language Theory

The preceding two sections described two milestone DNA computing experiments whose main "computational engine" was the Watson-Crick complementarity between DNA strands. We now turn our attention to the study of Watson-Crick complementarity from a theoretical point of view, and describe the impact this notion has had on theoretical computer science. This section focuses mainly on the formal language theoretical and coding theoretical approach to DNA-encoded information, and highlights some of the theoretical concepts and results that emerged from these studies.

The idea of formalizing and investigating DNA or RNA molecules and their interactions by using the apparatus of formal language theory is a natural one. Indeed, even though the processes involving DNA molecules are driven by complex biochemical reactions, the primary information is encoded in DNA sequences. Therefore, even without the inclusion of all the thermodynamic parameters that operate during DNA-DNA interactions, formal language theory and coding theory are capable of providing a uniform and powerful framework for an effective study of DNA-encoded information.

To briefly describe the problem of encoding information as DNA strands for DNA computing experiments, one of the main differences between electronic information and DNA-encoded information is that the former has a fixed address and is reusable, while the latter is not. More precisely, DNA strands float freely in solution in a test-tube and, if one of the input strands for a bio-operation has become involved in another, unprogrammed, hybridization, that input strand is unavailable for the bio-operation at hand, compromising thus the final result of the experiment. Thus, a considerable effort has been dedicated to finding "optimal" DNA sequences for encoding the information on DNA so as to prevent undesirable interactions and favour only the desirable programmed ones.

Standard biological methods evaluating and predicting hybridization between DNA molecules conditions rely on thermodynamical parameters such as the free energy ΔG (intuitively, the energy needed to melt DNA bonds), and the melting temperature of DNA molecules. For the calculation of these quantities, not only the WK complementarity between single bases is important, but also the WK complementarity (or lack thereof) of their neighbouring bases with their counterparts: the *nearest-neighbour model* [101] has been frequently used for this purpose. However, in many natural processes the WK complementarity alone plays a crucial role. Furthermore, together with various similarity metrics, the WK complementarity has been often used to obtain an approximate characterization of DNA hybridization interactions.

This section is devoted to describing methods of discrete mathematics and formal language theory which allow for rapid and mathematically elegant characterization of (partially) complementary DNA molecules, their sets, and set-properties relevant for potential cross-hybridizations. For other approaches to this problem and to the problem of design of molecules for DNA computing, the reader is referred to [?] in this handbook. The section is organized as follows. Section 5.1 describes the problem of optimal encoding of information for DNA computing experiments that was the initial motivation for this research, as well as it introduces the basic definitions and notation. Section 5.2 describes the problem of intramolecular hybridization (hybridization within one molecule, resulting, e.g., in hairpin formation) and results related to hairpin languages and hairpin-free languages. Section 5.3 describes the problem of intermolecular hybridization (interaction between two or more DNA molecules) and the theoretical concepts and results motivated by this problem. Section 5.4 investigates properties of languages that guarantee that even undesirable imperfect bonds between DNA strands are avoided.

5.1 DNA Encoding: Problem Setting and Notation

In the process of designing DNA computational experiments, as well as in many general laboratory techniques, special attention is paid to the design of an initial set of “good” DNA strands. References [87, 98] and others distinguish two elementary subproblems of the encoding sequence design:

- *Positive* design problem: Design a set of input DNA molecules such that there is a sequence of reactions which produces the correct result.
- *Negative* design problem: Design a set of input DNA molecules that do not interact in undesirable ways, i.e., do not produce incorrect outputs, and/or do not consume molecules necessary for other, programmed, interactions.

The positive design problem is usually highly related to a specific experiment and it is reported to be hard to find a general framework for its solution. In contrast, the negative design problem can be solved on a general basis by construction of a library of molecules which do not allow for undesired mutual hybridizations. According to [98], the following conditions must be guaranteed: (1) no strand forms any undesired secondary structure such as hairpin loops (Figure 9(a)), (2) no string in the library hybridizes with any string in the library, and (3) no string in the library hybridizes with the complement of any string in the library (Figure 9(b) or (c)). Many laboratory techniques stress the importance of a unified framework for the negative design. An example is the multiplex PCR in which a set of PCR primers is used simultaneously in a single test tube and mutual bonds between primers must be prevented.

A related issue often studied together with the problem of unwanted hybridization is the uniqueness of the oligonucleotides used in experiments. More precisely, one requires that individual oligonucleotides in a mixture differ substantially from each other such that they (and eventually longer sequences produced by their catenation) can be easily distinguished. Such a property in the mathematical sense is typical for *codes*, hence many authors adopted this naming convention for sets of oligonucleotides, usually of a fixed length, whose elements are then called *DNA codewords.*

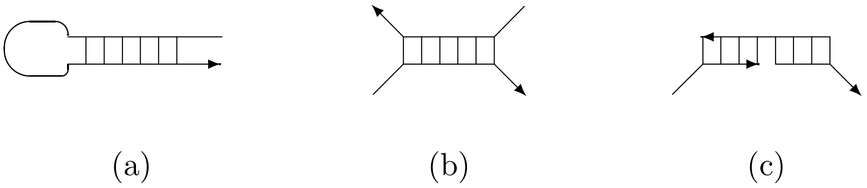

Figure 9: Types of undesired (a) intramolecular and (b), (c) intermolecular hybridizations.

A wide spectrum of methods devoted to the DNA encoding design exists. Thermodynamical methods such as those in [30, 33] provide the most precise results but are computationally the most expensive. Experimental studies trying to construct DNA codes *in vitro* with the help of the PCR operation can be found, e.g., in [16, 31]. An opposite approach relying solely on the WK complementarity allows for fastest but least precise methods (see, e.g., [54, 60, 65] as examples). Approximation methods trying to capture key aspects of the nearest neighbour thermodynamic model represent an intermediate step between these two methodologies. Various discrete metrics based often on Hamming or Levenshtein distance have been studied, e.g., in [36, 37, 46, 47]. The reader is referred to monographs [4, 58, 94] for an overview.

In the sequel we focus on the characterization of DNA hybridization and unwanted bonds by means of the concepts of formal language and coding theory. We also give simple examples of construction of DNA codes.

To describe DNA bonds formally, we represent the single-stranded DNA molecules by strings over the *DNA alphabet* $\Delta = \{A, C, T, G\}$, and we reduce their mutual reactions to formal manipulation of these strings. Therefore, some formal language prerequisites are necessary. For further details the reader is referred to [56], [18], [100].

An *alphabet* is a finite and nonempty set of symbols. In the sequel we shall use a fixed non-singleton alphabet Σ, as a generalization of the natural DNA alphabet Δ. The set of all words over Σ is denoted by Σ^*. This set includes the *empty word* λ. The length of a word $w \in \Sigma^*$ is denoted by $|w|$. For an $x \in \Sigma^+$, $|w|_x$ denotes the number of occurrences of x within w. For a non-negative integer n and a word w, we use w^n to denote the word that consists of n concatenated copies of w. A word v is a *subword* of w if $w = xvy$ for some words x and y. In this case, if $|x| + |y| > 0$ then v is a *proper subword*. By $\mathrm{Sub}(w)$ we denote the set of all subwords of w. For a positive integer k, we use $\mathrm{Sub}_k(w)$ to denote the set of subwords of length k of w. We say that $u \in \Sigma^*$ is a prefix of a word $v \in \Sigma^*$, and denote it by $u \leq v$, if $v = ut$ for some $t \in \Sigma^*$. Two words u, v are said to be *prefix comparable*, denoted by $u \sim_p v$ if one of them is a prefix of the other. In a similar manner, u is said to be a suffix of v if $v = su$ for some $s \in \Sigma^*$. By $\mathrm{Pref}(u)$ $(\mathrm{Suff}(u))$, we denote the sets of all prefixes (respectively suffixes) of u.

The relation of *embedding order* over words is defined as follows: $u \leq_e w$ iff (if and only if)

$$u = u_1 u_2 \cdots u_n,\ w = v_1 u_1 v_2 u_2 \cdots v_n u_n v_{n+1}$$

for some integer n with $u_i, v_j \in \Sigma^*$.

A language L is a set of words, or equivalently a subset of Σ^*. A language is said to be λ-free if it does not contain the empty word. If n is a non-negative integer, we write L^n for the language consisting of all words of the form $w_1 \cdots w_n$ such that each w_i is in L, and $L^{\geq n}$ for the language consisting of all catenations of at least n words from L. We also write L^* for the language $L^0 \cup L^1 \cup L^2 \cup \cdots$, and L^+ for the language $L^* \backslash \{\lambda\}$. The set $\mathrm{Sub}(L) = \bigcup_{w \in L} \mathrm{Sub}(w)$ we call the set of all subwords of L. The families of regular, linear, context-free, and recursively enumerable languages are denoted by REG, LIN, CF, and RE, respectively.

Many approaches to the construction of DNA encoding are based on the assumption that the set of molecules in a test tube (*tube language*) L is equal to, or a subset of, K^+, where K is a finite language whose elements are called *codewords*. In general, K might contain codewords of different lengths. In many cases, however, the set K consists of words of a certain fixed length l. In this case, we shall refer to K as a *code of length l*.

A mapping $\alpha : \Sigma^* \to \Sigma^*$ is called a *morphism* (*antimorphism*) of Σ^* if $\alpha(uv) = \alpha(u)\alpha(v)$ (respectively $\alpha(uv) = \alpha(v)\alpha(u)$) for all $u, v \in \Sigma^*$. Note that a morphism or an antimorphism of Σ^* are completely defined if we define their values on the letters of Σ.

If for a morphism α, $\alpha(a) \neq \lambda$ for each $a \in V$, then α is said to be *λ-free*. A *projection* associated to Σ is a morphism $pr_\Sigma : (V \cup \Sigma)^* \to \Sigma^*$ such that $pr_\Sigma(a) = a$ for all $a \in \Sigma$ and $pr_\Sigma(b) = \lambda$ otherwise. A morphism $h : V^* \to \Sigma^*$ is called a *coding* if $h(a) \in \Sigma$ for all $a \in V$ and *weak-coding* if $h(a) \in \Sigma \cup \{\lambda\}$ for all $a \in V$. For a family of languages FL, let us denote by $\mathrm{Cod}(FL)$ (respectively $\mathrm{wcod}(FL)$) the family of languages of the form $h(L)$, for $L \in FL$ and h a coding (respectively weak-coding).

The *equality set* of two morphisms $h_1, h_2 : V^* \to \Sigma^*$ is defined as:

$$\mathrm{EQ}(h_1, h_2) = \{w \in V^* \mid h_1(w) = h_2(w)\}.$$

An *involution* $\theta : \Sigma \to \Sigma$ of Σ is a mapping such that θ^2 is equal to the identity mapping, i.e., $\theta(\theta(x)) = x$ for all $x \in \Sigma$. It follows then that an involution θ is bijective and $\theta = \theta^{-1}$. The identity mapping is a trivial example of an involution. An involution of Σ can be extended to either a morphism or an antimorphism of Σ^*. For example, if the identity of Σ is extended to a morphism of Σ^*, we obtain the identity involution of Σ^*. However, if we extend the identity of Σ to an antimorphism of Σ^* we obtain instead the mirror-image involution of Σ^* that maps each word u into u^R where

$$u = a_1 a_2 \ldots a_k,\ \ u^R = a_k \ldots a_2 a_1, a_i \in \Sigma, 1 \leq i \leq k.$$

A word w which is equal to its reverse w^R is called a *palindrome*. If we consider the DNA alphabet Δ, then the mapping $\tau : \Delta \to \Delta$ defined by $\tau(A) =$

$T, \tau(T) = A, \tau(C) = G, \tau(G) = C$ can be extended in the usual way to an antimorphism of Δ^* that is also an involution of Δ^*. This involution formalizes the notion of Watson-Crick complementarity and will therefore be called the *DNA involution* [63].

5.2 Intramolecular Bond (Hairpin) Analysis

In this section we focus on mathematical properties of DNA hairpins and their importance in DNA encodings. A *DNA hairpin* is a particular type of DNA secondary structure illustrated in Figure 10.

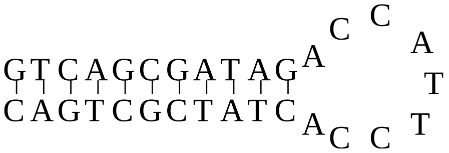

Figure 10: A single-stranded DNA molecule forming a hairpin loop.

Hairpin-like secondary structures play an important role in insertion/deletion operations with DNA. Hairpin-freeness is crucial in the design of primers for the PCR reaction. Among numerous applications of hairpins in DNA computing we mention only the Whiplash PCR computing techniques [97] and the DNA RAM [**?**]. We refer the reader, e.g., to [75, 76, 87] for a characterization and design of tube languages with or without hairpins. Coding properties of hairpin-free languages were studied in [59, 60]. A language-theoretical characterization of hairpins and hairpin languages was given in [95]. Hairpins have also been studied in the context of bio-operations occurring in single-celled organisms. For example, the operation of hairpin inversion was defined as one of the three molecular operations that accomplish gene assembly in ciliates [26, 27, 38]. Applications of hairpin structures in biocomputing and bio-nanotechnology are discussed in [**?**, **?**] in this handbook.

The following definition formally specifies hairpin as a structure described in Figure 10, whose stem consists of at least k base pairs. This condition is motivated by the fact that a hairpin with shorter stem is less stable. An oligonucleotide which does not satisfy this condition is said to be hairpin-free.

Definition 5.1 *([59, 66]) Let θ be a (morphic or antimorphic) involution of Σ^* and k be a positive integer. A word $u \in \Sigma^*$ is said to be θ-k-hairpin-free or simply hp(θ, k)-free if $u = xvy\theta(v)z$ for some $x, v, y, z \in \Sigma^*$ implies $|v| < k$.*

We denote by $hpf(\theta, k)$ the set of all hp(θ, k)-free words in Σ^*. The complement of $hpf(\theta, k)$ is the set of all hairpin-forming words over Σ and is denoted by $hp(\theta, k) = \Sigma^* \setminus hpf(\theta, k)$. Observe that $hp(\theta, k+1) \subseteq hp(\theta, k)$ for all $k > 0$. A language L is said to be θ-k-hairpin-free or simply hp(θ, k)-free if $L \subseteq hpf(\theta, k)$.

Example 1 Let $\theta = \tau$, the DNA involution over Δ^*. Then:

$$hpf(\theta, 1) = \{A, C\}^* \cup \{A, G\}^* \cup \{T, C\}^* \cup \{T, G\}^*$$

In the version of definition 5.1 given in [59], a θ-k-hairpin-free language was called θ-subword-k-code. The authors focused on their coding properties and relations to other types of codes. A restriction on the length of the loop of a hairpin was also considered: $1 \leq |y| \leq m$ for some $m \geq 1$. Most of the results mentioned in this section remain valid if this additional restriction is considered.

Theorem 5.2 *([95]) The languages $hp(\theta, k)$ and $hpf(\theta, k)$, $k \geq 1$, are regular.*

Figure 11 illustrates a nondeterministic finite automaton (NFA) accepting the language $hp(\theta, 2)$ over the alphabet $\{a, b\}$ where $\theta(a) = b$ and $\theta(b) = a$. Given the above characterization of $hp(\theta, k)$ and $hpf(\theta, k)$, the following result is rather immediate:

Theorem 5.3 *([66]) The following problem is decidable in linear (or cubic, respectively) time with respect to (w.r.t.) $|M|$:*

Input: A nondeterministic regular (pushdown, respectively) automaton M.

Output: Yes/No depending on whether $L(M)$ is $hp(\theta, k)$-free.

The *maximality problem* of hairpin-free languages is stated as follows: can a given language $L \subseteq \Sigma^*$, satisfying a certain property (e.g., to be a hairpin-free language), be still extended without loss of this property? Formally, a language $L \subseteq \Sigma^*$ satisfying a property $\mathcal{P}$ is said to be maximal w.r.t. $\mathcal{P}$ iff $L \cup \{w\}$ does not satisfy $\mathcal{P}$ for any $w \in M \setminus L$, where M is a fixed library of available strands.

Theorem 5.4 *([66]) The following problem is decidable in time $\mathcal{O}(|M_1| \cdot |M_2|)$ (or $\mathcal{O}(|M_1| \cdot |M_2|^3)$, respectively):*

Input: A positive integer k, a deterministic finite (pushdown, respectively) automaton M_1 accepting a $hp(\theta, k)$-free language, and an NFA M_2.

Output: Yes/No depending on whether there is a word $w \in L(M_2) \setminus L(M_1)$ such that $L(M_1) \cup \{w\}$ is $hp(\theta, k)$-free.

For hairpin-free languages it is relatively straightforward to solve the *optimal negative design problem*: to construct a set of hairpin-free DNA words of a given size, where the words can be chosen from a certain library. All the hairpin-free sets are subsets of $hpf(\theta, k)$. For example, if the length of the desired DNA words equals a constant ℓ, the optimal hairpin-free set is simply $hpf(\theta, k) \cap \Sigma^\ell$. Due to Theorem 5.2, the set $hpf(\theta, k)$ is regular and hence can be accepted by a finite automaton. The size of the automaton, however, grows exponentially with respect to k.

Theorem 5.5 *([66]) Consider the DNA alphabet $\Delta = \{A, C, T, G\}$ and the DNA involution τ.*

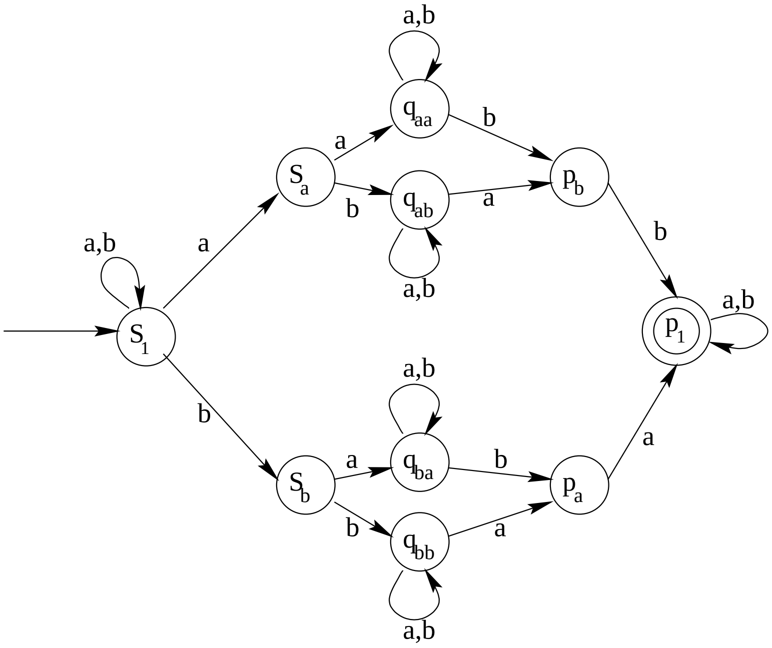

Figure 11: An NFA accepting the language $hp(\theta, 2)$ over the alphabet $\{a, b\}$, where the antimorphism is defined as $\theta(a) = b$ and $\theta(b) = a$.

(i) The size of a minimal NFA accepting $hp(\tau, k)$ is at most $15 \cdot 4^k$. The number of its states is between 4^k and $3 \cdot 4^k$.

(ii) The number of states of either a minimal deterministic finite automaton (DFA) or an NFA accepting $hpf(\tau, k)$ is between $2^{2^{k-1}}$ and $2^{3\cdot 2^{2k}}$.

The reader is referred to [66] for a generalization of the above theorem for the case of an arbitrary alphabet and arbitrary involution. The construction of the automaton is illustrated in Figure 11 for the case of alphabet $\{a, b\}$ and an antimorphism θ, where $\theta(a) = b$ and $\theta(b) = a$.

Problems analogous to Theorems 5.2–5.5 have been studied also in the case of *scattered hairpins* and *hairpin frames* which represent more complex but rather common types of intramolecular hybridization. The definition of scattered hairpins covers structures like the one described in Figure 12.

Definition 5.6 *([66]) Let θ be an involution of Σ^* and let k be a positive integer. A word $u = wy$, for $u, w, y \in \Sigma^*$, is θ-k-scattered-hairpin-free or simply shp(θ, k)-free if for all $t \in \Sigma^*$, $t \leq_e w$, $\theta(t) \leq_e y$ implies $|t| < k$.*

Similarly, the following definition of hairpin frames characterizes secondary structures containing several complementary sequences such as that in Figure 13.

Definition 5.7 *([66]) The pair $(v, \theta(v))$ in a word u of the form $u = xvy\theta(v)z$, for $x, v, y, z \in \Sigma^*$, is called an* hp-pair *of u. The sequence of hp-pairs $(v_1, \theta(v_1))$,*

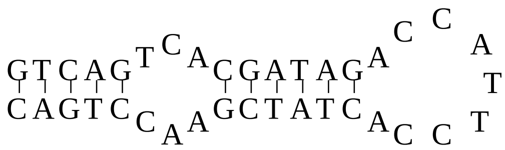

Figure 12: An example of a scattered hairpin – a word in *shp*$(\tau, 11)$.

$(v_2, \theta(v_2)), \cdots, (v_j, \theta(v_j))$ *of the word* u *in the form:*

$$u = x_1 v_1 y_1 \theta(v_1) z_1 x_2 v_2 y_2 \theta(v_2) z_2 \cdots x_j v_j y_j \theta(v_j) z_j$$

is called an hp-frame of degree j *of* u *or simply an* hp(j)-frame *of* u*.*

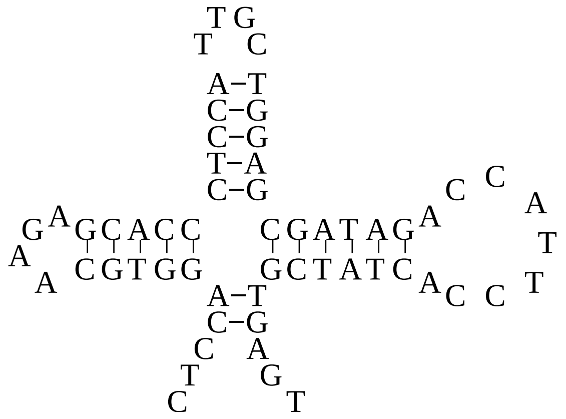

Figure 13: An example of a hairpin frame.

Several other studies have been published, focusing on formal-language aspects of the hairpin formation. A complete characterization of the syntactic monoid of the language consisting of all hairpin-free words over a given alphabet was given in [71]. The reference [83] described formal language operations of hairpin completion and reduction and studied their closure and other mathematical properties. DNA trajectories – a new formal tool for description of scattered hairpins – were presented in [34], where also complexity of the set of hairpin-free words described by a set of DNA trajectories and closure properties of hairpin language classes were studied. Hairpin finite automata with the ability to apply the hairpin inversion operation to the remaining part of the input were introduced in [10]. The authors studied the power of hairpin-inspired operations and the resulting language classes. Finally, the reference [74] focused on a related secondary structure based on intramolecular bonds – *pseudoknots* (see also [33] for more information on pseudoknots). Authors provided mathematical formalization of pseudoknots and obtained several properties of pseudoknot-bordered and -unbordered words.

5.3 How to Avoid DNA Intermolecular Bonds

In this section we formally characterize several properties of a tube language $L \subseteq \Sigma^+$, which prohibit various types of undesired hybridizations between two DNA strands. Many authors assume, for simplicity, that hybridization occurs only between those parts of single-stranded DNA molecules which are perfectly complementary. The following language properties have been considered in [57, 63, 67].

(A) **θ-non-overlapping**: $L \cap \theta(L) = \emptyset$.

(B) **θ-compliant**: $\forall w \in L,\ x, y \in \Sigma^*,\ w, x\theta(w)y \in L \Rightarrow xy = \lambda$.

(C) **θ-p-compliant**: $\forall w \in L,\ y \in \Sigma^*,\ w, \theta(w)y \in L \Rightarrow y = \lambda$.

(D) **θ-s-compliant**: $\forall w \in L,\ y \in \Sigma^*,\ w, y\theta(w) \in L \Rightarrow y = \lambda$.

(E) **strictly θ-compliant**: both θ-compliant and θ-non-overlapping.

(F) **θ-free**: $L^2 \cap \Sigma^+\theta(L)\Sigma^+ = \emptyset$.

(G) **θ-sticky-free**: $\forall w \in \Sigma^+,\ x, y \in \Sigma^*,\ wx, y\theta(w) \in L \Rightarrow xy = \lambda$.

(H) **θ-3$'$-overhang-free**: $\forall w \in \Sigma^+,\ x, y \in \Sigma^*,\ wx, \theta(w)y \in L \Rightarrow xy = \lambda$.

(I) **θ-5$'$-overhang-free**: $\forall w \in \Sigma^+,\ x, y \in \Sigma^*,\ xw, y\theta(w) \in L \Rightarrow xy = \lambda$.

(J) **θ-overhang-free**: both θ-3$'$-overhang-free and θ-5$'$-overhang-free.

For convenience, we agree to say that a language L containing the empty word has one of the above properties if $L \setminus \{\lambda\}$ has that property. Observe that (F) avoids situations like Figure 9(c), while other properties exclude special cases of 9(b).

In [60], a θ-non-overlapping language was said to be *strictly* θ. Generally, if any other property holds in conjunction with (A), we add the qualifier *strictly.* We have already used this notation for the property (E). Both *strict* and *non-strict* properties turn out to be useful in certain situations.

For example, a common way to check for the presence of a certain single-stranded molecule w is to add to the solution its complement $\tau(w)$, and use enzymes to destroy any molecules which are not fully double stranded. Simultaneously, we want to prevent all other hybridizations except w and $\tau(w)$. This condition is equivalent to testing whether the whole solution, including w and $\tau(w)$, is non-strictly bond-free (exact matches are allowed).

Further properties have been defined in [60] for a language L. Observe that the property (K) below avoids bonds like those in Figure 9(a), with a restricted length of the loop part:

(K) **$\theta(k, m_1, m_2)$-subword compliant**: $\forall u \in \Sigma^k$, $\Sigma^* u \Sigma^m \theta(u) \Sigma^* \cap L = \emptyset$ for $k > 0$, $m_1 \leq m \leq m_2$.

(L) **θ-k-code**: $\text{Sub}_k(L) \cap \text{Sub}_k(\theta(L)) = \emptyset$, $k > 0$.

The property (L) was also considered implicitly in [9] and [40]. In particular, the reference [9] considered tube languages of the form $(sZ)^+$ satisfying (L), where s is a fixed word of length k and Z is a code of length k – the notation sZ represents the set of all words sz such that z is in Z.

The following property was defined for $\theta = I$, the identity relation, in [54]. A language L is called

(M) **solid** if:

1. $\forall x, y, u \in \Sigma^*$, $u, xuy \in L \Rightarrow xy = \lambda$, and
2. $\forall x, y \in \Sigma^*, u \in \Sigma^+$, $xu, uy \in L \Rightarrow xy = \lambda$.

L is *solid relative* to a language $M \subseteq \Sigma^*$ if 1. and 2. above must hold only when there are $p, q \in \Sigma^*$ such that $pxuyq \in M$. L is called *comma-free* if it is solid relative to L^*. Solid languages were also used in [67] as a tool for constructing error-detecting tube languages that were invariant under bio-operations.

Figure 14 shows the hierarchy of some of the above language properties. Arrows stand for inclusion relations among language classes satisfying these properties.

Example 2 ([65]) Consider the language $L = \{A^nT^n \,|\, n \geq 1\} \subset \Delta^+$, and the DNA involution τ. Observe that $\tau(L) = L$. We can deduce that L is:

- neither τ-non-overlapping, nor τ-k-code for any $k \geq 1$;
- not τ-compliant, as for $w = A^nT^n$, $x = A$, $y = T$ we have $w, x\tau(w)y \in L$;
- τ-p-compliant, as $w, \theta(w)y \in L$ implies $w = A^nT^n$, $y = \lambda$; similarly, L is τ-s-compliant;
- not τ-free, as $A^nT^nA^mT^m$, $n, m > 1$, is both in L^2 and in $\Delta^+L\Delta^+$;
- not τ-sticky-free, as for $w = y = A^n$ $x = T^n$ we have $wx, y\tau(w) \in L$;
- τ-$3'$-overhang-free, as $wx, \tau(w)y \in L$ implies $w = A^nT^m$, $x = T^{n-m}$, $y = T^{m-n}$ and hence $xy = \lambda$; similarly, L is τ-$5'$-overhang-free and hence τ-overhang-free;
- not $\theta(k, m_1, m_2)$-subword compliant for any k, m_1, m_2.

For further details and relations between the above listed DNA language properties we refer the reader to [60, 65, 67].

To establish a common framework allowing to handle various types of unwanted hybridization in a uniform way, it is necessary to introduce the generalizing concept of word operations on trajectories. Consider a *trajectory alphabet* $V = \{0, 1\}$ and assume $V \cap \Sigma = \emptyset$. We call any string $t \in V^*$ a *trajectory.* A trajectory is essentially a syntactical condition which specifies how a binary word operation is applied to the letters of its two operands. Let $t \in V^*$ be a trajectory and let α, β be two words over Σ.

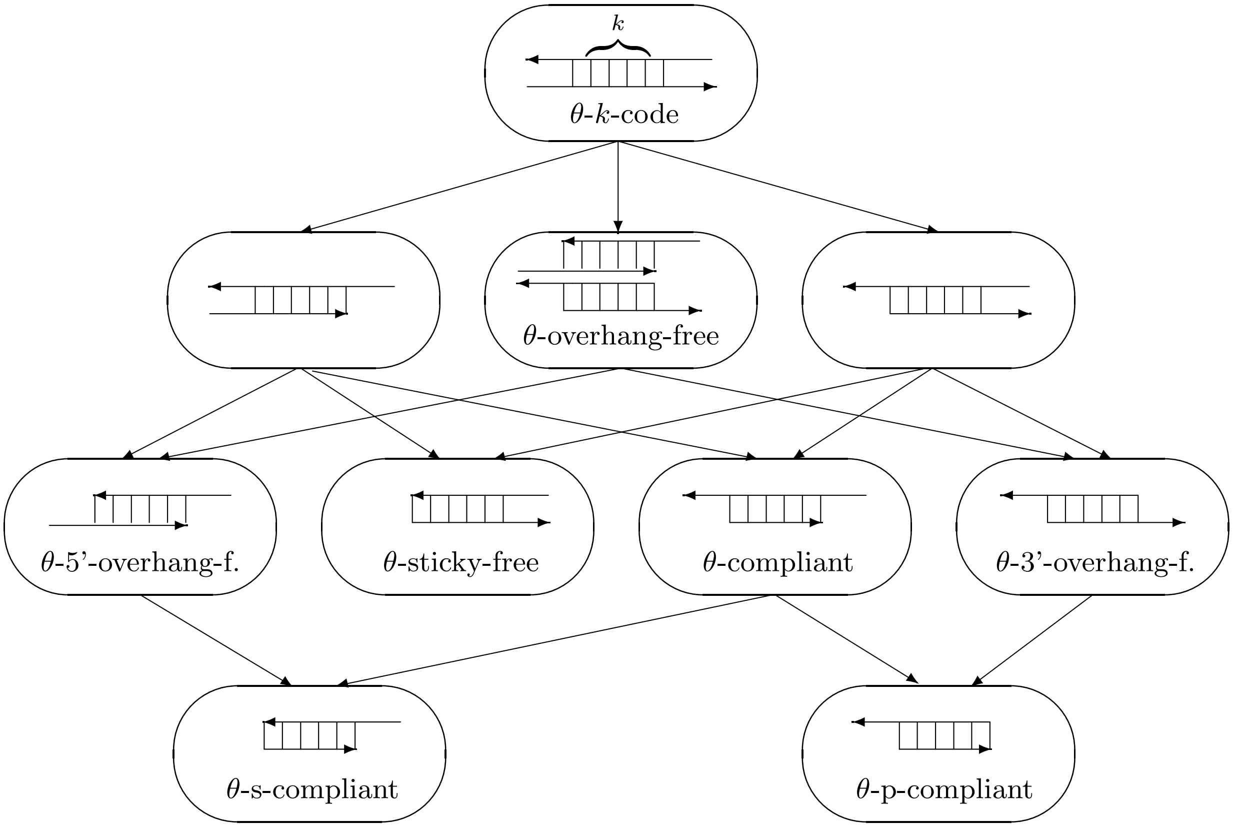

Figure 14: Classes of tube languages free of certain types of undesired hybridization.

Definition 5.8 *[86] The shuffle of α with β on a (fixed) trajectory t, denoted by $\alpha \sqcup\!\sqcup_t \beta$, is the following binary word operation:*

$$\begin{aligned}\alpha \sqcup\!\sqcup_t \beta \quad = \quad & \{\alpha_1\beta_1 \ldots \alpha_k\beta_k \mid \alpha = \alpha_1 \ldots \alpha_k,\ \beta = \beta_1 \ldots \beta_k,\ t = 0^{i_1}1^{j_1} \ldots 0^{i_k}1^{j_k}, \\ & \textit{where } |\alpha_m| = i_m \textit{ and } |\beta_m| = j_m \textit{ for all } m,\ 1 \leq m \leq k\}.\end{aligned}$$

Example 3 Let $\alpha = a_1a_2 \ldots a_8$, $\beta = b_1b_2 \ldots b_5$ and assume that $t = 0^31^20^310101$. The shuffle of α and β on the trajectory t is:

$$\alpha \sqcup\!\sqcup_t \beta = \{a_1a_2a_3b_1b_2a_4a_5a_6b_3a_7b_4a_8b_5\}.$$

Observe that the result of the operation is generally a set of words, though in the case of shuffle on trajectory it is always a singleton or the empty set. Notice also that $\alpha \sqcup\!\sqcup_t \beta = \emptyset$ if $|\alpha| \neq |t|_0$ or $|\beta| \neq |t|_1$.

A *set of trajectories* is any set $T \subseteq V^*$. The *shuffle of α with β on the set T*, denoted by $\alpha \sqcup\!\sqcup_T \beta$, is:

$$\alpha \sqcup\!\sqcup_T \beta = \bigcup_{t \in T} \alpha \sqcup\!\sqcup_t \beta. \qquad (1)$$

The shuffle on (sets of) trajectories generalizes several traditional word operations. Let, for example, $T = 0^*1^*$. Then $\sqcup\!\sqcup_T \; = \; \cdot$, the operation of catenation.

To characterize the properties of tube languages described above, we define formally a *property* $\mathcal{P}$ as a mapping $\mathcal{P} : 2^{\Sigma^*} \longrightarrow \{\text{true}, \text{false}\}$. We say that a language L has (or satisfies) the property $\mathcal{P}$ if $\mathcal{P}(L) = \text{true}$. The next definition introduces a general concept of *bond-free property* which covers surprisingly many types of undesired bonds studied in the literature.

Definition 5.9 *([65]) Consider a language property $\mathcal{P}$. Let there be binary word operations $\diamondsuit_{\text{lo}}$, $\diamondsuit_{\text{up}}$ and an involution θ such that for an arbitrary $L \subseteq \Sigma^*$,* $\mathcal{P}(L) = \text{true}$ iff

(i) $\forall w \in \Sigma^+,\ x, y \in \Sigma^*\ ((w \diamondsuit_{\text{lo}} x) \cap L \neq \emptyset,\ (\theta(w) \diamondsuit_{\text{up}} y) \cap L \neq \emptyset) \Rightarrow xy = \lambda$, *then $\mathcal{P}$ is called a* bond-free property (of degree 2);

(ii) $\forall w, x, y \in \Sigma^*\ ((w \diamondsuit_{\text{lo}} x) \cap L \neq \emptyset,\ (\theta(w) \diamondsuit_{\text{up}} y) \cap L \neq \emptyset) \Rightarrow w = \lambda$, *then $\mathcal{P}$ is called a* strictly bond-free property (of degree 2).

If not stated otherwise, we assume in the sequel that $\diamondsuit_{\text{lo}} = \sqcup\!\sqcup_{T_{\text{lo}}}$ and $\diamondsuit_{\text{up}} = \sqcup\!\sqcup_{T_{\text{up}}}$ for some sets of trajectories T_{lo}, T_{up}. Intuitively, w and $\theta(w)$ represent two complementary oligonucleotides. Then $w \sqcup\!\sqcup_{T_{\text{lo}}} x$ and $\theta(w) \sqcup\!\sqcup_{T_{\text{up}}} y$ represent two DNA single strands which could form a double-stranded DNA molecule with blunt ends, depicted in Figure 14. The bond-free property $\mathcal{P}$ guarantees that $w \sqcup\!\sqcup_{T_{\text{lo}}} x$ and $\theta(w) \sqcup\!\sqcup_{T_{\text{up}}} y$ with nonempty x, y (i), or w (ii), cannot simultaneously exist in L.

Theorem 5.10 *([65])*

(i) The language properties (B), (C), (D), (G), (H), (I), (M.1), (M.2) are bond-free properties.

(ii) The language properties (A), strictly (B)–(D), strictly (G)–(I), (L), strictly (L) are strictly bond-free properties.

Moreover, in both cases the associated sets of trajectories $T_{\text{lo}}, T_{\text{up}}$ are regular.

Proof. Let θ be an antimorphism and let the sets of trajectories T_{lo}, T_{up} corresponding to the listed bond-free properties be:

(A)	$T_{\text{lo}} = T_{\text{up}} = 0^+$
(B)	$T_{\text{lo}} = 0^+$, $T_{\text{up}} = 1^*0^+1^*$
(C)	$T_{\text{lo}} = 0^+$, $T_{\text{up}} = 0^+1^*$
(D)	$T_{\text{lo}} = 0^+$, $T_{\text{up}} = 1^*0^+$
(G)	$T_{\text{lo}} = 0^+1^*$, $T_{\text{up}} = 1^*0^+$
(H)	$T_{\text{lo}} = 0^+1^*$, $T_{\text{up}} = 0^+1^*$
(I)	$T_{\text{lo}} = 1^*0^+$, $T_{\text{up}} = 1^*0^+$
(L)	$T_{\text{lo}} = T_{\text{up}} = 1^*0^k1^*$
(L)	strictly: $T_{\text{lo}} = T_{\text{up}} = 1^*0^k1^* \cup 0^+$

If θ is a morphism, one can similarly define:

(M.1) $T_{\mathrm{lo}} = 0^*$, $T_{\mathrm{up}} = 1^*0^*1^*$
(M.2) $T_{\mathrm{lo}} = 1^*0^+$, $T_{\mathrm{up}} = 0^+1^*$

Consider, e.g., the property (H), θ-3'-overhang-freedom. Then $w \sqcup\!\sqcup_{T_{\mathrm{lo}}} x = \{wx\}$ and $\theta(w) \sqcup\!\sqcup_{T_{\mathrm{up}}} y = \{\theta(w)y\}$. The relations in Definition 5.9 (i) take the form $wx \in L$, $\theta(w)y \in L$ which corresponds to the definition of (H) above. The proofs of the other mentioned properties are analogous. □

Observe that $T_{\mathrm{lo}}, T_{\mathrm{up}}$ for a certain property correspond to the "shape" of the bonds prohibited in languages satisfying the property. The above theorem allows for a general characterization of bond-free properties via a solution of an *unique* language inequation in [65]. As a consequence we obtain the following result.

Theorem 5.11 *([65]) Let $\mathcal{P}$ be a (strictly) bond-free property associated with regular sets of trajectories $T_{\mathrm{lo}}, T_{\mathrm{up}}$. Then the following problem is decidable in quadratic time w.r.t. $|A|$:*

Input: an NFA A.

Output: Yes/No depending on whether $L(A)$ satisfies $\mathcal{P}$.

By Theorem 5.10, the above result applies to the properties (A) – (D), (G) – (J), (M), strictly (B) – strictly (D), strictly (G) – strictly (J), (L), strictly (L), in the case of regular languages, on one hand. On the other hand, for a given context-free language L it is undecidable whether it satisfies certain bond-free properties, e.g., (B) and (F).

Theorem 5.12 *([57]) The following problem is undecidable.*

Input: A bond-free property $\mathcal{P}$ associated with regular sets of trajectories T_{lo}, T_{up}, and a context-free language L.

Output: Yes/No depending on whether $\mathcal{P}(L) =$ true.

An important problem studied in the literature is the *optimal negative design problem:* how to construct a non-crosshybridizing set (i.e., a set of single-stranded molecules which do not mutually hybridize) of a certain required size, given a fixed library of available molecules. In general, even to decide whether such a *finite* set exists is an NP-complete problem and its equivalence with the maximal independent set problem can be easily shown [30]. Various heuristics were used to find a nearly-optimal solution [91]. Here we focus on a similar but in some cases easier *maximality problem* defined formally in Section 5.2.

Theorem 5.13 *([65]) Let $M \subseteq \Sigma^+$ be a regular set of words, and $L \subseteq M$ a regular language satisfying a bond-free property $\mathcal{P}$.*

(a) Let θ be an antimorphism and let $\mathcal{P}$ be one of the properties (B), (C), (D), (G), strictly (B) – strictly (D), strictly (G), (L), strictly (L), or

(b) let θ be a morphism and let $\mathcal{P}$ be one of the properties (B), (C), (D), (H), (I), strictly (B) – strictly (D), strictly (H), strictly (I), (L), strictly (L).

Then there is an algorithm deciding whether there is a $w \in M \setminus L$ such that $L \cup \{w\}$ satisfies $\mathcal{P}$.

Algorithms deciding the maximality of these properties can require an exponential time w.r.t. the size of an NFA accepting L. The same holds when we want to construct a maximal regular set of DNA strands satisfying these bond-free properties. In two important cases, however, a polynomial time can be achieved [65]: (i) for maximality of *regular* non-overlapping sets satisfying the property (A), and (ii) for maximality of *finite* θ-compliant sets satisfying the property (B).

5.4 Preventing Imperfect DNA-DNA Bonds

In Section 5.2 and 5.3 we presented properties of DNA codes based on the assumption that hybridization binds only two perfectly WK complementary single-stranded DNA molecules. In reality, however, thermodynamical laws allow for hybridization also in cases of some 'roughly' complementary molecules with certain irregularities in the WK complementarity sense. This section describes properties of languages that ensure that even such imperfect bindings can be described and eventually prevented.

As we have already mentioned, the negative design problem is rather computationally expensive when using thermodynamical methods. Various approximative methods using discrete similarity metrics have been therefore studied. In [84] and [106], the authors considered codes K of length k satisfying the following property ($H(u, v)$ is the Hamming distance, i.e., the number of mismatches at corresponding positions, between words u and v of the same length).

$\mathbf{X}[d, k]$**:** If u and v are any codewords in K then $H(u, \tau(v)) > d$.

In fact the above property is studied in conjunction with the uniqueness property $H(K) > d$ – here $H(K)$ is the smallest Hamming distance between any two different words in K.

The reference [46] introduced the H-measure based on Hamming distance for two words x and y of length k and explained how this measure can be used to encode instances of the Hamiltonian Path Problem. This measure was also used in [47] to search optimal codes for DNA computing using the shuffle operation on DNA strands. A similar measure was defined in [6, 7] extending the work of [45] and was applied to codes of length k whose words can be concatenated in arbitrary ways. Thus, the tube language here was $L = K^+$. The code K satisfied certain uniqueness conditions. In particular, the tube language $L = K^+$ satisfied the following property.

```
5'- v A A G C G T T C G A w -3'
       \ \  | | |   | | |
   3'-z T T G C A G G C T y -5'
```

Figure 15: Two DNA molecules in which the parts $5' - AAGCGTTCGA - 3'$ and $5' - TCGGACGTT - 3'$ bind together although these parts have different lengths.

Y$[d, k]$**:** If x is a subword of L of length k and v is a codeword in K then $H(x, \tau(v)) > d$.

This property was considered also in [96] for tube languages of the form $K_1 K_2 \cdots K_m$, where each K_i is a certain code of length k.

Finally, [64] introduced the following property of a tube language L, motivated by the fact that the above defined properties **X**, **Y** still allow for certain types of undesired bonds between DNA codewords.

Z$[d, k]$**:** If x and y are any subwords of L of length k then $H(x, \tau(y)) > d$.

The reader can observe that the property **X** is a generalization of (A) from the previous section. Similarly, **Y** is a generalization of (B) and **Z** generalizes (L). Note that any set L satisfying property **Z**$[d, k]$ satisfies also **Y**$[d, k]$. Further relations among different bond-free properties using similarity measures were studied in [64].

The choice of the Hamming distance in the condition $H(x, \tau(y)) \leq d$ for *similarity* between words is a very natural one and has attracted a lot of interest in the literature. One might argue, however, that parts of two DNA molecules could form a stable bond even if they have different lengths. In Figure 15, for example, the bound parts of the two molecules have lengths 10 and 9. Such hybridizations (and even more complex ones) were addressed in [5]. Based on this observation, the condition for two subwords x and y to bind together should be

$$|x|, |y| \geq k \quad \text{and} \quad Lev(x, \tau(y)) \leq d.$$

The symbol $Lev(u, v)$ denotes the *Levenshtein distance* between the words u and v – this is the smallest number of substitutions, insertions and deletions of symbols required to transform u into v.

To establish a general framework which would cover both the similarity measure H, Lev and possibly also others, [64] considered a general binary relation γ on words over Σ, i.e., a subset of $\Sigma^* \times \Sigma^*$. The expression '(u, v) is in γ' can be rephrased as '$\gamma(u, v)$ is true' when we view γ as a logic predicate. A binary relation is called *rational* if it can be realized by a finite transducer.

Definition 5.14 *([64]) A binary relation* sim *is called a* similarity relation with parameters (t, l)*, where t and l are non-negative integers, if the following conditions are satisfied.*

(i) If $\mathrm{sim}(u, v)$ *is true then* $\mathrm{abs}(|u| - |v|) \leq t$*, where* abs *is the absolute value function.*

(ii) If $\mathrm{sim}(u, v)$ *is true and* $|u|, |v| > l$ *then there are proper subwords* x *and* y *of* u *and* v*, respectively, such that* $\mathrm{sim}(x, y)$ *is true.*

We can interpret the above conditions as follows: (i) the lengths of two similar words cannot be too different and (ii) if two words are similar and long enough, then they contain two similar proper subwords.

We shall use the notation $H_{d,k}$ for the relation '$|u|, |v| \geq k$ and $H(u, v) \leq d$', and $Lev_{d,k}$ for '$|u|, |v| \geq k$ and $Lev(u, v) \leq d$'. It is evident that the $H_{d,k}$ is an example of a rational similarity relation with parameters $(0, k)$. It is also easy to show that $Lev_{d,k}$ is a rational similarity relation as well, with parameters $(d, d + k)$.

Based on the above definition, for any similarity relation $\mathrm{sim}(\cdot, \cdot)$ between words and for every involution θ, we define the following property of a language L with strong mathematical properties.

$\mathbf{P}[\theta, \mathrm{sim}]$**:** If x and y are any nonempty subwords of L then $\mathrm{sim}(x, \theta(y))$ is false.

Any language satisfying $\mathbf{P}[\theta, \mathrm{sim}]$ is called a (θ, sim)-*bond-free language*. Although this property seems to be quite general and covering many possible situations, we show that it is only a special case of the strict bond-freeness defined in Section 5.3.

Theorem 5.15 *([64])* $\mathbf{P}[\theta, \mathrm{sim}]$ *is a strictly bond-free property.*

Proof. We define the mappings sim_L and sim_R as follows:

$$\begin{aligned} \mathrm{sim}_L(y) &= \{x \in \Sigma^* \mid \mathrm{sim}(x, y)\}, \\ \mathrm{sim}_R(x) &= \{y \in \Sigma^* \mid \mathrm{sim}(x, y)\}. \end{aligned}$$

Recall that a language L is (θ, sim)-bond-free iff

$$\begin{array}{ll}
\forall x_1, y_1, x_2, y_2 \in \Sigma^*,\ w_1, w_2 \in \Sigma^+ & \\
\quad (x_1 w_1 y_1,\ x_2 w_2 y_2 \in L) \ \Rightarrow\ \text{not } \mathrm{sim}(w_1, \theta(w_2)) & \text{iff} \\
\forall x_1, y_1, x_2, y_2 \in \Sigma^*,\ w_1, w_2 \in \Sigma^+ & \\
\quad (x_1 w_1 y_1,\ x_2 \theta(w_2) y_2 \in L) \ \Rightarrow\ \text{not } \mathrm{sim}(w_1, w_2) & \text{iff} \\
\forall x_1, w_1, y_1, x_2, w_2, y_2 \in \Sigma^* & \\
\quad (x_1 w_1 y_1,\ x_2 \theta(w_2) y_2 \in L, w_2 \in \mathrm{sim}_R(w_1)) \Rightarrow (w_1 = \lambda \text{ or } w_2 = \lambda) & \text{iff} \\
\forall x_1, y_1, x_2, y_2, w \in \Sigma^* & \\
\quad (\{x_1 w y_1\} \cap L \neq \emptyset,\ \{x_2\} \cdot \theta(\mathrm{sim}_R(w) \cap \Sigma^+) \cdot \{y_2\} \cap L \neq \emptyset) \Rightarrow w = \lambda & \text{iff} \\
\forall x, y, w \in \Sigma^* & \\
\quad (w \sqcup\!\sqcup_T x \cap L \neq \emptyset,\ \theta(\mathrm{sim}_R(w)) \sqcup\!\sqcup_T y \cap L \neq \emptyset) \Rightarrow w = \lambda, &
\end{array}$$

where $T = 1^*0^+1^*$. □

Therefore, we have obtained an expression corresponding to the definition of strictly bond-free property. Notice that results analogous to Theorem 5.15 could be proved also for the properties $\mathbf{X}[d,k]$, $\mathbf{Y}[d,k]$ and $\mathbf{Z}[d,k]$. Observe, furthermore, that the operation on words w and y defined as $\theta(\mathrm{sim}_R(w)) \sqcup\!\sqcup_T y$ is 'almost' $\sqcup\!\sqcup_T$, and hence some results from Section 5.3 are applicable in the case of (θ, sim)-bond-free languages, provided that the relation sim is 'reasonable.'

Theorem 5.16 *([64]) Let* sim *be a rational relation. The following problem is decidable in quadratic time w.r.t.* $|A|$.

Input: An NFA A.

Output: YES/NO, depending on whether $L(A)$ is a (θ, sim)-bond-free language.

For the case where sim is one of the similarity relations $H_{d,k}$ or $Lev_{d,k}$, the algorithm runs at time $\mathcal{O}(dk|A|^2)$ (or $\mathcal{O}(dk^2|A|^2)$, respectively). The (θ, sim)-bond-freeness remains decidable even in the case of context-free tube languages, although the existence of a polynomial time algorithm cannot be guaranteed.

Two problems related to the design of large sets of bond-free molecules, the *optimal negative design problem* and the *maximality problem* have been addressed, too. Both were formally specified in previous sections. The optimal negative design problem remains NP complete even for *finite* tube languages in the case of various similarity metrics. The problem of maximality of *regular* (θ, sim)-bond-free languages has been shown decidable in [64], although the existence of a polynomial-time algorithm cannot be generally guaranteed. However, rather surprisingly, in the important Hamming case such an algorithm exists. Let us consider languages that are subsets of $(\Sigma^k)^+$, for some positive integer k. We call such languages *k-block languages.* Naturally, any regular k-block language can be represented by a special type of lazy DFA [109], which we call a *k-block DFA.*

Theorem 5.17 *([64]) Let d be fixed to be either* 0 *or* 1*. The following problem is computable in a polynomial time.*

Input: k-block DFA A such that $L(A)$ is a $(\theta, H_{d,k})$-bond-free subset of $(\Sigma^k)^+$.

Output: YES/NO, depending on whether the language $L(A)$ is maximal with that property. Moreover, if $L(A)$ is not maximal, output a minimal-length word $w \in (\Sigma^k)^+ \setminus L(A)$ such that $L(A) \cup \{w\}$ is a $(\theta, H_{d,k})$-bond-free subset of $(\Sigma^k)^+$.

In particular, the time complexity $t(|A|)$ is bounded as follows:

$$t(|A|) = \begin{cases} O(k|A|^3), & \textit{if } k \textit{ is odd and } d = 0; \\ O(|A|^6), & \textit{if } k \textit{ is even and } d = 0; \\ O(k^3|A|^6), & \textit{if } d = 1. \end{cases}$$

The concept of (θ, sim)-bond-free languages is quite general and could cover also subtler similarity measures than the Hamming or Levenshtein distance. Consider the original nearest-neighbour thermodynamical approach to the hybridization problem [101]. The calculation of ΔG_{min}, the minimum free energy among the free energies of all possible secondary substructures that may be formed by the examined DNA sequences, is frequently used to determine the most likely secondary structure that will actually form. Assume secondary substructures of a size limited from above (say, of at most 25 bp), which is reasonable from the practical point of view. Let us consider two DNA sequences *similar* if and only if they contain subsequences satisfying the condition $\Delta G_{min} \geq B$, where B is a threshold value for hybridization energy (the 'all or nothing' hybridization model). Such a similarity relation obviously fulfills the conditions of Definition 5.14 and, furthermore, it is rational (even finite). Therefore, for a fixed value of B, the hybridization analysis can possibly benefit from the above mentioned results and rapid algorithms.

We conclude with two examples of a construction of DNA codes in the Hamming case. The first example is based on the method of *k-templates* proposed originally in [7]. This method allows to produce codes which are subsets of $(\Sigma^k)^+$.

Theorem 5.18 *([64]) Let I be a nonempty subset of $\{1, \ldots, k\}$ of cardinality $m = \lfloor k/2 \rfloor + 1 + \lfloor (d + (k \text{ rem } 2))/2 \rfloor$. Then the language K^+ is $(\tau, H_{d,k})$-bond-free, where*

$$K = \{v \in \Sigma^k \mid \text{ if } i \in I \text{ then } v[i] \in \{A, C\}\}.$$

Observe that the size of the code K is $2^m 4^{k-m}$. An advantage of the method of Theorem 5.18 is that we can construct $(\tau, H_{d,k})$-bond-free languages with a large ratio d/k.

Another method is based on the operation of *subword closure* $K^{\otimes}$ of a set $K \subseteq \Sigma^k$:

$$K^{\otimes} = \{w \in \Sigma^* \mid |w| \geq k,\ \mathrm{Sub}_k(w) \subseteq K\}.$$

Denote further $K^{\oplus} \overset{\mathrm{def}}{=} K^{\otimes} \cap (\Sigma^k)^+$. The following theorem characterizes all maximal $(\tau, H_{d,k})$-bond-free subsets of $(\Sigma^k)^+$ and $\Sigma^k\Sigma^*$.

Theorem 5.19 *([64]) The class of all maximal $(\tau, H_{d,k})$-bond-free subsets of $(\Sigma^k)^+$ is finite and equal to*

$$\{K^{\oplus} \mid K \text{ is a maximal } (\tau, H_{d,k})\text{-bond-free subset of } \Sigma^k\}.$$

The above theorem holds also for subsets of $\Sigma^k\Sigma^*$ if we replace $K^{\oplus}$ with $K^{\otimes}$. As a consequence, if one constructs a maximal finite subset K of Σ^k satisfying $\tau(K) \cap H_d(K) = \emptyset$, then the language $K^{\oplus}$ is a maximal $(\tau, H_{d,k})$-bond-free subset of $(\Sigma^k)^+$. Another implication of Theorem 5.19 is that all maximal $(\tau, H_{d,k})$-bond-free subsets of $(\Sigma^k)^+$ or $\Sigma^k\Sigma^*$ are regular.

To conclude, in Section 5 we characterized several basic types of DNA interaction based on Watson-Crick complementarity, using the apparatus of formal language and automata theory. Besides its value as a contribution to theoretical computer science, the main application of this research is the description and construction of sets of DNA molecules (called also DNA codes) which are free of certain types of unwanted binding interactions. These codes are especially useful in DNA computing and many other laboratory techniques which assume that an undesirable hybridization between DNA molecules does not occur. We have shown that a uniform mathematical characterization exists for many types of DNA bonds, both perfect and imperfect with respect to the WK complementarity principle. This characterization, in turn, implies the existence of effective algorithms for manipulation and construction of these DNA codes.

6 Vectorial Models of DNA-Based Information

In Section 5 we saw that formal language theory is a natural tool for modelling, analyzing, and designing "good" DNA codewords which control DNA strand inter- and intra-molecular interactions based on Watson-Crick complementarity. This was achieved by formalizing a DNA single-strand in the 5'- 3' orientation as a linear word over the DNA alphabet $\{A, C, G, T\}$. A limitation of this representation of DNA single strand as words is that it does not model double-stranded DNA molecules, partially double-stranded DNA molecules (such as DNA strands with sticky ends), or interactions between DNA single strands that may lead to double strands.

This section offers an alternative natural way of representing DNA strands, namely as vectors. A (single, partially double-stranded or fully double-stranded) DNA molecule is modelled namely by a vector whose first component is a word over the DNA alphabet representing the "top" strand, and the second component is a word over the DNA alphabet representing the "bottom" strand. Using this representation one can model naturally the operations of annealing and ligation.

We now introduce two computational models that use this representation of DNA strands: a language generating device called *sticker system* (Section 6.1) [44, 73, 93], and its automata counterpart, *Watson-Crick automata* (Section 6.2) [44]. A summary of essential results on these topics can be found in [94].

We start by introducing a vectorial formalism of the notions of DNA single strand, double strand and partial double strand, as well of the bio-operations of annealing (hybridization) and ligation. Other notions and notations we use here have been defined in Section 5.1.

The notion of Watson-Crick complementarity is formalized by a symmetric relation. A relation $\rho \subseteq \Sigma \times \Sigma$ is said to be *symmetric* if for any $a, b \in \Sigma$, $(a, b) \in \rho$ implies $(b, a) \in \rho$. In order to define a symmetric relation ρ, it suffices to specify one of (a, b) and (b, a) as long as we explicitly note that ρ is symmetric.

DNA strands are modelled by 2×1 vectors, wherein the first row corresponds to the "top" DNA strand and the second row corresponds to the "bottom" DNA strand. In this formalism, DNA double strands are modelled as 2×1 vectors in square brackets where the top word is in relation ρ with the bottom word, while DNA single strands are modelled as 2×1 vectors in round brackets where one of the rows is the empty word. More concretely, we write $\begin{bmatrix} a \\ b \end{bmatrix}_\rho$ if two letters a, b are in the relation ρ. Whenever ρ is clear from context, the subscript ρ is omitted. We now define an alphabet of double-stranded columns

$$\Sigma_d = \begin{bmatrix} \Sigma \\ \Sigma \end{bmatrix}_\rho \cup \begin{pmatrix} \Sigma \\ \lambda \end{pmatrix} \cup \begin{pmatrix} \lambda \\ \Sigma \end{pmatrix},$$

where $\begin{bmatrix} \Sigma \\ \Sigma \end{bmatrix}_\rho = \left\{ \begin{bmatrix} a \\ b \end{bmatrix} \;\middle|\; a, b \in \Sigma, (a, b) \in \rho \right\}$, $\begin{pmatrix} \Sigma \\ \lambda \end{pmatrix} = \left\{ \begin{pmatrix} a \\ \lambda \end{pmatrix} \;\middle|\; a \in \Sigma \right\}$, and $\begin{pmatrix} \lambda \\ \Sigma \end{pmatrix} = \left\{ \begin{pmatrix} \lambda \\ b \end{pmatrix} \;\middle|\; b \in \Sigma \right\}$.

The *Watson-Crick domain* associated to Σ and ρ is the set $\mathrm{WK}_\rho(\Sigma)$ defined as $\mathrm{WK}_\rho(\Sigma) = \begin{bmatrix} \Sigma \\ \Sigma \end{bmatrix}_\rho^*$. An element $\begin{bmatrix} a_1 \\ b_1 \end{bmatrix} \begin{bmatrix} a_2 \\ b_2 \end{bmatrix} \cdots \begin{bmatrix} a_n \\ b_n \end{bmatrix} \in \mathrm{WK}_\rho(\Sigma)$ can be written as $\begin{bmatrix} a_1 a_2 \cdots a_n \\ b_1 b_2 \cdots b_n \end{bmatrix}$ succinctly. Note that $\begin{pmatrix} w_1 \\ w_2 \end{pmatrix}$ means no more than a pair of words w_1, w_2, whereas $\begin{bmatrix} w_1 \\ w_2 \end{bmatrix}$ imposes that $|w_1| = |w_2|$ and their corresponding letters are complementary in the sense of the relation ρ. Elements of $\mathrm{WK}_\rho(\Sigma)$ are called *complete double-stranded sequences* or *molecules*. Moreover, we denote $\mathrm{WK}_\rho^+(\Sigma) = \mathrm{WK}_\rho(\Sigma) \setminus \{(\lambda, \lambda)\}$.

Note that elements of $\mathrm{WK}_\rho(\Sigma)$ are fully double-stranded. In most DNA computing experiments, e.g., Adleman's first experiment, partially double-stranded DNA, i.e., DNA strands with sticky ends, are essential. To introduce sticky ends in the model, let $\mathrm{S}(\Sigma) = \left(\begin{smallmatrix} \lambda \\ \Sigma^* \end{smallmatrix} \right) \cup \left(\begin{smallmatrix} \Sigma^* \\ \lambda \end{smallmatrix} \right)$ be the set of sticky ends. Then we define a set $\mathrm{W}_\rho(\Sigma)$ whose elements are molecules with sticky ends at both sides as $\mathrm{W}_\rho(\Sigma) = \mathrm{L}_\rho(\Sigma) \cup \mathrm{R}_\rho(\Sigma) \cup \mathrm{LR}_\rho(\Sigma)$, where

$$\begin{aligned} \mathrm{L}_\rho(\Sigma) &= \mathrm{S}(\Sigma)\mathrm{WK}_\rho(\Sigma) \\ \mathrm{R}_\rho(\Sigma) &= \mathrm{WK}_\rho(\Sigma)\mathrm{S}(\Sigma) \\ \mathrm{LR}_\rho(\Sigma) &= \mathrm{S}(\Sigma)\mathrm{WK}_\rho^+(\Sigma)\mathrm{S}(\Sigma). \end{aligned}$$

Note that unlike an element in $\mathrm{L}_\rho(\Sigma)$ or $\mathrm{R}_\rho(\Sigma)$, elements of $\mathrm{LR}_\rho(\Sigma)$ must have at least one "column" $\begin{bmatrix} a \\ b \end{bmatrix}$. Any element of $\mathrm{W}_\rho(\Sigma)$ with at least a position $\begin{bmatrix} a \\ b \end{bmatrix}$, $a \neq \lambda$, $b \neq \lambda$, is called a *well-started (double stranded) sequence*. Thus, $\mathrm{LR}_\rho(\Sigma)$ is equivalent to the set of all well-started sequences.

Annealing and ligation of DNA molecules can be modelled as a partial operation among elements of $\mathrm{W}_\rho(\Sigma)$. A well-started molecule can be prolonged to the right or to the left with another molecule, provided that their sticky ends match. We define "*sticking* y to the right of x" operation, denoted by $\mu_r(x, y)$. In a symmetric way, $\mu_\ell(y, x)$ (sticking y to the left of x) is defined. Let $x \in \mathrm{LR}_\rho(\Sigma)$ and $y \in \mathrm{W}_\rho(\Sigma)$. Being well-started, $x = x_1 x_2 x_3$ for some

$x_1, x_3 \in \mathrm{S}(\Sigma)$, $x_2 \in \mathrm{WK}_\rho^+(\Sigma)$. Then $\mu_r(x,y)$ is defined as follows (also see Figure 16):

Case A If y is single-stranded, that is, $y \in \mathrm{S}(\Sigma)$, we have the following cases: for $r, p \geq 0$,

1 If $x_3 = \begin{pmatrix} a_1 \cdots a_r \\ \lambda \end{pmatrix}$ and $y = \begin{pmatrix} a_{r+1} \cdots a_{r+p} \\ \lambda \end{pmatrix}$,
then $\mu_r(x,y) = x_1 x_2 \begin{pmatrix} a_1 \cdots a_r a_{r+1} \cdots a_{r+p} \\ \lambda \end{pmatrix}$.

2 If $x_3 = \begin{pmatrix} a_1 \cdots a_r \\ \lambda \end{pmatrix} \begin{pmatrix} a_{r+1} \cdots a_{r+p} \\ \lambda \end{pmatrix}$, $y = \begin{pmatrix} \lambda \\ b_1 \cdots b_r \end{pmatrix}$, and $(a_i, b_i) \in \rho$ for $1 \leq i \leq r$, then $\mu_r(x,y) = x_1 x_2 \begin{bmatrix} a_1 \cdots a_r \\ b_1 \cdots b_r \end{bmatrix} \begin{pmatrix} a_{r+1} \cdots a_{r+p} \\ \lambda \end{pmatrix}$.

3 If $x_3 = \begin{pmatrix} a_1 \cdots a_r \\ \lambda \end{pmatrix}$, $y = \begin{pmatrix} \lambda \\ b_1 \cdots b_r \end{pmatrix} \begin{pmatrix} \lambda \\ b_{r+1} \cdots b_{r+p} \end{pmatrix}$, and $(a_i, b_i) \in \rho$ for $1 \leq i \leq r$, then $\mu_r(x,y) = x_1 x_2 \begin{bmatrix} a_1 \cdots a_r \\ b_1 \cdots b_r \end{bmatrix} \begin{pmatrix} \lambda \\ b_{r+1} \cdots b_{r+p} \end{pmatrix}$.

4 The counterparts of cases A1 – A3 where the roles of upper and lower strands are reversed.

Case B If y is well-started (partially double-stranded), that is, $y = y_1 y_2 y_3$ for some $y_1, y_3 \in \mathrm{S}(\Sigma)$ and $y_2 \in \mathrm{WK}_\rho^+(\Sigma)$: for $r \geq 0$,

1 If $x_3 = \begin{pmatrix} a_1 \cdots a_r \\ \lambda \end{pmatrix}$, $y_1 = \begin{pmatrix} \lambda \\ b_1 \cdots b_r \end{pmatrix}$, and $(a_i, b_i) \in \rho$ for $1 \leq i \leq r$, then $\mu_r(x,y) = x_1 x_2 \begin{bmatrix} a_1 \cdots a_r \\ b_1 \cdots b_r \end{bmatrix} y_2 y_3$.

2 The counterpart of case B1 when the roles of upper and lower strands are reversed.

If none of these cases applies, $\mu_r(x,y)$ is undefined. Note that in all cases, r can be 0, that is, the system is allowed to prolong the blunt ends of molecules. Moreover, we have $\mu_r(x, \binom{\lambda}{\lambda}) = \mu_\ell(\binom{\lambda}{\lambda}, x) = x$ for any $x \in \mathrm{LR}_\rho(\Sigma)$.

Note also that this vectorial model of DNA molecules allows – unlike its linear counterpart presented in the previous section – the differentiation between single-stranded DNA molecules, double-stranded DNA molecules, and DNA molecules with sticky ends, as well as for modelling of DNA-DNA interactions such as annealing and ligation.

6.1 Sticker Systems

Sticker systems are formal models of molecular interactions occurring in DNA computing, based on the sticking operation. Several variants of sticker systems have been defined in the literature. In this section we describe the simple regular sticker system, which is the most realistic variant but which is weak in terms

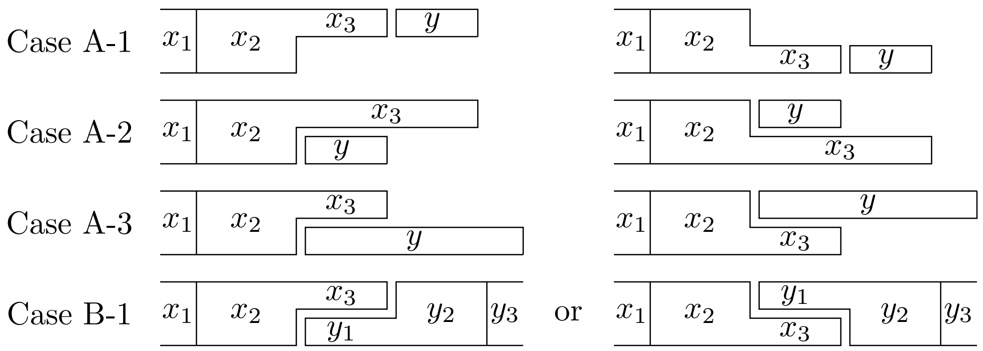

Figure 16: Sticking operations: prolongation of a well-started molecule to the right.

of generating capacity. We also introduce two practical ways to strengthen this variant: by using some complex DNA structures (Section 6.1.1), and by observing the sticker system externally (Section 6.1.2). Both enhance the computational power of the sticker system to Turing universality.

A sticker system prolongs given well-started DNA molecules, both to the left and to the right, by using the sticking operation, so as to turn them into complete double-stranded molecules. This system is an appropriate model of molecular interaction occurring, e.g., in Adleman's 1994 experiment, and was proposed under the name *bidirectional sticker system* [43].

A (bidirectional) *sticker system over a relation* ρ is a 4-tuple $\gamma = (\Sigma, \rho, A, P)$, where Σ is an alphabet endowed with the symmetric relation $\rho \subseteq \Sigma \times \Sigma$, $A \subseteq \mathrm{LR}_\rho(\Sigma)$ is a finite subset of well-started sequences (axioms), and P is a finite subset of $\mathrm{W}_\rho(\Sigma) \times \mathrm{W}_\rho(\Sigma)$. Starting from an axiom in A, the system prolongs it with using a pair in P as follows:

$$x \Rightarrow_\gamma w \text{ iff } w = \mu_r(\mu_\ell(y, x), z) \text{ for some } (y, z) \in P.$$

In other words, $x \Rightarrow_\gamma w$ iff sticking y to the left and z to the right of x results in w. The reflexive and transitive closure of $\Rightarrow_\gamma$ is denoted by $\Rightarrow_\gamma^*$. A sequence $x_1 \Rightarrow_\gamma x_2 \Rightarrow_\gamma \ldots \Rightarrow_\gamma x_k$, $x_1 \in A$, is called a *computation* in γ (of length $k-1$). A computation as above is *complete* if $x_k \in \mathrm{WK}_\rho^+(\Sigma)$.

The set of all molecules over Σ generated by complete computations in γ is defined as: $LM(\gamma) = \{w \in \mathrm{WK}_\rho^+(\Sigma) \mid x \Rightarrow_\gamma^* w, x \in A\}$. We can also consider the sticker systems as a generator of languages of strings rather than double-stranded molecules. To this aim, the following language is associated with $LM(\gamma)$:

$$L(\gamma) = \left\{ w \in \Sigma^* \;\middle|\; \begin{bmatrix} w \\ w' \end{bmatrix}_\rho \in LM(\gamma) \text{ for some } w' \in \Sigma^* \right\}.$$

A language L is called a *sticker language* if there exists a sticker system γ such that $L(\gamma) = L$.

A sticker system $\gamma = (\Sigma, \rho, A, P)$ is said to be *simple* (respectively *regular*) if for each pair $(y, z) \in P$, $y, z \in \mathrm{S}(\Sigma)$ (respectively $y = \lambda$). A simple regular system extends either the upper or lower strand, one at a time (hence the attribute "simple"), and only to the right (hence the attribute "regular"). Thus, a simple regular sticker system can be rewritten as 5-tuple $(\Sigma, \rho, A, D_u, D_\ell)$, where D_u and D_ℓ are finite subsets of $\binom{\Sigma^*}{\lambda}$ and $\binom{\lambda}{\Sigma^*}$, respectively. The family of languages generated by simple regular sticker systems is denoted by $\mathrm{SRSL}(n)$, where n means "no-restriction". This variant is the most precise and realistic model of the annealing/ligation-based hybridization occurring in the Adleman's experiment. In fact, since being proposed in [73], this type of sticker system has been intensively investigated.

A normal form for sticker systems with respect to the relation ρ was introduced in [55] for simple regular variants, and in [77] for general sticker systems. It says that the identity relation *id* suffices to generate any sticker language.

Theorem 6.1 *([55], [77]) For a sticker system γ over a relation ρ, one can construct a sticker system γ' over the identity relation id such that $L(\gamma) = L(\gamma')$.*

Proof. The ideas proposed in [55] and [77] are essentially the same; they work for arbitrary bidirectional sticker systems. Here we present their proofs applied to *regular* (uni-directional) sticker systems to suggest the fact that the identity relation suffices also for Watson-Crick automata introduced later.

Let $\gamma = (\Sigma, \rho, A, P_r)$ be a regular sticker system. We construct a regular sticker system $\gamma' = (\Sigma, id, A', P_r')$, Figure 17, where

$$\begin{aligned} A' &= \left\{ \binom{x_0}{z_0} \;\middle|\; \binom{x_0}{y_0} \in A \text{ for some } y_0 \text{ such that } \begin{bmatrix} z_0 \\ y_0 \end{bmatrix}_\rho \right\}, \\ P_r' &= \left\{ \binom{x_i}{z_i} \;\middle|\; \binom{x_i}{y_i} \in P_r \text{ for some } y_i \text{ such that } \begin{bmatrix} z_i \\ y_i \end{bmatrix}_\rho \right\}. \end{aligned}$$

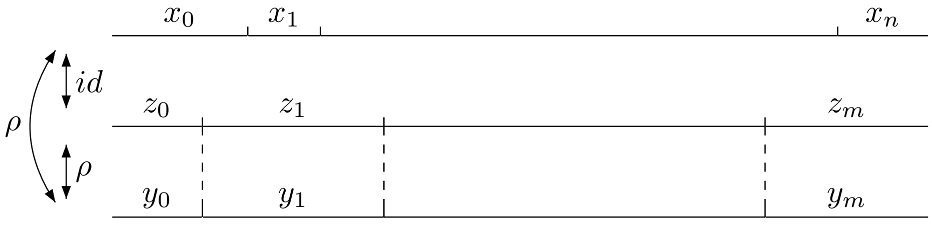

Figure 17: The idea of the proof of Theorem 6.1. For any sticker system γ over a relation ρ, one can construct a sticker system γ' over the identity relation, such that $L(\gamma) = L(\gamma')$. The newly-constructed sticker system γ', based on *id*, simulates the process of γ to generate $(x, y) \in \mathrm{WK}_\rho^+(\Sigma)$ where $x = x_0 x_1 \cdots x_n$ and $y = y_0 y_1 \cdots y_m$, by generating $(x, z) \in WK_{id}^+(\Sigma)$, where $z = z_0 z_1 \cdots z_m$.

Assume that $x \in L(\gamma)$, i.e., there is a word y such that $(x, y) \in \mathrm{WK}^{+}_{\rho}(\Sigma)$, $x = x_0 x_1 \cdots x_n$ and $y = y_0 y_1 \cdots y_m$, where $(x_0, y_0) \in A$, and $(x_i, y_j) \in P'_r$ $(1 \leq j \leq m)$. Due to the symmetric property of ρ, x can also be written as the catenation of words z_0 and $z_1, \ldots, z_m \in \Sigma^*$ such that $(z_k, y_k) \in \mathrm{WK}_{\rho}(\Sigma)$ $(0 \leq k \leq m)$. According to the definition of A', $(x_0, z_0) \in A'$, and $(x_j, z_j) \in P'_r$ $(1 \leq j \leq m)$. As a result, $\begin{bmatrix} x_0 x_1 \cdots x_n \\ z_0 z_1 \cdots z_m \end{bmatrix}_{id} = \begin{bmatrix} x \\ x \end{bmatrix}_{id} \in LM(\gamma')$, and hence $x \in L(\gamma')$. The proof that $L(\gamma') \subseteq L(\gamma)$ is similar. □

As mentioned in the previous proof, this theorem proved to be valid for general sticker systems in [77]. Moreover, in the paper, the authors show that an analogous result holds even for Watson-Crick automata, that is, the identity relation suffices for WK-automata. From a historical viewpoint of theory of computation, the normal forms for grammars and acceptors have proved useful tools. Analogously, Theorem 6.1 will be useful for several proofs in the rest of this section.

The simple regular sticker system is one of the most "natural" computational models for annealing/ligation-based hybridization. Kari et al. initiated in [73] an investigation into the generative capacity of general sticker systems, including the simple regular variant, and the investigation continued in [43], [93]. The conclusion was that some classes of sticker systems can even characterize the recursively enumerable languages. On the contrary, the simple regular variant turned out to be quite weak.

Theorem 6.2 *([73], [94])* $\mathrm{SRSL}(n) \subsetneq \mathrm{REG} = \mathrm{Cod}(\mathrm{SRSL}(n))$.

Thus, this "natural variant" of sticker systems has no more generative power than finite automata, even with the aid of encoding.

In [73], the notion of *fair computation* was proposed. Let γ be a simple regular sticker system. A complete computation in γ is said to be *fair* if through the computation, the number of extensions occurring on the upper strand is equal to the number of extensions occurring on the lower strand. A language L is called a *fair sticker language* if there exists a simple regular sticker system γ such that L is the set of all words which are generated by fair computations in γ. The family of fair sticker languages is denoted by $\mathrm{SRSL}(f)$.

It is known that $\mathrm{REG} \subsetneq \mathrm{Cod}(\mathrm{SRSL}(f))$ and also that we cannot obtain characterizations of RE starting from languages in $\mathrm{SRSL}(f)$ and using an arbitrary generalized sequential machine *(gsm)* mapping, including a coding (see [94]). Hence the question of whether $\mathrm{SRSL}(f)$ is included in CF (or even in LIN) arose. The answer to these question was obtained in [55]. Firstly, we introduce their example to show that $\mathrm{SRSL}(f) \not\subseteq \mathrm{LIN}$.

Example 1 Let $\gamma = (\{a, b, c, d\}, \rho, A, D_u, D_\ell)$ be a simple regular sticker system, where $\rho = \{(a, a), (b, b), (b, c), (d, d)\}$, $A = \left\{ \begin{bmatrix} d \\ d \end{bmatrix} \right\}$, $D_u = \left\{ \begin{pmatrix} aa \\ \lambda \end{pmatrix}, \begin{pmatrix} b \\ \lambda \end{pmatrix} \right\}$, and $D_\ell = \left\{ \begin{pmatrix} \lambda \\ a \end{pmatrix}, \begin{pmatrix} \lambda \\ bc \end{pmatrix} \right\}$. This is a technical modification of Example

2 from [55] in order to make an axiom well-started. Then $LM_n(\gamma) = \begin{bmatrix} d \\ d \end{bmatrix} \left\{ \begin{bmatrix} aa \\ aa \end{bmatrix}, \begin{bmatrix} bb \\ bc \end{bmatrix} \right\}^*$, and hence $L_n(\gamma) = d\{aa, bb\}^*$, $L_f(\gamma) = \{x \in L_n(\gamma) \mid \#_a(x) = \#_b(x)\}$. The pumping lemma for linear languages can be used to prove that $L_f(\gamma)$ is not linear.

Theorem 6.3 *([55])* $\mathrm{SRSL}(f) \subsetneq \mathrm{Cod}(\mathrm{SRSL}(f)) \subsetneq \mathrm{CF}$.

Kari et al. imposed an additional constraint on fair computations called *coherence* in [73], the use of which leads to a representation of RE. In the rest of this section, we introduce two new approaches to the problem of how to obtain characterizations of RE by using sticker systems augmented with more practical assumptions. Regarding the generative capacity and further topics about sticker systems, the reader is referred to the thorough summary in Chapter 4 of [94].

6.1.1 Sticker Systems with Complex Structures

Sakakibara and Kobayashi [99] proposed a novel use of stickers, that involved the formation of *DNA hairpins*. In Section 5, a hairpin was modelled as a linear word $w_1a_1 \cdots a_nw_2b_n \cdots b_1w_3$, where $(a_i, b_i) \in \rho$, $1 \le i \le n$. Here we represent the same hairpin vectorially as $\begin{pmatrix} \langle w_2 \rangle \\ w_1|w_3 \end{pmatrix} \in \begin{pmatrix} \Sigma^* \\ \Sigma^* \end{pmatrix}$ as shown in Figure 18. This hairpin-shaped molecule may stick to other molecules by its two sticky ends w_1 and w_3 or by its loop part w_2. The sets of this type and inverted type of hairpins are denoted by $T_u(\Sigma)$ and $T_\ell(\Sigma)$, respectively, i.e.,

$$T_u(\Sigma) = \left\{ \begin{pmatrix} \langle w_2 \rangle \\ w_1|w_3 \end{pmatrix} \;\middle|\; w_1, w_2, w_3 \in \Sigma^* \right\}, T_\ell(\Sigma) = \left\{ \begin{pmatrix} w_1|w_3 \\ \langle w_2 \rangle \end{pmatrix} \;\middle|\; w_1, w_2, w_3 \in \Sigma^* \right\},$$

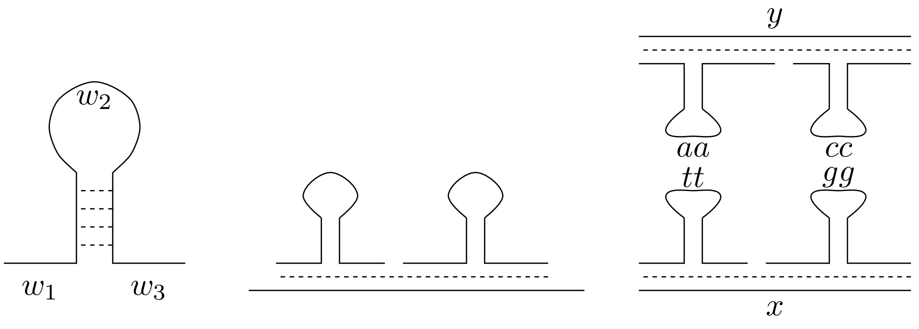

Figure 18: (Left) A hairpin which the word $w_1a_1 \cdots a_nw_2b_n \cdots b_1w_3$ may form if $(a_i, b_i) \in \rho$, $1 \le i \le n$; the sticky ends w_1, w_3 or the hairpin loop w_2 can bind to other molecules; (Middle) Two hairpins stick to the lower strand via their sticky ends, leaving sticky ends at both ends of the lower strand; (Right) Two "complete" molecules can bind together via their hairpin loops, if they are matched as shown.

The operation of “sticking a hairpin $x = \begin{pmatrix}\langle w_2\rangle \\ w_1|w_3\end{pmatrix}$ onto a single-stranded molecule y” is defined whenever $y = y_1y_2y_3$, and y_2 is complementary to w_1w_3 as follows:

$$\mu(y,x) = \left(\begin{pmatrix}\lambda \\ y_1\end{pmatrix} \quad \begin{matrix}\langle w_2\rangle \\ \begin{bmatrix} w_1|w_3 \\ y_2\end{bmatrix}\end{matrix} \quad \begin{pmatrix}\lambda \\ y_3\end{pmatrix} \right).$$

Moreover, this operation $\mu(y,x)$ is extended to the general case; for a molecule,

$$z = \left(\begin{pmatrix}\lambda \\ z_0\end{pmatrix} \quad \begin{matrix}\langle u_1\rangle \\ \begin{bmatrix} t_1|v_1 \\ z_1\end{bmatrix}\end{matrix} \quad \begin{matrix}\langle u_2\rangle \\ \begin{bmatrix} t_2|v_2 \\ z_2\end{bmatrix}\end{matrix} \quad \begin{matrix}\cdots \\ \cdots\end{matrix} \quad \begin{matrix}\langle u_n\rangle \\ \begin{bmatrix} t_n|v_n \\ z_n\end{bmatrix}\end{matrix} \quad \begin{pmatrix}\lambda \\ z_{n+1}z_{n+2}\end{pmatrix} \right)$$

such that w_1w_3 is complementary to z_{n+1}, $\mu(z,x)$ is defined as:

$$\mu(z,x) = \left(\begin{pmatrix}\lambda \\ z_0\end{pmatrix} \quad \begin{matrix}\langle u_1\rangle \\ \begin{bmatrix} t_1|v_1 \\ z_1\end{bmatrix}\end{matrix} \quad \begin{matrix}\langle u_2\rangle \\ \begin{bmatrix} t_2|v_2 \\ z_2\end{bmatrix}\end{matrix} \quad \begin{matrix}\cdots \\ \cdots\end{matrix} \quad \begin{matrix}\langle u_n\rangle \\ \begin{bmatrix} t_n|v_n \\ z_n\end{bmatrix}\end{matrix} \quad \begin{matrix}\langle w_2\rangle \\ \begin{bmatrix} w_1|w_3 \\ z_{n+1}\end{bmatrix}\end{matrix} \quad \begin{pmatrix}\lambda \\ z_{n+2}\end{pmatrix} \right)$$

Thus, this sticking operation forms “multiple hairpins” structures as illustrated in Figure 18 (Middle). The set of molecules with this type of structures and the set of molecules with inverted structures are denoted by $\mathrm{TW}_u(\Sigma)$ and $\mathrm{TW}_\ell(\Sigma)$, respectively, that is,

$$\mathrm{TW}_u(\Sigma) = \begin{pmatrix}\lambda \\ \lambda \\ \Sigma^*\end{pmatrix} \left(\begin{matrix}\langle \Sigma^*\rangle \\ \begin{bmatrix}\Sigma^* \\ \Sigma^*\end{bmatrix}\end{matrix} \right)^* \begin{pmatrix}\lambda \\ \lambda \\ \Sigma^*\end{pmatrix}, \quad \mathrm{TW}_\ell(\Sigma) = \begin{pmatrix}\Sigma^* \\ \lambda \\ \lambda\end{pmatrix} \left(\begin{matrix}\begin{bmatrix}\Sigma^* \\ \Sigma^*\end{bmatrix} \\ \langle \Sigma^*\rangle\end{matrix} \right)^* \begin{pmatrix}\Sigma^* \\ \lambda \\ \lambda\end{pmatrix}$$

Similarly the hybridization operation $\mu(z,x)$ is defined for $z \in \mathrm{TW}_\ell(\Sigma)$ and $x \in T_\ell(\Sigma)$. Note that $\begin{pmatrix}\lambda \\ \lambda \\ x\end{pmatrix} \in \mathrm{TW}_\ell(\Sigma)$ or $\begin{pmatrix}x \\ \lambda \\ \lambda\end{pmatrix} \in \mathrm{TW}_u(\Sigma)$ can be regarded as a word $x \in \Sigma^*$. Hence we will give also a word as a first argument of μ in the following. Now we can define another type of “complete” molecules, namely elements in the set $\mathrm{TWK}(\Sigma) = \mathrm{TWK}_u(\Sigma) \cup \mathrm{TWK}_\ell(\Sigma)$, where

$$\mathrm{TWK}_u(\Sigma) = \left(\begin{matrix}\langle \Sigma^*\rangle \\ \begin{bmatrix}\Sigma^* \\ \Sigma^*\end{bmatrix}\end{matrix} \right)^*, \qquad \mathrm{TWK}_\ell(\Sigma) = \left(\begin{matrix}\begin{bmatrix}\Sigma^* \\ \Sigma^*\end{bmatrix} \\ \langle \Sigma^*\rangle\end{matrix} \right)^*.$$

Secondly, we consider another sticking operation for the loop (w_2 of x) of a hairpin, which is illustrated in Figure 18 (Right). Consider two complete molecules $x \in \mathrm{TWK}_u(\Sigma)$ and $y \in \mathrm{TWK}_\ell(\Sigma)$ defined as:

$$x = \left(\begin{matrix}\langle s_1\rangle \\ \begin{bmatrix} r_1|t_1 \\ x_1\end{bmatrix}\end{matrix} \quad \begin{matrix}\langle s_2\rangle \\ \begin{bmatrix} r_2|t_2 \\ x_2\end{bmatrix}\end{matrix} \quad \begin{matrix}\cdots \\ \cdots\end{matrix} \quad \begin{matrix}\langle s_n\rangle \\ \begin{bmatrix} r_n|t_n \\ x_n\end{bmatrix}\end{matrix} \right), \qquad y = \left(\begin{matrix}\begin{bmatrix} y_1 \\ u_1|w_1\end{bmatrix} \\ \langle v_1\rangle\end{matrix} \quad \begin{matrix}\begin{bmatrix} y_2 \\ u_2|w_2\end{bmatrix} \\ \langle v_2\rangle\end{matrix} \quad \begin{matrix}\cdots \\ \cdots\end{matrix} \quad \begin{matrix}\begin{bmatrix} y_n \\ u_n|w_n\end{bmatrix} \\ \langle v_n\rangle\end{matrix} \right).$$

When s_i is complementary to v_i $(1 \leq i \leq n)$, $\phi(x, y)$ is defined as:

$$\phi(x,y) = \begin{pmatrix} \begin{bmatrix} y_1 \\ u_1|w_1 \end{bmatrix} & \begin{bmatrix} y_2 \\ u_2|w_2 \end{bmatrix} & \cdots & \begin{bmatrix} y_n \\ u_n|w_n \end{bmatrix} \\ \begin{bmatrix} \langle v_1 \rangle \\ \langle s_1 \rangle \end{bmatrix} & \begin{bmatrix} \langle v_2 \rangle \\ \langle s_2 \rangle \end{bmatrix} & \cdots & \begin{bmatrix} \langle v_n \rangle \\ \langle s_n \rangle \end{bmatrix} \\ \begin{bmatrix} r_1|t_1 \\ x_1 \end{bmatrix} & \begin{bmatrix} r_2|t_2 \\ x_2 \end{bmatrix} & \cdots & \begin{bmatrix} r_n|t_n \\ x_n \end{bmatrix} \end{pmatrix}.$$

Thus, for two complete molecules $x \in \mathrm{TWK}_u(\Sigma)$ and $y \in \mathrm{TWK}_\ell(\Sigma)$, $\phi(x, y)$ is well-defined to be an element of the set

$$\mathrm{DTWK}(\Sigma) = \begin{pmatrix} \begin{bmatrix} \Sigma^* \\ \Sigma^* \end{bmatrix} \\ \begin{bmatrix} \langle \Sigma^* \rangle \\ \langle \Sigma^* \rangle \end{bmatrix} \\ \begin{bmatrix} \Sigma^* \\ \Sigma^* \end{bmatrix} \end{pmatrix}^*.$$

A *sticker system with complex structures* is a 4-tuple $\gamma = (\Sigma, \rho, D_u, D_\ell)$, where D_u and D_ℓ are finite subsets of $T_u(\Sigma)$ and $T_\ell(\Sigma)$, respectively. For molecules $x, y \in \mathrm{TW}_u(\Sigma)$, we write $x \Rightarrow_u y$ if $y = \mu(x, v)$ for some $v \in D_u$. Analogously, for $x', y' \in \mathrm{TW}_\ell(\Sigma)$, we write $x' \Rightarrow_\ell y'$ if $y' = \mu(x', v')$ for some $v' \in D_\ell$. The reflexive and transitive closures of these operations are denoted by $\Rightarrow_u^*$ and $\Rightarrow_\ell^*$.

A sequence $x_1 \Rightarrow_\alpha x_2 \Rightarrow_\alpha \cdots \Rightarrow_\alpha x_k$, with $x_1 \in \Sigma^*$ and $\alpha \in \{u, \ell\}$, is called a computation in γ. (In this context, x_1, the start of the computation, can be regarded as either a word in Σ^* or an element of $\mathrm{TW}_\alpha(\Sigma)$.) A computation $x_1 \Rightarrow_\alpha^* x_k$ is said to be *complete* when $x_k \in \mathrm{TWK}_\alpha(\Sigma)$. Suppose that we have two complete computations $x \Rightarrow_u^* y$ and $x \Rightarrow_\ell^* z$ for a word $x \in \Sigma^*$. When $\phi(y, z)$ becomes a complete matching, that is, $\phi(y, z) \in \mathrm{DTWK}(\Sigma)$, this computation process is said to be *successful.* The following language is associated with γ:

$$\begin{aligned} L(\gamma) \quad = \quad & \{x \in (\Sigma \setminus \{\#\})^* \mid x\# \Rightarrow_u^* y,\, y \in \mathrm{TWK}_u(\Sigma), \\ & \qquad x\# \Rightarrow_\ell^* z,\, z \in \mathrm{TWK}_\ell(\Sigma), \text{ and } \phi(y, z) \in \mathrm{DTWK}(\Sigma)\}. \end{aligned}$$

Thus, we can consider this system as a language accepting device. The family of languages accepted by sticker systems with complex structures is denoted by SLDT.

Now we show that the use of hairpins enables us to characterize RE based on the following lemma.

Lemma 6.1 *([73]) For each recursively enumerable language $L \subseteq \Sigma^*$, there exist two λ-free morphisms $h_1, h_2 : \Sigma_2^* \to \Sigma_1^*$, a regular language $R \subseteq \Sigma_1^*$, and a projection $pr_\Sigma : \Sigma_1^* \to \Sigma^*$ such that $L = pr_\Sigma(h_1(\mathrm{EQ}(h_1, h_2)) \cap R)$.*

Theorem 6.4 *([99]) Every recursively enumerable language is the weak coding of a language in the family* SLDT.

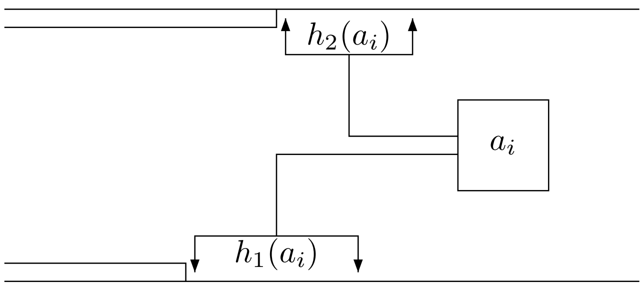

Figure 19: A brief sketch of the proof for Theorem 6.4. Two hairpins, one with sticky end $h_1(a_i)$ and the other with sticky end $h_2(a_i)$, binding together via their matching loops, can be abstracted as a finite control (square) labelled by a_i, with two heads (bifurcated allows). The control checks for each i, $1 \leq i \leq n$, if the single-stranded part of the upper strand begins with $h_2(a_i)$, and the lower strand begins with $h_1(a_i)$; if so, then the corresponding hairpins are sticked. This process is repeated until either this system cannot proceed in this way any more or it generates a complete matching of two complete molecules. In fact this system is essentially equivalent to Watson-Crick automata, which will be introduced in Section 6.2.

Proof. Let $L \in$ RE. Due to Lemma 6.1, L can be obtained from $h_1(\mathrm{EQ}(h_1, h_2)) \cap R$ by a projection, where $h_1, h_2 : \Sigma_2^* \to \Sigma_1^*$ are λ-free morphisms, and $R \in$ REG. Hence it suffices to construct a sticker system with complex structures γ which accepts an *encoding* of $h_1(\mathrm{EQ}(h_1, h_2)) \cap R$. Consider a complete deterministic finite automaton $M = (Q, \Sigma_1, \delta, q_0, F)$ for R, where $Q = \{q_0, q_1, \ldots, q_m\}$. Any word $w = b_1 b_2 \cdots b_k$ ($b_i \in \Sigma_1$) is encoded uniquely as $q_{l_0} b_1 q_{l_1} q_{l_1} b_2 q_{l_2} \cdots q_{l_{k-1}} b_k q_{l_k} q_{l_k}$, where $q_{l_0} = q_0$, $q_{l_1}, \ldots, q_{l_k} \in Q$ such that $\delta(q_{l_{j-1}}, b_j) = q_{l_j}$ for $1 \leq j \leq k$. So $w \in R$ iff $q_{l_k} \in F$.

Let $\Sigma_2 = \{a_1, \ldots, a_n\}$, and for each a_i, let $h_1(a_i) = c_1 c_2 \cdots c_{k_i}$ ($c_j \in \Sigma_1$). Note that for an arbitrary state in Q, there is a unique transition on M by $h_1(a_1)$ because M is a complete deterministic automaton. Thus, a set of encodings of all such transitions is defined for each $a_i \in \Sigma_2$ as follows:

$$T_1(h_1(a_i)) \quad = \quad \bigcup_{q_{l_0} \in Q} \{q_{l_0} b_1 q_{l_1} \cdots q_{l_{k_i - 1}} b_{k_i} q_{l_{k_i}} \mid \delta(q_{l_{j-1}}, b_j) = q_{l_j},\ 1 \leq j \leq k_i\}.$$

Following the same idea, $T_2(h_2(a_i))$ is defined for each $a_i \in \Sigma_2$. Now γ is constructed as $(\Sigma_1 \cup \Sigma_2 \cup Q \cup \{\#\}, id, D_u \cup D_\ell)$, where

$$\begin{aligned} D_\ell &= \left\{ \begin{pmatrix} t_2 \\ \langle a_i \rangle \end{pmatrix} \middle| \ t_2 \in T_2(h_2(a_i)) \right\} \cup \left\{ \begin{pmatrix} q_f\# \\ \langle \# \rangle \end{pmatrix} \middle| \ q_f \in F \right\}, \\ D_u &= \left\{ \begin{pmatrix} \langle a_i \rangle \\ t_1 \end{pmatrix} \middle| \ t_1 \in T_1(h_1(a_i)) \right\} \cup \left\{ \begin{pmatrix} \langle \# \rangle \\ q_f\# \end{pmatrix} \middle| \ q_f \in F \right\}. \end{aligned}$$

Figure 19 illustrates the idea of how γ recognizes the language $h_1(\mathrm{EQ}(h_1, h_2))$ (the encoding mentioned above for R is omitted for clarity).

By this construction, $L(\gamma)$ is the set of encodings of words $u \in R$ for which there exists a word $w \in \Sigma_2^*$ with $h_1(w) = h_2(w) = u$. Projection being a weak coding, there exists a weak coding h such that $h(L(\gamma)) = L$. □

6.1.2 Observable Sticker Systems

Another idea to strengthen a computational system is to let someone observe and report how the system works step by step. This composite system, inspired by the common practice of observing the progress of a biology or chemistry experiment, has been introduced in [15] to "observe" membrane systems. There, a finite automaton observed the change of configurations of a "computationally weak" membrane system (with context-free power). Surprisingly this composite system proved to be universal. Following this idea, many computations were "observed": splicing systems, derivations of grammars and string-rewriting systems, and also sticker systems. For the details of these observations as well as the formal definition of computation by observation in general, the readers are referred to [15], [14], and references thereof.

The idea of observing sticker systems was introduced in [3]. Informally, an *observable sticker system* is composed of a "computationally weak" sticker system and an external observer. Observing the computation of a sticker system, starting from an axiom, the observer notes – at each computational step – the current configuration, and processes it according to its own rules, producing an output. The catenation of all the outputs thus produced by the observer during a complete computation, constitutes a word in the language of the observable sticker system. The collection of all words thus obtained is the language generated by this observable sticker system.

Formally speaking, configurations of a sticker system, which are elements of Σ_d^*, are observed so that an observer is implemented as a finite automaton which works on elements of Σ_d^*. This automaton is defined as a 6-tuple $O = (Q, \Sigma_d, \Delta, q_0, \delta, \sigma)$, where a finite set of states Q, an input alphabet Σ_d, the initial state $q_0 \in Q$, and a complete transition function $\delta : \Sigma_d \times Q \to Q$ are defined as usual for conventional finite automata: whereas Δ is an output alphabet and σ is a labelling function $Q \to \Delta \cup \{\perp, \lambda\}$, $\perp$ being a special symbol.

An *observable (simple regular) sticker system* is a pair $\phi = (\gamma, O)$ for a simple regular sticker system γ and an observer O. For a computation $c : x_0 \Rightarrow_\gamma x_1 \Rightarrow_\gamma \cdots \Rightarrow_\gamma x_k$ $(x_i \in \mathrm{R}_\rho(\Sigma))$, $O(c)$ is defined as $O(x_0)O(x_1) \cdots O(x_k)$. The language $L(\phi)$ generated by ϕ is defined as $L(\phi) = \{O(c) \mid c \text{ is a complete computation by } \gamma\}$.

Theorem 6.5 *([3]) There exists an observable simple regular sticker system which generates a non-context-free language.*

This theorem shows how stronger very restricted sticker systems can get with the aid of the observer (cf. Theorem 6.2). A natural question which follows is of how to get universality within the framework of observable sticker systems.

The next theorem proves that observers with the capability to discard any "bad" evolution endow simple regular sticker systems with universality.

The symbol $\perp \notin \Delta$ makes it possible for an observer O to distinguish bad evolutions by a sticker system γ from good ones in a way that the observation of bad evolutions leads to the output of $\perp$. For the observable sticker system $\phi = (\gamma, O)$, we can weed out any word which contains $\perp$ from $L(\phi)$ by taking $\widehat{L}(\phi) = L(\phi) \cap \Delta^*$.

Theorem 6.6 *([3]) For each $L \in \mathrm{RE}$, there exists an observable simple regular sticker system $\phi = (\gamma, O)$ such that $\widehat{L}(\phi) = L$.*

Due to the fact that recursive languages are closed under intersection with regular languages, Theorem 6.6 has the following result as its corollary.

Corollary 6.1 *([3]) There exists an observable simple regular sticker system $\phi = (\gamma, O)$ such that $L(\phi)$ is a non-recursive language.*

6.2 Watson-Crick Automata

While sticker systems *generate* complete double-stranded molecules by using the sticking operation, their *accepting* counterparts, the Watson-Crick automata parse a given complete double-stranded molecule and determine whether the input is accepted or not. A Watson-Crick automaton is equipped with a finite state machine with two heads. This machine has its heads read the respective upper and lower strands of a given complete double-stranded molecule simultaneously, and changes its state accordingly. The basic idea of how Watson-Crick automata work was described in Figure 19.

In parallel to the research on sticker systems, these biologically-inspired automata have been intensively investigated within the last decade. Early studies including, e.g., [44, 85] investigate variants of WK-automata, relationships among them with respect to generative capacity, and universal Watson-Crick automata, topics summarized in Chapter 5 of [94]. A more recent survey [22] includes results on complexity measures [92] and Watson-Crick automata systems [21, 20]. Other studies on WK-automata comprise Watson-Crick ω-automata [90], the role of the complementarity relation [77] (see Theorem 6.1), local testability and regular reversibility [102, 103], 5' → 3' sensing Watson-Crick finite automata [89], and deterministic Watson-Crick automata [23].

In this section we present recent results in this field, such as studies of deterministic WK-automata, and the role of the complementarity relation. In particular, deterministic WK-automata are essential for the design of efficient molecular parsers.

A *(non-deterministic) Watson-Crick (finite) automaton over a symmetric relation* $\rho \subseteq \Sigma \times \Sigma$ is a 6-tuple $M = (\Sigma, \rho, Q, q_0, F, \delta)$, where Σ, Q, q_0, and F are defined in the same manner as for finite automata. The *transition function* δ is a mapping $\delta : Q \times \Sigma^* \times \Sigma^* \to 2^Q$ such that $\delta(q, (\begin{smallmatrix} w_1 \\ w_2 \end{smallmatrix})) \neq \emptyset$ only for *finitely many* pairs $(q, w_1, w_2) \in Q \times \Sigma^* \times \Sigma^*$. We can replace the transition function by rewriting rules, by denoting $q\,\binom{w_1}{w_2} \to q'$ instead of $q' \in \delta(q, \binom{w_1}{w_2})$. Transitions in M are defined as follows. For $q, q' \in Q$ and $\binom{x_1}{x_2}, \binom{w_1}{w_2}, \binom{y_1}{y_2} \in \binom{\Sigma^*}{\Sigma^*}$ such that $\begin{bmatrix} x_1w_1y_1 \\ x_2w_2y_2 \end{bmatrix} \in \mathrm{WK}_\rho(\Sigma)$, we write $\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} q \begin{pmatrix} w_1 \\ w_2 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} \Rightarrow \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \begin{pmatrix} w_1 \\ w_2 \end{pmatrix} q' \begin{pmatrix} y_1 \\ y_2 \end{pmatrix}$ iff $q' \in \delta(q, \binom{w_1}{w_2})$. If $\Rightarrow^*$ is the reflexive and transitive closure of $\Rightarrow$, then the language accepted by M is:

$$L(M) \quad = \quad \left\{ w_1 \in \Sigma^* \;\middle|\; q_0 \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} \Rightarrow^* \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} q_f \text{ for some } q_f \in F \text{ and } w_2 \in \Sigma^* \text{ such that } \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} \in \mathrm{WK}_\rho(\Sigma) \right\}.$$

Other languages are also considered in [44, 94] such as control words associated to computations - but we do not introduce them here. By convention, as suggested also in [92], in this section we consider two languages differing only by the empty word λ as identical.

A WK-automaton $M = (\Sigma, \rho, Q, q_0, F, \delta)$ is said to be: *stateless* if $Q = F = \{s_0\}$; *all-final* if $Q = F$; *simple* if for any rewriting rule $q\,\binom{w_1}{w_2} \to q'$, either w_1 or w_2 is λ; *1-limited* if for any transition $q\,\binom{w_1}{w_2} \to q'$, we have $|w_1w_2| = 1$. By AWK, NWK, FWK, SWK, and 1WK, we denote the families of languages accepted by WK-automata which are arbitrary (A), stateless (N, no-state), all-final (F), simple (S), and 1-limited (1). When two restrictions are imposed at the same time, both of the corresponding symbols are used to identify the family.

As customary in automata theory, normal forms for WK-automata are available. For example, we can convert any WK-automaton into a 1-limited one without changing the language accepted, or the symmetric relation over which the original WK-automaton is defined [85]. Another normal form, standardizing the symmetric relation, is as follows:

Theorem 6.7 *([77]) For any WK-automaton M, we can construct a WK-automaton M_{id} over the identity relation id with $L(M) = L(M_{id})$.*

A 1-limited WK-automaton over the identity relation is equivalent to a *two-head finite automaton.* Therefore the next theorem follows, where TH denotes the family of languages accepted by two-head finite automata.

Theorem 6.8 *([94])* 1WK $=$ SWK $=$ 1SWK $=$ AWK $=$ TH.

In [23], Czeizler et al. have proposed three criteria of determinism. A WK-automaton M is said to be: *weakly deterministic* if at any point of computation by M, there is at most one possibility to continue the computation; and *deterministic* if for any pair of transition rules $q\,\binom{u}{v} \to q'$ and $q\,\binom{u'}{v'} \to q''$, we have $u \nsim_p u'$ or $v \nsim_p v'$. Clearly a deterministic WK-automaton is weakly-deterministic. Moreover, a deterministic WK-automaton over a symmetric relation ρ is said to be *strongly deterministic* if ρ is the identity. The families of languages accepted by *weakly deterministic*, *deterministic*, and *strongly deterministic* WK-automata are denoted by wdAWK, dAWK, and sdAWK, respectively. The symbol "A" can be replaced with N, F, S, 1, or their combination as in the non-deterministic case.

Proposition 6.1 *([23])* sdAWK $\subseteq$ dAWK $\subseteq$ wdAWK $\subseteq$ AWK.

The generative power of finite automata or Turing machines remains unchanged by bringing in determinism, while the determinism strictly weakens pushdown automata [56]. Thus, a natural question is whether the relation in Proposition 6.1 includes some strict inclusion or all of the families are the same.

Czeizler et al. [23] proved that dAWK = d1WK using a similar proof technique for AWK = 1WK. As mentioned above, the technique keeps the symmetric relation unchanged. As such, sdAWK = sd1WK follows immediately. Following the same reasoning to prove AWK = TH (Theorem 6.8), we can see that sdAWK is equivalent to the family dTH of languages accepted by *deterministic two-head finite automata*. It is known that there exists a language in TH $\setminus$ dTH, e.g., $L' = \{w \in \Sigma^* \mid w \neq w^R\}$. This means that non-deterministic WK-automata are strictly more powerful than strongly deterministic ones. In fact, the following stronger result holds.

Theorem 6.9 *([23])* sdAWK $\subsetneq$ dAWK.

Proof. It suffices to prove that $L' \in$ dAWK. We prove the statement only for the case when Σ is binary, but the statement holds for arbitrary finite alphabets.

Let $M = (\Sigma \cup \{c, d_a, d_b\}, \rho, Q, q_0, F, \delta)$ be a WK-automaton, where $\rho = \{(a,a),(a,d_a),(b,b),(b,d_b),(a,c),(b,c)\}$, $Q = \{q_0, q_f, q_a, q_b\}$, $F = \{q_f\}$, and δ consists of the following rules:

$$\begin{aligned}
q_0 \binom{\lambda}{x} &\to q_0, \qquad q_0 \binom{\lambda}{d_x} \to q_x, \text{with } x \in \{a,b\},\\
q_x \binom{y}{z} &\to q_x, \text{with } x,y,z \in \{a,b\},\\
q_x \binom{zy}{c} &\to q_f, \text{with } x,y,z \in \{a,b\},\ x \neq y,\\
q_f \binom{x}{\lambda} &\to q_f, \text{with } x \in \{a,b\}.
\end{aligned}$$

It is clear that M is deterministic. Let $w = w_1 w_2 \cdots w_n$ with $w_i \in \Sigma$. If $w \neq w^R$, then there exists a position k in the first half of w such that $w_k \neq w_{n-k+1}$. The characters d_a, d_b are used as a marker of this position. When M runs on the input $\binom{w_1 \cdots w_{k-1} w_k w_{k+1} \cdots w_{n-1} w_n}{w_1 \cdots w_{k-1} d_{w_k} w_{k+1} \cdots w_{n-1} c}$, it accepts w. On the other hand, M does not accept any palindrome regardless of what complement we choose; thus $L(M) = \{w \in \{a,b\}^+ \mid w \neq w^R\}$. □

This contrasts with the non-deterministic case where the complementarity relation does not play any active role (Theorem 6.7). Though it is natural now to ask if dAWK $\subseteq$ wdAWK is strict or not, this question remains open. Note that there exists a weakly deterministic WK-automaton which is not deterministic.

Proposition 6.1 and Theorem 6.9 conclude that strong determinism strictly weakens WK-automata. However deterministic WK-automata are still more powerful than finite automata. Since the construction of a strongly deterministic WK-automaton which simulates a given deterministic finite automaton is straightforward, in the following we include a stronger result.

Theorem 6.10 REG $\subseteq$ sdF1WK.

Proof. In order to simulate a deterministic finite automaton $A = (Q, \Sigma, q_0, F, \delta)$ with $Q = \{q_0, q_1, \dots, q_n\}$, we construct a strongly deterministic all-final 1-limited WK-automaton $M = (\Sigma, id, Q', q_{0,0}, Q', \delta')$, where $Q' = \{q_{i,j}, \overline{q_{i,j}} \mid 0 \leq i, j \leq n\}$, and for $a \in \Sigma$, $0 \leq i, j \leq n$,

$$\begin{aligned}
\delta'\left(q_{i,j}, \begin{pmatrix} a \\ \lambda \end{pmatrix}\right) &= \begin{cases} q_{k,j} & \text{if } \delta(q_i, a) = q_k \text{ and } q_k \notin F, \\ \overline{q_{k,j}} & \text{if } \delta(q_i, a) = q_k \text{ and } q_k \in F, \end{cases} \\
\delta'\left(\overline{q_{i,j}}, \begin{pmatrix} \lambda \\ a \end{pmatrix}\right) &= \begin{cases} \overline{q_{i,k}} & \text{if } \delta(q_j, a) = q_k \text{ and } q_k \notin F, \\ q_{i,k} & \text{if } \delta(q_j, a) = q_k \text{ and } q_k \in F, \end{cases}
\end{aligned}$$

The recognition of a sequence $\begin{bmatrix} w \\ w \end{bmatrix}$ consists of two identical simulations of recognition process of w on A over the upper and lower strands. Current states of the automaton A working on upper and lower strands are recorded as first and second subscripts of states in Q'. The overline of states indicates that M is now in the simulation over the lower strand. One switches these two phases every time the simulated automaton reaches some final state of A. We can see easily that $L(M) = L(A) \cup \{\lambda\}$. □

From Theorem 6.10, it is clear that WK-automata are more powerful than finite automata. Additional results on the generative capacity of WK-automata can be found in, e.g., [94]. Moreover, it was shown in [23, 92] that WK-automata recognize some regular languages in a less space-consuming manner. Usage of space is measured by the *state complexity* (for more details, see [92, 111]). It is well-known that the state complexity of some families of finite languages is unbounded when we consider the finite automata recognizing them. In other words, for any $k \geq 1$, there is a finite language which cannot be recognized by any finite automaton with at most $k - 1$ states. On the contrary, any finite language can be recognized by a WK-automaton with two states [23].

Determinism is one of the most essential properties for the design of an efficient parser. Due to the time-space trade-off, the stronger the determinism is, the more space-consuming WK-automata get. For example, it was shown in [23] that in order to recognize a finite language $L_k = \{a, aa, ..., a^{k-1}\}$, the strongly-deterministic WK-automaton needs at least k states, while for any k, two states are enough once the strong determinism is changed to determinism. Little is known about the state complexity with respect to the (strongly-, weakly-) deterministic WK-automata.

As a final result on this topic, we mention the following undecidability property about determinism of WK-automata.

Theorem 6.11 *([23]) It is undecidable whether a given WK-automaton is weakly deterministic.*

7 DNA Complementarity and Combinatorics on Words

Section 5 and Section 6 described new concepts and results in formal language theory and automata theory that attest to the influence that the notion of DNA-based information and its main aspect, the Watson-Crick complementarity, has had on formal language theory, coding theory and automata theory. This section is devoted to describing some of the ideas, notions and results that DNA-based information has brought to another area of theoretical computer science, namely combinatorics on words. Indeed, the "equivalence" from the informational point of view between a DNA single strand and its Watson-Crick complement led to several interesting generalizations of fundamental notions such as bordered word, palindrome, periodicity, conjugacy, and commutativity. Moreover, these generalizations led to natural extensions of two of the most fundamental results in combinatorics on words: the Fine and Wilf theorem and the Lyndon-Schützenberger equation. This section presents some of these concepts and results in combinatorics of words motivated by the Watson-Crick complementarity property of DNA-based information.

In the following, θ will denote an antimorphic involution. For further details on combinatorics on words, the reader is referred to [18].

As mentioned in Section 2, recognition sites of enzymes are often palindromic in a biological sense. Conventionally speaking, a palindrome is a word that reads the same way from the left and from the right, such as "racecar" in English. In contrast, in molecular biology terms, a palindromic DNA sequence is one which is equal to its WK-complementary sequence. For example, $TGGATCCA$ is palindromic in this sense. The biological palindromic motif is herein modelled as *θ-palindrome* (or, more generally, *pseudopalindrome*) defined as follows: A word w is a θ-palindrome if $w = \theta(w)$. θ-palindromes were investigated intensively from a theoretical computer science perspective, see e.g., in [29, 68].

The properties of θ-sticky-freeness and θ-overhang-freeness motivate the generalization of the notions of *border*, *commutativity* and *conjugacy* of words into *θ-border*, *θ-commutativity* and *θ-conjugacy* of words. A word $w \in \Sigma^+$ is *bordered* if there exists a word $v \in \Sigma^+$ satisfying $w = vx = yv$ for some $x, y \in \Sigma^+$. Kari and Mahalingam in [69] proposed an extended notion called the θ-borderedness of words as: w is *θ-bordered* if $w = vx = y\theta(v)$ for some $v, x, y \in \Sigma^+$. Note that θ-sticky-free languages do not contain any θ-bordered word. The *pseudoknot-bordered-word* proposed in [74] is a further extension of the notion of θ-bordered word. A word $w \in \Sigma^+$ is *pseudoknot-bordered* if $w = uvx = y\theta(u)\theta(v)$. A pseudoknot-bordered word models the crossing dependency occurring in DNA and RNA pseudoknots. A word $v \in \Sigma^+$ is said to be a *conjugate* of another word $u \in \Sigma^+$ if $v = yx$ and $u = xy$ for some $x, y \in \Sigma^*$, that is, $ux = xv$ holds. This notion was extended in [70] as follows: a word $u \in \Sigma^+$ is a *θ-conjugate* of a word $v \in \Sigma^+$ if $ux = \theta(x)v$ for some $x \in \Sigma^+$. In this case, either $v = \theta(u)$ or $u = \theta(x)z$, $v = zx$ for some $z \in \Sigma^+$; hence a language which contains both u and v cannot be strictly θ-sticky-free.

In contrast with the above two notions, the *θ-commutativity of words* introduced in [70] has a purely theoretical significance, being a mathematical tool for obtaining results involving WK-complementarity. Two words $u, v \in \Sigma^+$ are said to *commute* if $uv = vu$ holds. For two words u, v, we say that u θ-commutes with v if $uv = \theta(v)u$. This equation is a special case of conjugacy equations. Thus by applying a known result in combinatorics on words, one can deduce the following results.

Theorem 7.1 *([70]) For an antimorphic involution θ, and two words $u, v \in \Sigma^+$, if $uv = \theta(v)u$ holds, then $u = r(tr)^i$, $v = (tr)^j$ for some $i \geq 0$, $j \geq 0$, and θ-palindromes $r, t \in \Sigma^*$ such that rt is primitive.*

This theorem relates the three important notions: θ-commutativity of words, θ-palindrome, and primitivity.

Another interesting perspective is the informational equivalence between the two strands of a DNA double helix. Indeed, the two constituent single strands of a DNA double strand are "equivalent" with respect to the information they encode, and thus we can say that w and $\theta(w)$ are equivalent in this sense. Practical applications of this idea include an extended Lempel-Ziv algorithm for DNA molecules proposed in [50].

A word $w \in \Sigma^+$ is called *primitive* if it cannot be written as a power of another word; that is, $w = u^n$ implies $n = 1$ and $w = u$. For a word $w \in \Sigma^+$, the shortest $u \in \Sigma^+$ such that $w = u^n$ for some $n \geq 1$ is called the *primitive root* of the word w and is denoted by $\rho(w)$. It is well-known that for each word $w \in \Sigma^*$, there exists a unique primitive word $t \in \Sigma^+$ such that $\rho(w) = t$, i.e., $w = t^n$ for some $n \geq 1$, see e.g., [18]. In [25], the primitivity was extended to θ-primitivity for an antimorphic involution θ. A word $u \in \Sigma^+$ is said to be *θ-primitive* if there does not exist any word $t \in \Sigma^+$ such that $u \in \{t, \theta(t)\}^{\geq 2}$. For a word $w \in \Sigma^+$, we define the *θ-primitive root of w*, denoted by $\rho_\theta(w)$, as the shortest word $t \in \Sigma^+$ such that $w \in t\{t, \theta(t)\}^*$. Note that if w is θ-primitive, then $\rho_\theta(w) = w$. Note also that θ-primitivity can be defined also for θ being a morphic involution. However, as it is meant as a model of WK-complementarity, we will continue to assume that θ is antimorphic. For counterparts of the following results when θ is a morphic involution see [25].

We start by looking at some basic properties of θ-primitive words.

Proposition 7.1 *([25]) If a word $w \in \Sigma^+$ is θ-primitive, then it is also primitive. Moreover, the converse is not always true.*

Proof. By definition, it is clear that θ-primitive words are primitive. Remark that if there exists $a \in \Sigma$ satisfying $a \neq \theta(a)$, then the word $a\theta(a)$ is not θ-primitive, but is primitive.

□

If a word $w \in \Sigma^+$ is in $\{t, \theta(t)\}^+$ for some word $t \in \Sigma^+$ and t in turn is in $\{s, \theta(s)\}^+$ for some word $s \in \Sigma^+$, then we have that $w \in \{s, \theta(s)\}^+$. Thus, we have the following result.

Proposition 7.2 *([25]) The θ-primitive root of a word is θ-primitive, and hence primitive.*

It is a well-known fact that any conjugate of a primitive word is primitive. This fact is heavily employed in obtaining fundamental results including the Fine and Wilf theorem and solutions to the Lyndon-Schützenberger equation. In contrast, a conjugate of a θ-primitive word need not be θ-primitive. For instance, for the DNA involution τ defined in Section 5, the word $w = GCTA$ is τ-primitive, while its conjugate $w' = AGCT = AG\tau(AG)$ is not.

Proposition 7.3 *([25]) The class of θ-primitive words is not closed under circular permutations.*

An essential property of primitive words is that a primitive word cannot be equal to its conjugate. In other words, for a primitive word u, the equation $uu = xuy$ implies that either x or y is empty. Thus, u^i and u^j, with $i, j \geq 1$, cannot overlap non-trivially on a sequence longer than $|u|$. This is not the case when considering overlap between $\alpha(v, \theta(v)), \beta(v, \theta(v)) \in \{v, \theta(v)\}^+$ for some θ-primitive word $v \in \Sigma^+$. For instance, for the DNA involution τ and a τ-primitive word $v = CCGGAT$, $v^2 = CCGG \cdot \tau(v) \cdot AT$ holds. Nevertheless, an analogous result for θ-primitive words was obtained.

Theorem 7.2 *([72]) Let $v \in \Sigma^+$ be a θ-primitive word. Neither $v\theta(v)$ nor $\theta(v)v$ can be a proper infix of a word in $\{v, \theta(v)\}^3$.*

Furthermore, Czeizler et al. completely characterized all such nontrivial overlaps with the set of all solutions of the corresponding equation [24].

Theorem 7.3 *([24]) Let $v \in \Sigma^+$ be a θ-primitive word. The only possible proper overlaps of the form $\alpha(v, \theta(v)) \cdot x = y \cdot \beta(v, \theta(v))$ with $\alpha(v, \theta(v)), \beta(v, \theta(v)) \in \{v, \theta(v)\}^+$, $x, y \in \Sigma^+$ and $|x|, |y| < |v|$ are given in Table 1 (modulo a substitution of v by $\theta(v)$) together with the characterization of their sets of solutions.*

We now shift our attention to extensions of two essential results in combinatorics on words. The following theorem is known as the *Fine and Wilf theorem* [42], in its form for words, [18, 81]. It illustrates a fundamental periodicity property of words. Its concise proof is available, in, e.g., [18]. As usual, $\gcd(n, m)$ denotes the *greatest common divisor of n and m.*

Table 1: Characterization of possible proper overlaps of the form $\alpha(v, \theta(v)) \cdot x = y \cdot \beta(v, \theta(v))$. For the last three equations, $n \geq 0$, $m \geq 1$, $r, t \in \Sigma^+$ such that $r = \theta(r)$, $t = \theta(t)$, and rt is primitive. Note that the 4th and 5th equations are the same up to the antimorphic involution θ

Equation	Solution
$v^i x = y\theta(v)^i, i \geq 1$	$v = yp, x = \theta(y), p = \theta(p)$, and whenever $i \geq 2$, $y = \theta(y)$
$vx = yv$	$v = (pq)^{j+1}p$, $x = qp$, $y = pq$ for some $p, q \in \Sigma^+$, $j \geq 0$
$v\theta(v)x = yv\theta(v)$,	$v = (pq)^{j+1}p$, $x = \theta(pq)$, $y = pq$, with $j \geq 0$, $qp = \theta(qp)$
$v^{i+1}x = y\theta(v)^i v, i \geq 1$	$v = r(tr)^{n+m}r(tr)^n$, $x = (tr)^m r(tr)^n$, $y = r(tr)^{n+m}$
$v\theta(v)^i x = yv^{i+1}$, $i \geq 1$	$v = (rt)^n r(rt)^{m+n} r$, $y = (rt)^n r(rt)^m$, $x = (rt)^{m+n} r$
$v\theta(v)^i x = yv^i\theta(v)$, $i \geq 2$	$v = (rt)^n r(rt)^{m+n} r$, $y = (rt)^n r(rt)^m$, $x = (tr)^m r(tr)^n$

Theorem 7.4 *Let $u, v \in \Sigma^*$, $n = |u|$, and $m = |v|$. If a power of u and a power of v have a common prefix of length at least $n + m - gcd(n, m)$, then $\rho(u) = \rho(v)$. Moreover, the bound $n + m - gcd(n, m)$ is optimal.*

A natural question is whether one can obtain an extension of this result when, instead of taking powers of two words u and v, one looks at a word in $u\{u, \theta(u)\}^*$ and a word in $v\{v, \theta(v)\}^*$. The answer is yes. Note that without loss of generality one can suppose that the two words start with u and v because θ is an involution. Czeizler, Kari, and Seki provided extensions of Theorem 7.4 in two forms (Theorems 7.5 and 7.6) [25]. As illustrated in the following example, the bound given by Theorem 7.4 is not sufficient anymore.

Example 1 ([25]) Let $\theta : \{a, b\}^* \to \{a, b\}^*$ be the mirror image mapping defined as follows: $\theta(a) = a, \theta(b) = b$, and $\theta(w_1 \ldots w_n) = w_n \ldots w_1$, where $w_i \in \{a, b\}$ for all $1 \leq i \leq n$. Obviously, θ is an antimorphic involution on $\{a, b\}^*$. Let $u = (ab)^k b$ and $v = ab$. Then, u^2 and $v^k \theta(v)^{k+1}$ have a common prefix of length $2|u| - 1 > |u| + |v| - \gcd(|u|, |v|)$. Nevertheless u and v do not have the same θ-primitive root, that is $\rho_\theta(u) \neq \rho_\theta(v)$.

In the following, $\operatorname{lcm}(n, m)$ denotes the *least common multiple of n and m.*

Theorem 7.5 *([25]) Let $u, v \in \Sigma^+$, and $\alpha(u, \theta(u)) \in u\{u, \theta(u)\}^*$, $\beta(v, \theta(v)) \in v\{v, \theta(v)\}^*$ be two words sharing a common prefix of length at least $lcm(|u|, |v|)$. Then, there exists a word $t \in \Sigma^+$ such that $u, v \in t\{t, \theta(t)\}^*$, i.e., $\rho_\theta(u) = \rho_\theta(v)$. In particular, if $\alpha(u, \theta(u)) = \beta(v, \theta(v))$, then $\rho_\theta(u) = \rho_\theta(v)$.*

This theorem provides us with an alternative definition of the θ-primitive root of a word.

Corollary 7.1 *([25]) For any word $w \in \Sigma^+$ there exists a unique θ-primitive word $t \in \Sigma^+$ such that $w \in t\{t, \theta(t)\}^*$, i.e., $\rho_\theta(w) = t$.*

Corollary 7.2 *([25]) Let $u, v \in \Sigma^+$ be two words such that $\rho(u) = \rho(v) = t$. Then, $\rho_\theta(u) = \rho_\theta(v) = \rho_\theta(t)$.*

Next, we provide another bound for this extended Fine and Wilf theorem, which is in many cases much shorter than the bound given in Theorem 7.5. As noted before, due to Proposition 7.3, it is intuitive that we cannot use the concise proof technique based on the fact that a conjugate of a primitive word is primitive. The proof given in [25] involves rather technical case analyses.

Theorem 7.6 *([25]) Given two words $u, v \in \Sigma^+$ with $|u| > |v|$, if there exist two words $\alpha(u, \theta(u)) \in u\{u, \theta(u)\}^*$ and $\beta(v, \theta(v)) \in v\{v, \theta(v)\}^*$ having a common prefix of length at least $2|u| + |v| - gcd(|u|, |v|)$, then $\rho_\theta(u) = \rho_\theta(v)$.*

We conclude this section with an extension of another fundamental result related to the periodicity on words, namely the solution to the *Lyndon-Schützenberger equation.* The equation is of the form $u^\ell = v^n w^m$ for some $\ell, n, m \geq 2$ and $u, v, w \in \Sigma^+$. Lyndon and Schützenberger proved that this equation implies $\rho(u) = \rho(v) = \rho(w)$ [82]. A concise proof when u, v, w belong to a free semigroup can be found in, e.g., [52].

Incorporating the idea of θ-periodicity, the Lyndon-Schützenberger equation has been extended in [24] in the following manner. Let $u, v, w \in \Sigma^+$, θ be an antimorphic involution over Σ, and ℓ, n, m be integers ≥ 2. Let $\alpha(u, \theta(u)) \in \{u, \theta(u)\}^\ell$, $\beta(v, \theta(v)) \in \{v, \theta(v)\}^n$, and $\gamma(w, \theta(w)) \in \{w, \theta(w)\}^m$. The *extended Lyndon-Schützenberger equation* is

$$\alpha(u, \theta(u)) = \beta(v, \theta(v)) \quad \gamma(w, \theta(w)).$$

In [24], the authors investigated the problem of finding conditions on ℓ, n, m such that if the equation holds, then $u, v, w \in \{t, \theta(t)\}^+$ for some $t \in \Sigma^+$. If such t exists, we say that (ℓ, n, m) *imposes θ-periodicity on* u, v, w. The original condition $\ell, n, m \geq 2$ is not enough for this extension as shown below.

Example 2 ([24]) Let $\Sigma = \{a, b\}$ and θ be the mirror image. Take now $u = a^k b^2 a^{2k}$, $v = \theta(u)^l a^{2k} b^2 = (a^{2k} b^2 a^k)^l a^{2k} b^2$, and $w = a^2$, for some $k, l \geq 1$. Then, although $\theta(u)^{l+1} u^{l+1} = v^2 w^k$, there is no word $t \in \Sigma^+$ with $u, v, w \in \{t, \theta(t)\}^+$.

Example 3 ([24]) Consider again $\Sigma = \{a, b\}$ and the mirror image θ, and take $u = b^2(aba)^k$, $v = u^l b = (b^2(aba)^k)^l b$, and $w = aba$ for some $k, l \geq 1$. Then, although $u^{2l+1} = v\theta(v)w^k$, there is no word $t \in \Sigma^+$ with $u, v, w \in \{t, \theta(t)\}^+$.

Example 4 Let $\Sigma = \{a, b\}$ and θ be the mirror image. Let $v = a^{2m} b^{2j}$ and $w = aa$ for some $m \geq 1$ and $j \geq 1$. Then $v^n w^m = (a^{2m} b^{2j})^n a^{2m}$. This is θ-palindrome of even length and hence it can be written as $u\theta(u)$ for some $u \in \Sigma^+$. Clearly $\rho_\theta(w) = a$ and $\rho_\theta(v) \neq a$ because v contains b. Hence $(2, n, m)$ is not enough to impose the θ-periodicity.

Thus, once either n or m is 2, it is not always the case that there exists a word $t \in \Sigma^+$ such that $u, v, w \in \{t, \theta(t)\}^+$. On the other hand, it was proved that (ℓ, n, m) imposes θ-periodicity on u, v, w if $\ell \geq 5$, $n, m \geq 3$.

Theorem 7.7 *([24]) For words $u, v, w \in \Sigma^+$ and $\ell, n, m \geq 2$, let $\alpha(u, \theta(u)) \in \{u, \theta(u)\}^\ell$, $\beta(v, \theta(v)) \in \{v, \theta(v)\}^n$, and $\gamma(w, \theta(w)) \in \{w, \theta(w)\}^m$. If $\alpha(u, \theta(u)) = \beta(v, \theta(v))\gamma(w, \theta(w))$ holds and $\ell \geq 5$, $n, m \geq 3$, then $u, v, w \in \{t, \theta(t)\}^+$ for some $t \in \Sigma^+$.*

This theorem requires rather complex case analyses, too, and hence the proof is omitted here. However, if ℓ, n, m are "big" enough, we can easily see that there is much room to employ the extended Fine and Wilf theorem.

We conclude this section with Table 2 which summarizes the results obtained so far on the extended Lyndon-Schützenberger equation.

Table 2: Summary of the extended Lyndon-Schützenberger equation results.

l	n	m	θ-periodicity	proved by
≥ 5	≥ 3	≥ 3	YES	Theorem 7.7
4	≥ 3	≥ 3	?	
3	≥ 3	≥ 3	?	
≥ 3	2	≥ 2	NO	Examples 2 and 3
≥ 3	≥ 2	2	NO	
2	≥ 2	≥ 2	NO	Examples 4

8 Conclusions

In this review we described how information can be encoded on DNA strands and how bio-operations can be used to perform computational tasks. In addition, we presented examples of the influence that fundamental properties of DNA-encoded information, especially the Watson-Crick complementarity, have had on various areas of theoretical computer science such as formal language theory, coding theory, automata theory, and combinatorics on words.

A few final remarks are in order regarding DNA-encoded data and the bio-operations that are used to act on it. The descriptions of DNA structure and DNA bio-operations point to the fact that DNA-encoded information is very different from electronically-encoded information, and bio-operations also differ from electronic computer operations. A fundamental difference arises, for example, from the fact that in electronic computing data interaction is fully controlled while, in a test-tube DNA-computer, free-floating data-encoding DNA single strands can interact because of Watson-Crick complementarity. Another difference is that, in DNA computing, a bio-operation usually consumes both operands. This implies that, if one of the operands is either involved in an illegal binding or has been consumed by a previous bio-operation, it is unavailable for the desired computation. Yet another difference is that, while in electronic computing a bit is a single individual element, in DNA experiments each submicroscopic DNA molecule is usually present in millions of identical copies. The bio-operations operate in a massively parallel fashion on all identical strands and this process is governed by the laws of chemistry and thermodynamics, with the output obeying statistical laws.

Differences like the ones mentioned above point to the fact that a fresh approach is needed when employing as well as when theoretically investigating bioinformation and biocomputation, and offer a wealth of problems to explore at this rich intersection between molecular biology and computer science.

Examples of such fascinating research areas and topics include models and wet implementations of molecular computing machineries, membrane computing, DNA computing by splicing and insertion-deletion, bacterial computing and communication, DNA memory, DNA computing by self-assembly, and computational aspects of gene asembly in ciliates, as described in, e.g., Chapters **??** of this handbook.

References

[1] L. Adleman. Molecular computation of solutions to combinatorial problems. *Science*, 26f6(5187):1021–1024, November 1994.

[2] L. Adleman. Computing with DNA. *Scientific American*, 279:54–61, 1998.

[3] A. Alhazov and M. Cavaliere. Computing by observing bio-systems: The case of sticker systems. In Feretti et al. [41], pages 1–13.

[4] M. Amos. *Theoretical and Experimental DNA Computation.* Springer-Verlag, Berlin, 2005.

[5] M. Andronescu, D. Dees, L. Slaybaugh, Y. Zhao, A. Condon, B. Cohen, and S. Skiena. Algorithms for testing that sets of DNA words concatenate without secondary structure. In Hagiya and Ohuchi [51], pages 182–195.

[6] M. Arita. Writing information into DNA. In N. Jonoska, G. Păun, and G. Rozenberg, editors, *Aspects of Molecular Computing*, volume 2950 of *Lecture Notes in Computer Science*, pages 23–35. Springer-Verlag, Berlin, 2004.

[7] M. Arita and S. Kobayashi. DNA sequence design using templates. *New Generation Computing*, 20:263–277, 2002.

[8] C. Bancroft, T. Bowler, B. Bloom, and C. Clelland. Long-term storage of information in DNA. *Science*, 293:1763–1765, 2001.

[9] E. Baum. DNA sequences useful for computation. In L. Landweber and E. Baum, editors, *DNA Based Computers II*, volume 44 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 235–246. American Mathematical Society, 1998.

[10] H. Bordihn, M. Holzer, and M. Kutrib. Hairpin finite automata. In T. Harju, J. Karhumäki, and A. Lepistö, editors, *Developments in Language Theory*, volume 4588 of *Lecture Notes in Computer Science*, pages 108–119, Berlin, 2007. Springer-Verlag.

[11] R. Braich, N. Chelyapov, C. Johnson, P. Rothemund, and L. Adleman. Solution of a 20-variable 3-SAT problem on a DNA computer. *Science*, 296:499–502, 2002.

[12] C. Calladine and H. Drew. *Understanding DNA: the molecule and how it works (2nd edition).* Academic Press, 1997.

[13] A. Carbone and N. Pierce, editors. *Proc. DNA Computing 11*, volume 3892 of *Lecture Notes in Computer Science*, Berlin, 2006. Springer-Verlag.

[14] M. Cavaliere. Computing by observing: A brief survey. In *Logic and Theory of Algorithms*, volume 5028 of *Lecture Notes in Computer Science*, pages 110–119. Springer, Berlin Heidelberg, 2008.

[15] M. Cavaliere and P. Leupold. Evolution and observation - a new way to look at membrane systems. In C. Martin-Vide, G. Mauri, G. Păun, G. Rozenberg, and A. Salomaa, editors, *Membrane Computing*, volume 2933 of *Lecture Notes in Computer Science*, pages 70–87. Springer, 2004.

[16] J. Chen, R. Deaton, M. Garzon, J. Kim, D. Wood, H. Bi, D. Carpenter, and Y.-Z. Wang. Characterization of non-crosshybridizing DNA oligonucleotides manufactured in vitro. *Natural Computing*, 5(2):165–181, 2006.

[17] J. Chen and J. Reif, editors. *Proc. DNA Computing 9*, volume 2943 of *Lecture Notes in Computer Science*, Berlin, 2004. Springer.

[18] C. Choffrut and J. Karhumäki. Combinatorics of words. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 1, pages 329–438. Springer-Verlag, Berlin-Heidelberg-New York, 1997.

[19] J. Cox. Long-term data storage in DNA. *Trends in Biotechnology*, 19:247–250, 2001.

[20] E. Czeizler and E. Czeizler. On the power of parallel communicating Watson-Crick automata systems. *Theoretical Computer Science*, 358:142–147, 2006.

[21] E. Czeizler and E. Czeizler. Parallel communicating Watson-Crick automata systems. *Acta Cybernetica*, 17:685–700, 2006.

[22] E. Czeizler and E. Czeizler. A short survey on Watson-Crick automata. *Bulletin of the EATCS*, 89:104–119, 2006.

[23] E. Czeizler, E. Czeizler, L. Kari, and K. Salomaa. Watson-Crick automata: Determinism and state complexity. In *Proc. Descriptional Complexity of Formal Systems DCFS'08*, pages 121–133, 2008.

[24] E. Czeizler, E. Czeizler, L. Kari, and S. Seki. An extension of the Lyndon Schützenberger result to pseudoperiodic words. In V. Diekert and D. Nowotka, editors, *Proc. Developments in Language Theory DLT'09*, volume 5583 of *Lecture Notes in Computer Science*, Berlin, 2009. Springer-Verlag.

[25] E. Czeizler, L. Kari, and S. Seki. On a special class of primitive words. In *Proc. of Mathematical Foundations of Theoretical Computer Science MFCS 2008*, volume 5162 of *Lecture Notes in Computer Science*, pages 265–277, Berlin-Heidelberg, 2008. Springer.

[26] M. Daley, O. Ibarra, and L. Kari. Closure and decidability properties of some language classes with respect to ciliate bio-operations. *Theoretical Computer Science*, 306(1):19–38, 2003.

[27] M. Daley, L. Kari, and I. McQuillan. Families of languages defined by ciliate bio-operations. *Theoretical Computer Science*, 320:51–69, 2004.

[28] M. Daley and L.Kari. DNA computing: Models and implementations. *Comments on Theoretical Biology*, 7:177–198, 2002.

[29] A. de Luca and A. D. Luca. Pseudopalindrome closure operators in free monoids. *Theoretical Computer Science*, 362:282–300, 2006.

[30] R. Deaton, J. Chen, H. Bi, and J. Rose. A software tool for generating non-crosshybridizing libraries of DNA oligonucleotides. In Hagiya and Ohuchi [51], pages 252–61.

[31] R. Deaton, J. Chen, J. Kim, M. Garzon, and D. Wood. Test tube selection of large independent sets of DNA oligonucleotides. In J. Chen, N. Jonoska, and G. Rozenberg, editors, *Nanotechnology: Science and Computation*, pages 147–161. Springer-Verlag, Berlin, 2006.

[32] S. Diaz, J. Esteban, and M. Ogihara. A DNA-based random walk method for solving k-SAT. In A. Condon and G. Rozenberg, editors, *Proc. DNA Computing 6*, volume 2054 of *Lecture Notes in Computer Science*, pages 209–219, Berlin, 2001. Springer-Verlag.

[33] R. Dirks and N. Pierce. An algorithm for computing nucleic acid base-pairing probabilities including pseudoknots. *Journal of Computational Chemistry*, 25:1295–1304, 2004.

[34] M. Domaratzki. Hairpin structures defined by DNA trajectories. *Theory of Computing Systems*, online 2007. DOI 10.1007/s00224-007-9086-6.

[35] K. Drlica. *Understanding DNA and gene cloning: a guide for the curious.* Wiley and Sons, 1996.

[36] A. Dyachkov, A. Macula, W. Pogozelski, T. Renz, V. Rykov, and D. Torney. New t-gap insertion-deletion-like metrics for DNA hybridization thermodynamic modeling. *Journal of Computational Biology*, 13(4):866–881, 2006.

[37] A. Dyachkov, A. Macula, V. Rykov, and V. Ufimtsev. DNA codes based on stem similarities between DNA sequences. In Garzon and Yan [48], pages 146–151.

[38] A. Ehrenfeucht, T. Harju, I. Petre, D. Prescott, and G. Rozenberg. *Computation in Living Cells: Gene Assembly in Ciliates*. Natural Computing Series. Springer-Verlag, Berlin, 2004.

[39] D. Faulhammer, A. Cukras, R. Lipton, and L. Landweber. Molecular computation: RNA solutions to chess problems. *Proc. of the National Academy of Sciences of the USA*, 97:1385–1389, 2000.

[40] U. Feldkamp, S. Saghafi, W. Banzhaf, and H. Rauhe. DNASequenceGenerator – a program for the construction of DNA sequences. In Jonoska and Seeman [61], pages 23–32.

[41] C. Feretti, G. Mauri, and C. Zandron, editors. *Proc. DNA Computing 10*, volume 3384 of *Lecture Notes in Computer Science*, Berlin, 2005. Springer-Verlag.

[42] N. Fine and H. Wilf. Uniqueness theorem for periodic functions. *Proceedings of the American Mathematical Society*, 16(1):109–114, February 1965.

[43] R. Freund, G. Păun, G. Rozenberg, and A. Salomaa. Bidirectional sticker systems. In R. Altman, A. Dunker, L. Hunter, and T. Klein, editors, *Pacific Symposium on Biocomputing 98*, pages 535–546, Singapore, 1998. World Scientific.

[44] R. Freund, G. Păun, G. Rozenberg, and A. Salomaa. Watson-Crick finite automata. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 48:297–327, 1999.

[45] A. Frutos, Q. Liu, A. Thiel, A. Sanner, A. Condon, L. Smith, and R. Corn. Demonstration of a word design strategy for DNA computing on surfaces. *Nucleic Acids Research*, 25(23):4748–57, 1997.

[46] M. Garzon, P. Neathery, R. Deaton, R. Murphy, D. Franceschetti, and S. Stevens Jr. A new metric for DNA computing. In J. Koza, K. Deb, M. Dorigo, D. Vogel, M. Garzon, H. Iba, and R. Riolo, editors, *Proc. Genetic Programming 1997*, pages 479–490, San Francisco, CA, 1997. Morgan Kaufmann.

[47] M. Garzon, V. Phan, S. Roy, and A. Neel. In search of optimal codes for DNA computing. In C. Mao and T. Yokomori, editors, *Proc. DNA Computing 12*, volume 4287 of *Lecture Notes in Computer Science*, pages 143–156, Berlin, 2006. Springer-Verlag.

[48] M. Garzon and H. Yan, editors. *Proc. DNA Computing 13*, volume 4848 of *Lecture Notes in Computer Science*, Berlin, 2008. Springer-Verlag.

[49] L. Gonick and M. Wheelis. *The Cartoon Guide to Genetics.* Collins, updated edition, 1991.

[50] S. Grumbach and F. Tahi. Compression of DNA sequences. In *Proc. IEEE Symposium on Data Compression*, pages 340–350, 1993.

[51] M. Hagiya and A. Ohuchi, editors. *Proc. DNA Computing 8*, volume 2568 of *Lecture Notes in Computer Science*, Berlin, 2003. Springer-Verlag.

[52] T. Harju and D. Nowotka. The equation $x^i = y^j z^k$ in a free semigroup. *Semigroup Forum*, 68:488–490, 2004.

[53] J. Hartmanis. On the weight of computations. *Bulletin of the EATCS*, 55:136–138, 1995.

[54] T. Head. Relativised code concepts and multi-tube DNA dictionaries. In C. Calude and G. Păun, editors, *Finite Versus Infinite: Contributions to an Eternal Dilemma*, pages 175–186. Springer-Verlag, London, 2000.

[55] H. Hoogeboom and N. van Vugt. Fair sticker languages. *Acta Informatica*, 37:213–225, 2000.

[56] J. Hopcroft and J. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979.

[57] S. Hussini, L. Kari, and S. Konstantinidis. Coding properties of DNA languages. *Theoretical Computer Science*, 290(3):1557–1579, 2003.

[58] Z. Ignatova, I. Martínez-Pérez, and K.-H. Zimmermann. *DNA Computing Models*. Springer-Verlag, Berlin, 2008.

[59] N. Jonoska, D. Kephart, and K. Mahalingam. Generating DNA code words. *Congressus Numerantium*, 156:99–110, 2002.

[60] N. Jonoska and K. Mahalingam. Languages of DNA based code words. In Chen and Reif [17], pages 61–73.

[61] N. Jonoska and N. Seeman, editors. *Proc. DNA Computing 7*, volume 2340 of *Lecture Notes in Computer Science*, Berlin, 2002. Springer-Verlag.

[62] L. Kari. DNA computing: The arrival of biological mathematics. *The Mathematical Intelligencer*, 19(2):9–22, 1997.

[63] L. Kari, R. Kitto, and G. Thierrin. Codes, involutions and DNA encoding. In W. Brauer, H. Ehrig, J. Karhumäki, and A. Salomaa, editors, *Formal and Natural Computing*, volume 2300 of *Lecture Notes in Computer Science*, pages 376–393. Springer-Verlag, Berlin, 2002.

[64] L. Kari, S. Konstantinidis, and P. S. k. Bond-free languages: formalizations, maximality and construction methods. *International Journal of Foundations of Computer Science*, 16(5):1039–1070, 2005.

[65] L. Kari, S. Konstantinidis, and P. S. k. On properties of bond-free DNA languages. *Theoretical Computer Science*, 334(1–3):131–159, 2005.

[66] L. Kari, S. Konstantinidis, E. Losseva, P. S. k, and G. Thierrin. A formal language analysis of DNA hairpin structures. *Fundamenta Informaticae*, 71(4):453–475, 2006.

[67] L. Kari, S. Konstantinidis, E. Losseva, and G. Wozniak. Sticky-free and overhang-free DNA languages. *Acta Informatica*, 40:119–157, 2003.

[68] L. Kari and K. Mahalingam. Watson-Crick palindromes in DNA computing. Natural Computing 2009, to appear.

[69] L. Kari and K. Mahalingam. Involutively bordered words. *International Journal of Foundations of Computer Science*, 18(5):1089–1106, 2007.

[70] L. Kari and K. Mahalingam. Watson-Crick conjugate and commutative words. In Garzon and Yan [48], pages 273–283.

[71] L. Kari, K. Mahalingam, and G. Thierrin. The syntactic monoid of hairpin-free languages. *Acta Informatica*, 44(3–4):153–166, 2007.

[72] L. Kari, B. Masson, and S. Seki. Towards a complete characterization of an extended Lyndon Schützenberger equation. In preparation, 2009.

[73] L. Kari, G. Păun, G. Rozenberg, A. Salomaa, and S. Yu. DNA computing, sticker systems, and universality. *Acta Informatica*, 35:401–420, 1998.

[74] L. Kari and S. Seki. On pseudoknot-bordered words and their properties. *Journal of Computer and System Sciences*, 75(2):113–121, 2009.

[75] A. Kijima and S. Kobayashi. Efficient algorithm for testing structure freeness of finite set of biomolecular sequences. In Carbone and Pierce [13], pages 171–180.

[76] S. Kobayashi. Testing structure-freeness of regular sets of biomolecular sequences. In Feretti et al. [41], pages 192–201.

[77] D. Kuske and P. Weigel. The role of the complementarity relation in Watson-Crick automata and sticker systems. In C. Calude, E. Claude, and M. Dinneen, editors, *DLT 2004*, volume 3340 of *Lecture Notes in Computer Science*, pages 272–283, Berlin Heidelberg, 2004. Springer-Verlag.

[78] B. Lewin. *Genes IX*. Johns and Bartlett Publishers, 2007.

[79] R. Lipton. Using DNA to solve NP-complete problems. *Science*, 268:542–545, 1995.

[80] W. Liu, L. Gao, Q. Zhang, G. Xu, X. Zhu, X. Liu, and J. Xu. A random walk DNA algorithm for the 3-SAT problem. *Current Nanoscience*, 1:85–90, 2005.

[81] M. Lothaire. *Combinatorics on Words*, volume 17 of *Encyclopedia of Mathematics and its Applications*. Addison-Wesley, 1983.

[82] R. Lyndon and M. Schützenberger. The equation $a^m = b^n c^p$ in a free group. *Michigan Mathematical Journal*, 9:289–298, 1962.

[83] F. Manea, V. Mitrana, and T. Yokomori. Two complementary operations inspired by the DNA hairpin formation: Completion and reduction. *Theoretical Computer Science*, 410(4–5):417–425, 2009.

[84] A. Marathe, A. Condon, and R. Corn. On combinatorial DNA word design. *Journal of Computational Biology*, 8(3):201–220, 2001.

[85] C. Martin-Víde, G. Păun, G. Rozenberg, and A. Salomaa. Universality results for finite H systems and for Watson-Crick automata. In G. Păun, editor, *Computing with Bio-Molecules. Theory and Experiments*, pages 200–220. Springer, Berlin, 1998.

[86] A. Mateescu, G. Rozenberg, and A. Salomaa. Shuffle on trajectories: Syntactic constraints. *Theoretical Computer Science*, 197:1–56, 1998.

[87] G. Mauri and C. Ferretti. Word design for molecular computing: A survey. In Chen and Reif [17], pages 37–46.

[88] N. Morimoto, M. Arita, and A. Suyama. Stepwise generation of Hamiltonian Path with molecules. In D. Lundh, B. Olsson, and A. Narayanan, editors, *Proc. Biocomputing and Emergent Computation*, pages 184–192. World Scientific, 1997.

[89] B. Nagy. On 5' → 3' sensing Watson-Crick finite automata. In Garzon and Yan [48], pages 256–262.

[90] E. Petre. Watson-Crick ω-automata. *Journal of Automata, Languages and Combinatorics*, 8:59–70, 2003.

[91] V. Phan and M. Garzon. On codeword design in metric DNA spaces. *Natural Computing*, online 2008. DOI 10.1007/s11047-008-9088-6.

[92] A. Păun and M. Păun. State and transition complexity of Watson-Crick finite automata. In G. Ciobanu and G. Păun, editors, *FCT'99*, volume 1684 of *Lecture Notes in Computer Science*, pages 409–420, Berlin Heidelberg, 1999. Springer-Verlag.

[93] G. Păun and G. Rozenberg. Sticker systems. *Theoretical Computer Science*, 204:183–203, 1998.

[94] G. Păun, G. Rozenberg, and A. Salomaa. *DNA Computing: New Computing Paradigms.* Springer, 1998.

[95] G. Păun, G. Rozenberg, and T. Yokomori. Hairpin languages. *International Journal of Foundations of Computer Science*, 12(6):837–847, 2001.

[96] J. Reif, T. LaBean, M. Pirrung, V. Rana, B. Guo, C. Kingsford, and G. Wickham. Experimental construction of very large scale DNA databases with associative search capability. In Jonoska and Seeman [61], pages 231–247.

[97] J. Rose, R. Deaton, M. Hagiya, and A. Suyama. PNA-mediated whiplash PCR. In Jonoska and Seeman [61], pages 104–116.

[98] J. Sager and D. Stefanovic. Designing nucleotide sequences for computation: a survey of constraints. In Carbone and Pierce [13], pages 275–289.

[99] Y. Sakakibara and S. Kobayashi. Sticker systems with complex structures. *Soft Computing*, 5:114–120, 2001.

[100] A. Salomaa. *Formal Languages.* Academic Press, 1973.

[101] J. SantaLucia. A unified view of polymer, dumbbell and oligonucleotide DNA nearest-neighbor thermodynamics. *Proc. of the National Academy of Sciences of the USA*, 95(4):1460–1465, 1998.

[102] J. Sempere. On local testability in Watson-Crick finite automata. In *Proc. the International Workshop, Automata for Cellular and Molecular Computing*, pages 120–128, 2007.

[103] J. Sempere. Exploring regular reversibility in Watson-Crick finite automata. In *Proc. 13th International Symposium on Artificial Life and Robotics (AROB2008)*, pages 505–509, 2008.

[104] G. Smith, C. Fiddes, J. Hawkins, and J. Cox. Some possible codes for encrypting data in DNA. *Biotechnology Letters*, 25:1125–1130, 2003.

[105] W. Stemmer, A. Crameri, K. Ha, T. Brennan, and H. Heyneker. Single-step assembly of a gene and entire plasmid from large numbers of oligodeoxyribonucleotides. *GENE*, 164:49–53, 1995.

[106] D. Tulpan, H. Hoos, and A. Condon. Stochastic local search algorithms for DNA word design. In Hagiya and Ohuchi [51], pages 229–241.

[107] P. Turner, A. McLennan, A. Bates, and M. White. *Instant Notes in Molecular Biology.* Garland Publishing Inc., 2nd edition, 2000.

[108] X. Wang, Z. Bao, J. Hu, S. Wang, and A. Zhan. Solving the SAT problem using a DNA computing algorithm based on ligase chain reaction. *Biosystems*, 91:117–125, 2008.

[109] D. Wood. *Theory of Computation.* Harper & Row, New York, 1987.

[110] H. Yoshida and A. Suyama. Solution to 3-SAT by breadth-first search. In E. Winfree and D. Gifford, editors, *DNA Based Computers V*, volume 54 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 9–22. American Mathematical Society, 2000.

[111] S. Yu. State complexity of finite and infinite regular languages. *Bulletin of the EATCS*, 76:142–152, 2002.

Index