 Transactions on Engineering Sciences vol 15, © 1997 WIT Press, www.witpress.com, ISSN 1743-3533

Computability paradigms based on DNA complementarity

Arto Salomaa

Turku Centre for Computer Science and Mathematics
Department, University of Turku, 20014 Turku, Finland

Abstract

The paper discusses the fundamental role of the Watson-Crick complementarity in DNA computing. The close relation of the complementarity to the twin-shuffle language guarantees the universal computing power for any model of DNA computing possessing sufficient input and output facilities. Some such models are also briefly discussed.

1 Introduction

Adleman demonstrated, how standard methods of molecular biology could be used to solve a difficult computational problem. Since then the interest on "DNA based computers" has been growing rapidly. There are still considerable obstructions to creating a practical molecular computer, and also very pessimistic views have been expressed.

Although the real practical feasibility of molecular computers remains still in doubt, the field has opened new vistas and important research problems both for computer scientists and biologists. The computer scientist and mathematician is looking for new models of computation. I would like to call such models "Watson-Crick machines" because of the Watson-Crick complementarity in DNA: the four nucleotides A (adenine), T (thymine), C (cytosine) and G (guanine) form two complementary pairs (A,T) and (C,G). In a Watson-Crick machine, Turing's slow diligent clerk is replaced by DNA strands acting in a test tube. In this way the massive parallelism of DNA strands may render tractable some computational problems that

were beyond the reach of the diligent clerk; not because the DNA strands are smarter but simply because they can make many tries at once. For the biologist, the unexpected results in DNA computing indicate that models of DNA computers could be significant also for the study of important biological problems such as evolution. Moreover, the techniques of DNA manipulation developed originally for computational purposes could find relevant applications in genetic engineering. However, because of the rather diverse research traditions, it will not be easy to establish common idioms, let alone common vocabulary, for the researchers in computer science and biology.

In the issue of Science containing Adleman's seminal work, David Gifford wrote, [6]: "If we are able to construct a universal machine out of biological macromolecular components, then we could perform any computation by means of biological techniques. There are certainly powerful practical motivations for this approach, including the information-encoding density offered by macromolecules and the high energy efficiency of enzyme systems. At present, there is no known way of creating a synthetic universal system based on macromolecules. Universal systems require the ability to store and retrieve information, and DNA is certainly up to the task if one could design appropriate molecular mechanisms to interpret and update the information in DNA. This ultimate goal remains elusive, but once solved, it will revolutionize the way we think about both computer science and molecular biology."

Although such an ultimate goal still remains elusive, there are by now already many DNA based universal computational models. (For instance, see [12].) That there are many diverse ways of constructing DNA based universal computers is due to the following overall observation. Watson-Crick complementarity guarantees universal computations in any model of DNA computers having sufficient capabilities of handling inputs and outputs. This is a consequence of the close similarity between the Watson-Crick complementarity and the *twin-shuffle language*, [15]. The latter is a language over four letters, known to be powerful enough to serve as a basis for arbitrary computations. This state of affairs can be viewed also as a mathematical explanation to the number of nucleotides in DNA being four, rather than three or five.

It seems obvious that theoretical studies about DNA computing must make use of the following two advantages stemming from DNA molecules. (i) Watson-Crick complementarity which renders the power of the twin-shuffle language available, and (ii) the multitude of DNA molecules which brings massive parallelism to the computing scene. It seems that theoretical studies have so far concentrated on (i). It is still quite an open area to model (ii) mathematically, as well as to combine (i) and (ii) into one model of DNA computing.

The purpose of this paper is to discuss the significance of the Watson-

Transactions on Engineering Sciences vol 15, © 1997 WIT Press, www.witpress.com, ISSN 1743-3533

Crick complementarity in an arbitrary theoretical setup of DNA computing. The universality of the twin-shuffle language goes back to [4]. The interconnection between the twin-shuffle language and Watson-Crick complementarity was made explicit in [15]. Sections 2 and 3 are devoted to a general discussion. Two specific setups are then presented in Sections 4 and 5. A grammatical setup, *matching systems*, [8], forms the subject matter of Section 4. Finally, Section 5 discusses an automata-theoretic setup, [5].

The paper assumes only little background on the part of the reader. The necessary prerequisites about formal languages can be found, for instance, in [17] or [18].

As regards DNA computing in general, the literature is already quite extensive and growing rapidly. A site containing a good bibliography is

http://www.wi.LeidenUniv.nl/~jdassen/dna.bib

Research concerning *splicing systems* (for instance, see [12] and [10]), as well as *insertion and deletion systems*, [13], is closely connected with matters discussed in this paper. A general exposition, [11], is forthcoming. Important overall aspects can be found in [7], [16].

2 Watson-Crick complementarity

DNA (deoxyribonucleic acid) is found in every living organism as the storage medium for genetic information. It consists of polymer chains, usually referred to as *DNA strands*. A chain is composed of *nucleotides*, also referred to as *bases*. Sometimes the chains are also referred to as *oligonucleotides*, briefly *oligos*. The four DNA nucleotides or bases are customarily denoted by A, C, G and T.

According to a chemical convention, each strand has a $5'$ end and a $3'$ end, for instance,

$$5'CATTAG3' \quad or \quad 3'GTAATC5'.$$

Thus, the strands are oriented. The familiar double helix of DNA arises by the bondage of two separate strands. Actually, single strands are fragile and, consequently, information is stored in double strands.

The phenomenon known as *Watson-Crick complementarity* comes into the picture in the formation of double strands. Bonding happens by the pairwise attraction of bases: A bonds with T, and C bonds with G. Therefore, the two unordered pairs (A,T) and (C,G) are known as *complementary* pairs of bases. Bonding will only occur if the bases in the two strands are pairwise complementary and, moreover, the strands have opposite orientations: one of them extends from $5'$ to $3'$, and the other from $3'$ to $5'$. This is the case with the two strands mentioned above and, consequently,

they form the double strand

$$5'CATTAG3'$$
$$3'GTAATC5'$$

Double strands can again be dissolved into single strands by heating the solution. This process is usually referred to as *melting*. The reverse process, referred to as *annealing*, is performed by cooling the solution. We do not try to give here any survey about the operations possible or feasible with DNA strands, or about the error-freeness of such operations. Studies of DNA computing have brought forward many significant new problem areas along these lines. However, for our purposes, the following observation is important. Watson-Crick complementarity gives us something important "for free": whenever bondage happens, we know that the bases at corresponding places must be complementary. If A appears in one of the strands, we know that T appears in the corresponding place in the other strand. If we know one member of a bond, we know also the other member "without looking". Otherwise, bondage would not have taken place. This observation is very basic for DNA computing. As we will see, it means that we have the twin-shuffle language freely available. This leads to universality of models of DNA computing satisfying certain requirements of handling inputs and outputs.

The observation mentioned is relevant also for simple tasks of DNA computing. It is also behind Adleman's fundamental experiment, [1], of solving an instance of the Hamiltonian path problem, HPP.

Given a directed graph, we want to solve its HPP. Each vertex is encoded by an oligonucleotide. (Adleman used strands of length 20.) Edges are encoded by oligonucleotides of the same length such that the first (resp. second) half of the oligonucleotide encoding an edge equals the (Watson-Crick) complement of the second (resp. first) half of the oligonucleotide encoding the outgoing (resp. incoming) vertex of the edge.

In symbols, let x and y, $|x| = |y| = 20$, be the oligonucleotides encoding two vertices such that there is an edge from x to y. Write

$$x = x_1x_2,\ y = y_1y_2,\ |x_1| = |x_2| = |y_1| = |y_2| = 10.$$

(We ignore the orientation markers $5'$ and $3'$ in these length considerations. The orientation markers are irrelevant for our purposes also in the sequel.) Consider the alphabet $\Sigma_{DNA} = \{A, C, G, T\}$ and define the letter-to-letter morphism $h_w : \Sigma^*_{DNA} \rightarrow \Sigma^*_{DNA}$ by

$$h_w(A) = T,\ h_w(T) = A,\ h_w(C) = G,\ h_w(G) = C.$$

The morphism h_w will be referred to as the *Watson-Crick morphism*. Then the edge from x to y is encoded by the oligo $h_w(x_2y_1)$.

The experiment begins by forming a "DNA soup" containing, in large quantities, oligos of the vertices, as well as of the edges of the graph. Now the ligation reaction resulting from the Watson-Crick complementarity will link together compatible edges. This "domino game" continues, identifying longer and longer paths. In this way DNA molecules encoding random paths through the graph are formed. By a filtering procedure not of interest for us here, one can check whether or not paths satisfying HPP are present.

It will be seen below that any model of DNA computing, where the filtering procedures available can simulate gsm mappings, is capable of Turing machine computations. Opinions may differ as to whether computational universality is of practical relevance. It is quite reasonable to argue that one should not aim to fit DNA models too tightly to Turing models, but should rather try to completely rethink the notion of computation. For some specific classes of hard computational problems, Turing universality might not at all be important. On the other hand, in a specific model of DNA computing, where computational universality is unknown, one could suddenly face a situation that numerous practical efforts of solving a particular problem are proven to have been futile.

3 Universality of the twin-shuffle language

We begin with the explicit definition of the *twin-shuffle language* TS. Consider the binary alphabet $\{0,1\}$, as well as its "complement" $\{\bar{0},\bar{1}\}$. For a word x over $\{0,1\}$, we denote by $\bar{x}$ the word over $\{\bar{0},\bar{1}\}$, where every letter of x is replaced by its "barred version". (Observe that the bar defines in this fashion a letter-to-letter morphism of $\{0,1\}^*$ onto $\{\bar{0},\bar{1}\}^*$.) For instance, if $x = 001100$, then $\bar{x} = \bar{0}\bar{0}\bar{1}\bar{1}\bar{0}\bar{0}$.

For two words x and y, denote by $sh(x,y)$ the set of words obtained by *shuffling* x and y, without changing the order of letters in x or y. For instance, each of the words

$$0\bar{0}\bar{0}0\bar{1}\bar{1}\bar{1}\bar{0}0\bar{0}\bar{0}, \quad \bar{0}\bar{0}\bar{1}\bar{1}\bar{0}\bar{0}0001100, \quad \bar{0}00\bar{1}\bar{0}10\bar{1}\bar{1}\bar{0}\bar{0}\bar{0}$$

is in $sh(x,\bar{x})$, where $x = 001100$, but $00\bar{1}\bar{1}\bar{0}\bar{0}0001\bar{1}\bar{0}\bar{0}$ is not in $sh(x,\bar{x})$.

By definition, the language TS consists of all words in the sets $sh(y,\bar{y})$, where y ranges over all words over $\{0,1\}$. The following is a simple way of finding out whether or not a given word z, built from the four letters $0,1,\bar{0},\bar{1}$, actually is in TS. Erase first all occurrences of $\bar{0}$ and $\bar{1}$, leaving a word z'. Erase next all occurrences of 0 and 1, as well as the bars from the remaining letters, leaving a word z''. Then z is in TS if and only if $z' = z''$. Applying this technique to the word $00\bar{1}\bar{1}\bar{0}\bar{0}0001\bar{1}\bar{0}\bar{0} = z$ considered above gives

$$z' = 000011 \quad \text{and} \quad z'' = 110000,$$

which shows that z is not in TS.

Let us now go back to the alphabet $\Sigma_{DNA} = \{A, C, G, T\}$. We can view Σ_{DNA} as the union of the alphabets $\{0, 1\}$ and $\{\bar{0}, \bar{1}\}$ by identifying the letters as follows:

$$(*) \qquad A = 0 \; C = 1, \; G = \bar{1}, \; T = \bar{0}.$$

Moreover, in this identification, providing a numerical character with a bar has the same effect as applying the Watson-Crick morphism h_w to its counterpart. (We make here the obvious convention: $\bar{\bar{0}} = 0$ and $\bar{\bar{1}} = 1$.) Thus, the correspondence defined above is a natural one between the DNA-alphabet Σ_{DNA} and the union of the binary alphabet and its complement.

Consider now DNA double strands, for instance,

$$\begin{array}{cccccccccc} A & C & A & T & T & A & G & A & T & T \\ T & G & T & A & A & T & C & T & A & A. \end{array}$$

(We have omitted the orientation markers $5'$ and $3'$.) If we switch to the binary alphabet, this double strand reads

$$\begin{array}{l} 010\bar{0}\bar{0}0\bar{1}0\bar{0}\bar{0} \\ \bar{0}\bar{1}\bar{0}00\bar{0}1\bar{0}00. \end{array}$$

We want to read this double strand as a word over the alphabet $\{0, 1, \bar{0}, \bar{1}\}$. There are various conventions for doing this in such a way that the resulting word is uniquely determined and, moreover, one can go back from the resulting word to the original double strand. A natural convention is to follow the orientation: from left to right in the upper strand, and from right to left in the lower strand. This convention produces the word

$$(**) \qquad 010\bar{0}\bar{0}0\bar{1}0\bar{0}\bar{0}000\bar{0}1\bar{0}00\bar{0}\bar{1}\bar{0}.$$

Another convention would be to take letters from the two strands by turns, starting from the upper strand. In this way the word

$$(**)' \qquad 0\bar{0}1\bar{1}0\bar{0}\bar{0}0\bar{0}00\bar{0}\bar{1}10\bar{0}\bar{0}00$$

results. Observe that the word $(**)'$ is in TS, whereas $(**)$ is not in TS.

We will now investigate in general the relation between DNA double strands and words in TS. We assume that DNA double strands are written in the binary notation specified by $(*)$. This means that the double strands are of the form

$$\begin{array}{l} x_1 x_2 \ldots x_n \\ \bar{x_1}\bar{x_2} \ldots x_n \end{array}$$

where each $x_i \in \{0, 1, \bar{0}, \bar{1}\}$ and $\bar{\bar{0}} = 0$, $\bar{\bar{1}} = 1$. It is clear that the method used to produce $(**)'$ from a double strand, that is, to take letters by turns from the two strands, always yields a word in TS.

Similarly, the method used to produce (**) from a double strand always yields a word in the *reverse twin-shuffle language* RTS. By definition, RTS consists of all words in the sets $sh(y, \bar{y}^r\}$, where y ranges over all words over $\{0,1\}$ and $\bar{y}^r$ denotes the reverse of $\bar{y}$, that is, $\bar{y}$ written backwards.

Although the two methods considered yield words in TS and RTS, respectively, all words in these two languages cannot be obtained in this fashion. In fact, the method used to produce $(**)'$ yields only the subset $\{0\bar{0}, \bar{0}0, 1\bar{1}, \bar{1}1\}^*$ of TS; words having more than two consecutive non-barred or barred letters cannot be obtained. The method used to produce (**) yields only words of the form $y\bar{y}^r$, where $y \in \{0,1,\bar{0},\bar{1}\}^*$. (Here barring is extended to concern the letters $\bar{0}$ and $\bar{1}$, with our convention about double bars in mind.) Such words constitute a special subclass of words in RTS.

We have given explicit methods to produce words in TS or RTS from DNA double strands. The methods do not yield the whole language TS (or RTS). However, in view of the importance of TS as a basis for computation, we would like to have a method of producing from an arbitrary word in TS a DNA double strand. Such a method will now be described.

Let z be an arbitrary word in TS. Necessarily, z is of even length, $|z| = 2n$ for some $n \geq 1$. (We exclude here the empty word.) Moreover, the scattered subword z' (resp. z'') of z consisting of plain (resp. barred) symbols is of length n. For $1 \leq i \leq n$, we denote by z'_i (resp. z''_i) the ith letter of z' (resp. z''). Because z is in TS, we have, for each i, $1 \leq i \leq n$, $(z'_i, z''_i) = (0, \bar{0})$ or $(z'_i, z''_i) = (1, \bar{1})$. When we speak of z'_i and z''_i, we have these particular ith occurrences in mind. One of them occurs to the left of the other in z; this is always the case, although the occurrences might lie far apart. The left occurrence is referred to as the *up occurrence at position* i, the other occurrence similarly as the *down occurrence at position* i.

We now denote by $LP(z)$ the DNA double strand of length n, where for $1 \leq i \leq n$, the ith letter in the upper (resp. lower) strand is the up (resp. down) occurrence at position i in z. It is obvious that indeed a DNA double strand always results in this fashion, since the unordered pair consisting of an up and down occurrence is always one of the pairs $(0, \bar{0})$ and $(1, \bar{1})$. (The notation LP comes from "left parsing".)

For instance,

$$LP(\bar{0}\bar{1}\bar{1}011) = LP(\bar{0}0\bar{1}\bar{1}1) = LP(\bar{0}\bar{1}01\bar{1}1) = \begin{matrix} \bar{0}\bar{1}\bar{1} \\ 011 \end{matrix} = \begin{matrix} TGG \\ ACC \end{matrix}.$$

Thus, the mapping induced by LP is not injective. In fact, no injective mapping of the subset of TS consisting of words of length $2n$ into the set of DNA double strands of length n is possible. This is immediately seen by considering the cardinalities of the two sets.

Given a DNA double strand Z, denote by $UD(Z)$ ("up-down" in Z) the word in TS obtained by the method used in $(**)'$. (Take letters from the two strands of Z by turns, starting from the upper strand.) Similarly,

denote by $FO(Z)$ ("follow orientation" in Z) the word in RTS obtained by the method used in $(**)$. We know that $UD(Z)$ is a word in the subset $\{0\bar{0}, \bar{0}0, 1\bar{1}, \bar{1}1\}^*$. In fact, all words of this subset are obtained as images of UD. Moreover, the restriction of LP to the words in this subset equals the inverse of UD.

We have, thus, established the following result.

Theorem 3.1 *For any nonempty word x in the twin-shuffle language TS, $LP(x)$ is a unique DNA double strand. Conversely, for any DNA double strand Z, $UD(Z)$ is a unique word in the subset $\{0\bar{0}, \bar{0}0, 1\bar{1}, \bar{1}1\}^*$ of TS. When restricted to this subset, LP is the inverse of UD.*

Theorem 3.1 shows how to go from words in TS to DNA double strands, and vice versa. We now come to the universality of the language TS. For this we need the notion of a *gsm-mapping*. Here the abbreviation "gsm" comes from "generalized sequential machine"; the acronym was used in computer science long before the era of mobile phones. Such a machine is a strictly finitary device which translates input words into output words according to the simple instructions $(s, x) \rightarrow (s', y)$; when reading x from the input in the state s, produce the output y and go to the state s'. Here x and y are words, possibly empty, whereas s and s' come from a finite set of states. Each particular gsm is specified by a finite list of such instructions.

Every Turing machine (generally accepted as a universal model for computations) defines a language, the subset of acceptable ones among all possible input words. Thus, all computations are characterized by such Turing-machine-acceptable languages L_0. The universality of the language TS, and hence by Theorem 3.1, the universality of DNA double strands, is based on the following theorem. The theorem is due to [4], see also [18].

Theorem 3.2 *For every Turing-machine-acceptable language L_0, a gsm-mapping g such that*

$$L_0 = g(TS)$$

can be effectively constructed.

Thus, TS is universal because it remains always the same, whereas g is specified according to the needs of each particular L_0. An elegant argument given in [3] shows that Theorem 3.2 remains valid also when TS is replaced by RTS. Thus, the representation theorem is quite robust in the sense that one can replace TS by RTS. This is important because one could argue that the reading of the strands should proceed in the "natural" $5'$ to $3'$ direction. Hence if in the "upper strand" (in our representation of double strands) left-to-right corresponds to $5'$ to $3'$ direction, then we have to read the lower strand in the right-to-left direction. This leads to the mapping FO discussed above, and to words in RTS.

Although the gsm-mapping g can in principle be constructed for every language L_0, it is still a major task in genetic engineering to make g feasibly realizable. In a general model of DNA computing, it might be advisable to let some features of the underlying gsm-computing be carried out by conventional computers. A step in this direction was already taken in [14]. Altogether the representation $L_0 = g(TS)$ is flexible in the sense that, given a computational problem, there is usually much leeway in choosing L_0 and, consequently, g. Which types of computational techniques or, theoretically, which aspects of gsm-mappings are adaptable to DNA-computing? This is the key problem in developing DNA based computers. It is also essential to make known algorithms more suitable for DNA computing, as already done in [2].

Some fundamental questions remain so far unanswered. The essential mathematical content of the representation theorem remains unchanged under many restrictions concerning the appearance of the bases in the two strands. For instance, we might require that in the strand we start with initially (that is, in the "upper strand") only non-barred letters (A and C) appear, or only purines (A and G) appear. Is it biologically feasible to have strands only with A's and C's? What is their computational significance, especially from the point of view of computational complexity?

The representation result, Theorem 3.2, is very fundamental, yet the basic idea behind its proof is very conspicuous, [18]. Perhaps this reflects also the simplicity of the basic DNA structure.

4 Matching systems

We now briefly indicate some formal models, where Watson-Crick complementarity and, consequently, at least to some extent also the ideas presented in the preceding section are visible. Following the customary duality in automata and formal language theory, we consider in this section a generative, grammar-like model, whereas the next section is devoted to an accepting, automaton-like model. Our discussion here is mainly on an intuitive level, proofs and formal details can be found in [5], [8]. Watson-Crick complementarity is of course in some sense present in all studies about DNA computing; ideas relevant to matching systems are contained in [9], [14].

Consider an alphabet V and a special symbol # (*blank*) not in V. Let ρ be a symmetric binary relation, called the relation of *complementarity*, defined on V. We consider also *composite* symbols in the sets

$$\begin{pmatrix} V \\ V \end{pmatrix} = \left\{ \begin{pmatrix} a \\ b \end{pmatrix} \mid a, b \in V,\ \rho(a,b) \text{ holds} \right\},$$

$$\begin{pmatrix} \# \\ V \end{pmatrix} = \left\{ \begin{pmatrix} \# \\ a \end{pmatrix} \mid a \in V \right\},$$

$$\begin{pmatrix} V \\ \# \end{pmatrix} = \left\{ \begin{pmatrix} a \\ \# \end{pmatrix} \mid a \in V \right\},$$

as well as strings of composite symbols in the sets

$$W(V) = \begin{pmatrix} V \\ V \end{pmatrix}^+ S(V), \text{ where } S(V) = \begin{pmatrix} \# \\ V \end{pmatrix}^* \bigcup \begin{pmatrix} V \\ \# \end{pmatrix}^*.$$

The strings in $W(V)$ are called *well-started sequences.* If a well-started sequence actually belongs to $\binom{V}{V}^+$, it is referred to as *blunt.* Elements in the first (resp. second) set of the union defining $S(V)$ are referred to as *down* (resp. *up*) *strands.*

A *matching system* is a construct

$$M = (V, \rho, A, B_d, B_u),$$

where V and ρ are as above, A (*axioms*) is a finite subset of $W(V)$, and B_d (resp. B_u) is a finite set of down (resp. up) strands.

We now define a *derivation step* according to the matching system M. Consider a well-started sequence

$$\alpha = \begin{matrix} a_1 & a_2 & \dots & a_k & a_{k+1} & \dots & a_{k+l} \\ b_1 & b_2 & \dots & b_k & \# & \dots & \# \end{matrix}.$$

We have omitted the parentheses from the symbols. Of the two symmetric possibilities, we have chosen the one where the blanks are *down*, implying that we have to use an element of B_d, say

$$\begin{matrix} \# & \dots & \# \\ c_1 & \dots & c_p \end{matrix}.$$

If $\rho(a_{k+i}, c_i)$ holds for $1 \leq i \leq p$ (in case $p > l$ this is required only for $1 \leq i \leq l$), we write $\alpha \Rightarrow \beta$, where β results from α by replacing the first p #'s by $c_1, \dots, c_p$, in this order. In case $p > l$, we have

$$\beta = \begin{matrix} a_1 & \dots & a_k & a_{k+1} & \dots & a_{k+l} & \# & \dots & \# \\ b_1 & \dots & b_k & c_1 & \dots & c_{k+l} & c_{k+l+1} & \dots & c_p \end{matrix}.$$

If the blanks are *up* in the original α, an element of B_u has to be used in the same way in order to obtain β. If $\alpha \Rightarrow \beta$ we say that α *yields directly* β. If $\alpha \Rightarrow^* \beta$, where $\Rightarrow^*$ is the reflexive transitive closure of $\Rightarrow$, we say that α *yields* β. In a *complete* derivation $\alpha \Rightarrow^* \beta$, α is an axiom and β is blunt.

For instance, assume that V is the DNA alphabet $V = \{A, C, T, G\}$ and ρ is the Watson-Crick complentarity. A matching system M is defined by taking $\binom{CAT}{G\#\#}$ as the only axiom, letting B_d consist of the sole element $\binom{\#\#\#\#}{TACA}$ and letting B_u consist of the two elements $\binom{GT}{\#\#}$ and $\binom{AT}{\#\#}$. Then

$$\begin{matrix} CAT \\ G\#\# \end{matrix} \Rightarrow \begin{matrix} CAT\#\# \\ GTACA \end{matrix} \Rightarrow \begin{matrix} CATGT \\ GTACA \end{matrix}$$

is a complete derivation according to M. It is not the only complete derivation, because the blunt element at the end can be continued by the element of B_d, after which the two elements of B_u become again applicable.

We now return to the general discussion. We want to "squeeze out" a language from the last words of complete derivations. This means the elimination of composite symbols. A gsm-mapping of a special kind will be suitable for this purpose. Let T be an alphabet, $h : T^* \to \binom{V}{V}^*$ a morphism and $h' : T^* \to T^*$ a morphism mapping each letter to a letter or to the empty word. Consider again the matching system M defined above. The *language generated* by the triple (M, h, h') consists of all words $h'(h^{-1}(w))$, where w runs over last words of complete derivations according to M. In symbols,

$$L(M, h, h') = \{h'(h^{-1}(w) | x \Rightarrow^* w,\ x \in A,\ w \in \binom{V}{V}^*\}.$$

Observe that the composition $h'h^{-1}$ is a gsm-mapping of a special kind. The part h^{-1} is used, for instance, when a double strand is applied to encode a vertex, just as in Adleman's experiment. The part h' is needed to erase unnecessary items.

By imposing further restrictions on complete derivations, we get certain subsets of the language $L(M, h, h')$. A complete derivation is *primitive* if no proper initial part of it is complete. Assume next that the elements in the sets B_u, B_d are labelled by numbers from 1 to $\text{card}(B_i)$, $i = u, d$. Consider a complete derivation D. The *u-control word* $ctrl_u(D)$ is obtained by writing, in the proper order, the labels of the elements of B_u applied in D. The *d-control word* $ctrl_d(D)$ is defined similarly. The derivation D is *coherent* (resp. *fair*) is $ctrl_u(D) = ctrl_d(D)$ (resp. $|ctrl_u(D)| = |ctrl_d(D)|$). The index i in $L_i(M, h, h')$, where $i = p, c, f$, indicates that last words in only primitive, coherent or fair derivations, respectively, are taken into consideration.

We are now ready to state the main results concerning matching systems. We denote by RE (resp. REG) the family of recursively enumerable (resp. regular) languages. The notation ML is used for languages generated by matching systems, that is, languages $L(M, h, h')$. Finally, the notation M_iL, where $i = p, c, f$, stands for languages $L_i(M, h, h')$.

Theorem 4.1 $ML = M_pL = REG$.

Theorem 4.2 $REG \subset M_fL \subset RE$.

Theorem 4.3 $M_cL = RE$.

Theorems 4.1–4.3 show that we get devices of varying strength based on matching systems, depending on the squeezing mechanisms applied. The

universality result Theorem 4.3 can be seen as a manifestation of Theorems 3.1 and 3.2. Indeed, TS is present in that coherence and the complementarity in composite symbols guarantee that two disguises of the same word are shuffled.

5 Automata on double strands

We are now ready to discuss the automata-theoretic counterpart of matching systems. The automata are fundamentally new in the sense that they work on data structures generalized from DNA double strands. In comparison with ordinary linear data structures, strings, such double strands increase the computational power. This depends on the complementarity relation ρ which gives us something "for free": whenever we know a letter a on one of the strands, we know also that the corresponding letter on the other strand is in the relation ρ to a. An automaton reading the two strands separately does not have to check this in any way but can use it as "freely given information".

We consider here only the *finite automaton* on *double strands*. Other models such as transducers and pushdown automata have been introduced in [5]. By definition, a *finite automaton on double strands* is a construct

$$DS = (V, \rho, Q, q_0, F, \delta).$$

Here V and ρ are as in the preceding section, Q is a finite set (*states*), $q_0 \in Q$ (*initial state*), $F \subseteq Q$ (the set of *final* states) and δ is a finite set of quadruples (q, x, y, q'), where $q, q' \in Q$ and $x, y \in V^*$.

The automaton DS receives as its inputs double strands in $\binom{V}{V}^*$. It reads both strands independently from left to right, beginning in the initial state. It *accepts* a given input if it is in a final state, after having reached the right end in both strands. The behaviour is nondeterministic. The quadruples in δ are viewed as instructions: if DS is in the state q and scans x from the upper and y from the lower strand, it may pass x and y and go to the state q'.

All *upper* strands in the accepted inputs constitute a language, denoted by $L_u(DS)$. We can also label the instructions in δ and consider strings of labels coming from accepting computations. The language of such strings, the *control language* of DS, is denoted by $L_{ctrl}(DS)$. The family of languages of the form $L_i(DS)$, where $i = u, ctrl$, is denoted briefly by DS_i.

We consider also certain special types of automata DS and the resulting subfamilies of DS_i. An automaton DS is *without states* (appreviated ws) if $Q = \{s_0\} = F$. It is *all-final* (af) if $Q = F$. It is *single-reading* (sr) if, for every (q, x, y, q') in δ, we have either $x = \lambda$ or $y = \lambda$. Thus, we speak of the families

$$(*) \qquad DS_i,\ wsDS_i,\ afDS_i,\ srDS_i,\ wssrDS_i,\ afsrDS_i,$$

Transactions on Engineering Sciences vol 15, © 1997 WIT Press, www.witpress.com, ISSN 1743-3533

where $i = u, ctrl$.

We finally present, without any detailed comments, a selection of results, [5], showing the extent of the families introduced. The first result deals with the languages $L_{ctrl}(DS)$.

Theorem 5.1 *For each $j \in \{\lambda, ws, af, sr, wssr, afsr\}$, an arbitrary recursively enumerable language is a gsm-image of a language in jDS_{ctrl}.*

The next result concerns the languages $L_u(DS)$. A morphism is called a *weak coding* (resp. *projection*) if it maps every letter either to a letter or to the empty word λ (resp. either to itself or to the empty word).

Theorem 5.2 *For each $j \in \{\lambda, af, sr\}$, an arbitrary recursively enumerable language is a weak coding of a language in jDS_u. For each $j \in \{ws, afsr\}$, an arbitrary recursively enumerable language L_0 can be expressed in the form $L_0 = h(L \cap R)$, where $L \in jDS_u$, R is regular and h is a projection.*

The preceding theorems give evidence of the extent of the families (*). This evidence is made still more explicit in our final theorem. It shows that none of the families (*) where $i = u$, with the possible exception of the family $wssrDS_u$, is contained in certain classical families in language theory, such as ETOL.

Theorem 5.3 *Assume that $\mathcal{L}$ is a family of languages strictly contained in RE and closed under intersection with regular languages and arbitrary morphisms. Then, for each $j \in \{\lambda, ws, af, sr, afsr\}$, the difference $jDS_u - \mathcal{L}$ is nonempty.*

References

1. Adleman, L. Molecular computation of solutions to combinatorial problems, *Science*, 1994, **266**, 1021–1024.

2. Boneh, D., Dunworth, C., Lipton, R. & Sgall, J. On the computational power of DNA, *Discrete Appl. Math.*, 1996, **71**, 79–94.

3. Engelfriet, J. Reverse twin shuffles, *Bulletin of EATCS*, 1996, **60**, 144.

4. Engelfriet, J. & Rozenberg, G. Fixed point languages, equality languages and representation of recursively enumerable languages, *J. Assoc. Comput. Mach.*, 1980, **27**, 499–518.

 Transactions on Engineering Sciences vol 15, © 1997 WIT Press, www.witpress.com, ISSN 1743-3533

5. Freund, R., Paun, G., Rozenberg, G. & Salomaa, A. Automata on DNA-like double strands, submitted.

6. Gifford, D. On the path to computation with DNA, *Science*, 1994, **266**, 993–994.

7. Kari, L. DNA computing: the arrival of biological mathematics, *Mathematical Intelligence*, to appear.

8. Kari, L., Paun, G., Rozenberg, G., Salomaa, A. & Yu, S. DNA computing, matching systems and universality, *TUCS Technical Report*, 1996, **49**.

9. Mihalache, V. Prolog approach to DNA computing, *Proceedings of ICEC'97*, Indianapolis, forthcoming.

10. Paun, G., Rozenberg, G. & Salomaa, A. Computing by splicing: programmed and evolving splicing systems, *Proceedings of ICEC'97*, Indianapolis, forthcoming.

11. Paun, G., Rozenberg, G. & Salomaa, A. *DNA Computing – New Computability Paradigms*, in preparation.

12. Paun, G. & Salomaa, A. DNA computing based on the splicing operation, *Mathematica Japonica*, 1996, **43**, 607–632.

13. Paun, G. & Salomaa, A. From DNA recombination to DNA computing via formal languages, *TUCS Technical Report*, 1996, **43**; *Springer LNCS*, to appear.

14. Roweis, S., Winfree, E., Burgoyne, R., Chelyapov, N., Goodman, M., Rothemund, P. & Adleman, L. A sticker based architecture for DNA computation, *Proceedings of the 2nd Ann. Princeton Conf. on DNA Based Computers*, 1996, 1–27.

15. Rozenberg, G. & Salomaa, A. Watson-Crick complementarity, universal computations and genetic engineering, *Leiden University, Computer Science Technical Report*, 1996, **28**.

16. Rozenberg, G. & Salomaa, A. (ed.). *Handbook of Formal Languages*, Vol. 1–3, Springer-Verlag, Berlin, Heidelberg, New York, 1997.

17. Salomaa, A. *Formal Languages*, Academic Press, New York, 1973.

18. Salomaa, A. *Jewels of Formal Language Theory*, Computer Science Press, Rockville, Md., 1981.