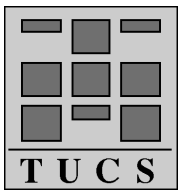
From DNA Recombination to DNA Computing Via Formal Languages

Gheorghe Păun
Institute of Mathematics of the Romanian Academy
PO Box 1-764, 70700 Bucureşti, Romania

Arto Salomaa
Academy of Finland and Turku University
Department of Mathematics, 20014 Turku, Finland



Turku Centre for Computer Science
TUCS Technical Report No 43
September 1996
ISBN 951-650-836-7
ISSN 1239-1891

Abstract

We briefly present notions and results from three directions of research which use formal language theory tools for modelling operations specific to DNA (and RNA) recombinations; in all cases one obtains computability models which are universal (language generating devices are obtained which are equivalent in power with Turing machines). The basic operations are those of *matching* (a model of the Watson-Crick complementarity), of *splicing* (a model of the recombinant behaviour of DNA sequences under the influence of restriction enzymes), and of *insertion/deletion* (known to hold both for DNA and for RNA sequences).

TUCS Research Group
Mathematical Structures of Computer Science

1 Matching systems

The matching systems introduced in [10] start from the basic ingredient of Adleman's successful experiment of computing a Hamiltonian path in a graph by handling DNA sequences, [1], the operation of prolonging a sequence of (single or double) symbols by using given single stranded strings, matching them with portions of the current sequence according to a complementarity relation. We do not repeat here the details of [1], but introduce directly our model.

Consider an alphabet V (a finite set of abstract symbols) augmented with a symmetric relation ρ (of *complementarity*), $\rho \subseteq V \times V$. Consider also a special symbol, #, not in V, denoting an empty space (the *blank* symbol).

Using the elements of $V \cup \{\#\}$ we construct the *composite* symbols of the following sets:

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

We denote

$$W(V) = \begin{pmatrix} V \\ V \end{pmatrix}^* S(V),$$

where

$$S(V) = \begin{pmatrix} \# \\ V \end{pmatrix}^+ \cup \begin{pmatrix} V \\ \# \end{pmatrix}^+,$$

and we call the elements of $W(V)$ *well-started sequences*. Stated otherwise, the elements of $W(V)$ start with pairs of symbols in V, as selected by the complementarity relation, and end either with a suffix consisting of pairs of the form $\begin{pmatrix} \# \\ a \end{pmatrix}$ or with a suffix consisting of pairs $\begin{pmatrix} b \\ \# \end{pmatrix}$, for $a, b \in V$ (the symbols $\begin{pmatrix} \# \\ a \end{pmatrix}$, $\begin{pmatrix} b \\ \# \end{pmatrix}$ are not mixed).

The matching operation, denoted by μ, is a partially defined mapping from $W(V) \times S(V)$ to $W(V)$, defined as suggested in Figure 1: In case 1 we add complementary symbols on the lower level, possibly without completing all the blank spaces, in case 2 we complete the blank spaces on the lower level of x and we still add some composite symbols of the form $\begin{pmatrix} \# \\ c \end{pmatrix}$. Cases 3 and 4 are symmetric, completing blank spaces on the upper level of the string. In all cases the string y must contain at least one composite symbol. In cases 2 and 4 we also allow the prolongation of "blunt" strings in $W(V)$,

that is with no blank position in x. Of course, for strings x, y which do not fulfil the previous conditions, $\mu(x, y)$ is not defined.

We define here formally the string $\mu(x, y)$ only in the first case, the others are similar. For $x \in W(V), y \in S(V), z \in W(V)$ we write $\mu(x, y) = z$ if

$$x = \binom{a_1}{b_1} \ldots \binom{a_k}{b_k} \binom{a_{k+1}}{\#} \ldots \binom{a_{k+r}}{\#} \binom{a_{k+r+1}}{\#} \ldots \binom{a_{k+r+p}}{\#},$$

$$y = \binom{\#}{c_1} \ldots \binom{\#}{c_r},$$

$$z = \binom{a_1}{b_1} \ldots \binom{a_k}{b_k} \binom{a_{k+1}}{c_1} \ldots \binom{a_{k+r}}{c_r} \binom{a_{k+r+1}}{\#} \ldots \binom{a_{k+r+p}}{\#},$$

for $k \geq 0, r \geq 1, p \geq 1, a_i \in V, 1 \leq i \leq k + r + p,\ b_i \in V, 1 \leq i \leq k,$ $c_i \in V, 1 \leq i \leq r,$ and $(a_{k+i}, c_i) \in \rho, 1 \leq i \leq r.$

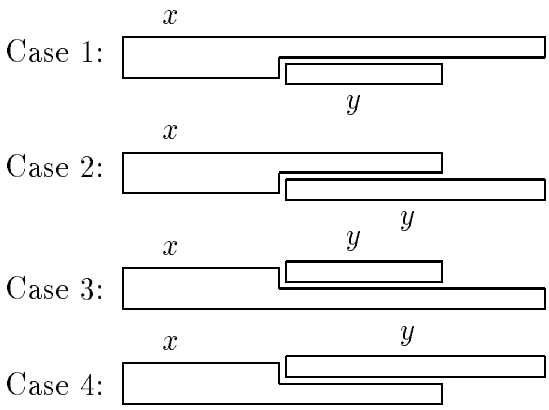


Figure 1

Using the matching operation we can define a generative/computing mechanism as follows. A *matching system* is a construct

$$\gamma = (V, \rho, T, f, g, A, B_d, B_u),$$

where V is an alphabet, $\rho \subseteq V \times V$ is a symmetric relation on V, T is an alphabet, $f : T \longrightarrow T \cup \{\lambda\}$ is a weak coding, $g : T^* \longrightarrow \binom{V}{V}^*$ is a morphism, A is a finite subset of $W(V)$ (of axioms), and $B_d \subseteq \binom{\#}{V}^+,\ B_u \subseteq \binom{V}{\#}^+$.

The idea behind considering such a machinery is the following. We start with the sequences in A and we prolong them to the right hand according to the strings in B_d, B_u, using the matching operation (the elements of B_d are used on the lower row, *down*, and those of B_u are used on the *upper*

row). When no blank symbol is present, we associate a string in T^* to the sequence obtained, by means of the mappings f, g: by g^{-1} we map blocks of the sequence into symbols of T, then certain symbols are erased by f or renamed. The language of all such strings is the language generated by γ.

Formally, we define this language as follows. For two strings $x, z \in W(V)$ we write

$$x \Longrightarrow z \text{ iff } z = \mu(x, y) \text{ for some } y \in B_d \cup B_u.$$

We denote by $\Longrightarrow^*$ the reflexive and transitive closure of the relation $\Longrightarrow$.

A sequence $x_1 \Longrightarrow x_2 \Longrightarrow \ldots \Longrightarrow x_k, x_1 \in A$, is called a *computation* in γ (of length $k-1$). A computation as above is *complete* when $x_k \in \binom{V}{V}^*$ (no blank symbol is present in the last string of composite symbols).

The language generated by γ, denoted by $L(\gamma)$, is defined by

$$L(\gamma) = \{f(g^{-1}(w)) \mid x \Longrightarrow^* w,\ x \in A, w \in \binom{V}{V}^*\}.$$

Therefore, only the complete computations are taken into account when defining $L(\gamma)$. Note that a complete computation can be continued, because we allow prolongations starting from blunt sequences.

One sees the close resemblance with the operations used in the Adleman experiment: B_d corresponds to the codes of graph nodes, B_u corresponds to the complementary strings identifying the arrows in the graph (or conversely), the morphism g, by its associated inverse morphism g^{-1}, assigns a "meaning" to the blocks corresponding to the graph nodes in the double stranded sequences. The use of such an inverse morphism cannot be avoided if we want to remain close to DNA manipulation by Watson-Crick complementarity, because we have to codify the information using blocks of nucleotides (not by single nucleotides). The fact that we use here also a given set of axioms and a weak coding f adds flexibility to the model and makes it more similar to usual generative mechanisms investigated in the formal language theory.

A complete computation $x_1 \Longrightarrow x_2 \Longrightarrow \ldots \Longrightarrow x_k, x_1 \in A, x_k \in \binom{V}{V}^*$, with respect to γ, is said:

- *primitive* if no properly initial part of it is complete;
- *balanced* if in each step $x_i \Longrightarrow x_{i+1}$ one uses a matching operation corresponding to cases 2 or 4 in Figure 1, namely alternating from a step to the next one (from a step to the next one we have to change the set B_d, B_u from which we take the used string).

Thus, in a primitive computation we do not use matching operations as in cases 1 – 4 with $p = 0$, except in the last step.

Let us denote by $L_p(\gamma), L_b(\gamma), L_{pb}(\gamma)$ the language of the strings $f(g^{-1}(w))$, for w obtained by a complete computation in γ which is primitive, balanced, respectively both primitive and balanced.

Assume now that the strings in the sets B_d, B_u are labelled in a one-to-one manner by natural numbers from 1 to $\text{card}(B_\alpha)$, $\alpha \in \{d, u\}$; denote by $e_\alpha : B_\alpha \longrightarrow \{1, \ldots, \text{card}(B_\alpha)\}, \alpha \in \{d, u\}$, the labellings. For a computation $D : x_1 \Longrightarrow x_2 \Longrightarrow \ldots x_k,\ x_1 \in A$, with $x_k \in \binom{V}{V}^*$, and for $1 \leq j \leq k-1$, we denote

$$e_\alpha(x_j \Longrightarrow x_{j+1}) = \begin{cases} e_\alpha(y), & \text{if } x_{j+1} = \mu(x_j, y), y \in B_\alpha, \\ \lambda, & \text{otherwise,} \end{cases}$$

and we define

$$e_\alpha(D) = e_\alpha(x_1 \Longrightarrow x_2) e_\alpha(x_2 \Longrightarrow x_3) \ldots e_\alpha(x_{k-1} \Longrightarrow x_k),$$

for $\alpha \in \{d, u\}$. We say that $e_d(D)$ is the d-control word and $e_u(D)$ is the u-control word associated with D.

A computation D such that $e_d(D) = e_u(D)$ is called *coherent*. When $|e_d(D)| = |e_u(D)|$ (where $|x|$ is the length of the string x) we say that D is a *fair* computation.

We denote by $L_c(\gamma), L_f(\gamma)$ the languages of the strings $f(g^{-1}(w))$, for w obtained by a coherent complete computation, respectively, by a fair complete computation in γ.

We denote by *ML, PML, BML, PBML, CML, FML* the families of languages of the form $L(\gamma), L_p(\gamma), L_b(\gamma), L_{pb}(\gamma), L_c(\gamma), L_f(\gamma)$, respectively, defined as above.

Proofs of the following results can be found in [10] (*REG, CF, CS, RE* are the four families in Chomsky hierarchy, *FIN* is the family of finite languages).

Theorem 1. $REG = ML = PML = BML = PBML$.

(The Adleman way of computing cannot transgress the power of finite automata.)

Theorem 2. $RE = CML$.

(The coherence restriction proves to be very powerful, leading to universal computability. The proof of this result is based on characterizations of recursively enumerable languages starting from equality sets, see, e.g., [25].) The fair computations lead to an intermediate case:

Theorem 3. $REG \subset FML \subset RE$.

2 Splicing systems

When two enzymes cut (at sites defined by well-specified associated patterns) two DNA (double stranded) sequences, leaving sticky ends which match (the single stranded ends are complementary in the Watson-Crick sense), then the obtained fragments can be recombined (by ligasion). The formal model of this phenomenon is the *splicing* operation, introduced in [8].

Consider an alphabet V and two special symbols, $\#, \$$, not in V. A *splicing rule* (over V) is a string $r = u_1\#u_2\$u_3\#u_4$, where $u_i \in V^*, 1 \leq i \leq 4$. For such a rule r and strings $x, y, z \in V^*$ we define

$$\begin{aligned}(x, y) \vdash_r z \quad \text{iff} \quad & x = x_1u_1u_2x_2,\ y = y_1u_3u_4y_2, \\ & z = x_1u_1u_4y_2, \text{ for some } x_1, x_2, y_1, y_2 \in V^*.\end{aligned}$$

A pair $\sigma = (V, R)$, where V is an alphabet and $R \subseteq V^*\#V^*\$V^*\#V^*$ is called an *H scheme*. For an H scheme $\sigma = (V, R)$ and a language $L \subseteq V^*$, we define

$$\begin{aligned}&\sigma(L) = \{z \in V^* \mid (x, y) \vdash_r z, \text{ for some } x, y \in L, r \in R\}, \\ &\sigma^0(L) = L,\ \sigma^{i+1}(L) = \sigma^i(L) \cup \sigma(\sigma^i(L)),\ i \geq 0, \\ &\sigma^*(L) = \bigcup_{i \geq 0} \sigma^i(L).\end{aligned}$$

In this way we obtain two operations with languages, σ (one-step splicing) and σ^* (iterated splicing). Their properties (relationships with other operations and closure properties of abstract families of languages – hence also of families in Chomsky hierarchy) are relatively well undersood. The reader can find a survey of results and bibliographical information in [13]. We only mention two results:

Lemma 1. ([5], [24]) *If $\sigma = (V, R)$ has a finite set R and $L \in REG$, then $\sigma^*(L) \in REG$.*

Lemma 2. ([19]) *Let F be a family of languages closed under intersection with regular languages and restricted morphisms. For any $L \subseteq V^*, L \notin F$, and $c, d \notin V$, consider the language*

$$L' = (dc)^*L(dc)^* \cup c(dc)^*L(dc)^*d.$$

Then there is no H scheme $\sigma = (V, R)$, no matter which is the type of R, and $L_0 \in F$ such that $L' = \sigma^*(L_0)$.

The previous two lemmas show that the splicing alone cannot provide characterizations of "large" families of languages. However, a simple squeezing mechanism, as usual in Chomsky grammars and in Lindenmayer systems,

can fill in this gap. This leads to the appealing notion of an *extended H system*, as introduced in [21].

Such a system is a quadruple $\gamma = (V, T, A, R)$, where V is an alphabet, $T \subseteq V$ (terminal symbols), $A \subseteq V^*$ (axiom set), and $R \subseteq V^*\#V^*\$V^*\#V^*$, for $\#, \$$ not in V; $\sigma = (V, R)$ is the *underlying H scheme* of γ. The *language generated* by γ is defined by

$$L(\gamma) = \sigma^*(A) \cap T^*.$$

For two families of languages, F_1, F_2, we denote by $EH(F_1, F_2)$ the family of languages $L(\gamma)$, for $\gamma = (V, T, A, R)$ with $A \in F_1$ and $R \in F_2$ (note that both A and R are sets of strings, i.e. languages, hence the definition makes sense).

Two important results about these families are:

Lemma 3. $EH(FIN, FIN) = REG$.

Lemma 4. (The Basic Universality Theorem) $EH(FIN, REG) = RE$.

Lemma 3 follows from Lemma 1 and the closure of REG under intersection, Lemma 4 is proved in [15]. The simple step from finite sets of splicing rules to regular sets entails the jump from REG to RE.

From the proof of the Basic Universality Theorem one can see that an equality as that in Lemma 4 can be obtained by using extended H systems with finite sets of splicing rules, having associated control mechanisms of the following types:

- *permitting contexts* (pc): each rule is given in a triple $(r; C_1, C_2)$, where C_1, C_2 are sets of symbols; a splicing $(x, y) \vdash_r z$ is allowed only when all symbols in C_1 appear in x and all symbols in C_2 appear in y;

- *forbidding contexts* (fc): as above, but no symbol of C_1 should appear in x and no symbol of C_2 should appear in y;

- *local targets* (lt): each rule is given as a pair (r, Q), where Q is a regular set, and $(x, y) \vdash_r z$ is allowed only if $z \in Q$;

- *global targets* (gt): as above, with all target languages being equal;

- *fitness mapping* (fit): a mapping $\varphi : V^* \longrightarrow [0, 1]$ is given (actually, it is enough to have $\varphi : V^* \longrightarrow \{0, 1\}$) and $(x, y) \vdash_r z$ is allowed only for x, y with high enough values of $\varphi(x), \varphi(y)$ (this resembles considerations in the area of genetic algorithms).

Denoting by $EH(F_1, \alpha F_2)$ the family of languages $L(\gamma)$, for γ an extended H system with the axiom set in family F_1 and the rule set in family F_2, with the use of rules controlled according to $\alpha, \alpha \in \{pc, fc, lt, gt, fit\}$, we get

Theorem 4. $EH(FIN, \alpha FIN) = RE,\ \alpha \in \{pc, fc, lt, gt, fit\}$.

Moreover, from the proof we find that *universal* H systems of the previous types can be constructed, that is systems γ_u with all components fixed and able to simulate any given H system γ, after adding a *code* of the particular system γ to the axiom set of γ_u (so, γ can be "run" on γ_u, the "program" being a single new axiom added to the "computer" γ_u). This looks quite encouraging – from a theoretical point of view – in what concerns the possibility of designing universal (hence programmable) DNA computers based on the splicing operation.

Proofs of the results summarized in Theorem 4 can be found in [7], [18], [22].

A very fruitful idea of how to reach the power of Turing machines using only finitely many splicing rules is the *distributed computing*, following suggestions from grammar system area. Particularly useful are the *parallel communicating grammar systems* introduced in [23] (see also [2]). They consist of several usual grammars (the *components* of the system) working synchronously on their own sentential forms (in each time unit each component uses a rewriting rule), and *communicating*, on request (this variant has been considered in [23]) or by command (a variant introduced in [4]). Communication has priority over rewriting. A component is designated as the *master* of the system and the language it generates, with the help of the other components, is the language of the system.

A direct counterpart of such a model are the *splicing grammar systems* introduced in [6]: the components are usual context-free Chomsky grammars; they rewrite their sentential forms as in a usual PC grammar system (componentwise, synchronously, starting from specific axioms) and "communicate" by splicing the sentential forms according to a given finite set of splicing rules; the splicing does not have priority over rewriting.

Denoting by $SGS_n(X)$ the family of languages generated by splicing grammar systems with at most $n, n \geq 1$, components, using rewriting rules of type X, we get

Theorem 5. ([14]) $CF = SGS_1(CF) \subset SGS_2(CF) = RE$.

These systems are quite hybrid, involving both rewriting and splicing operations. In *communicating distributed* H systems, introduced in [3], we use only splicing.

Such systems have as components triples of the form (A_i, R_i, V_i), where

A_i are finite sets of axioms, R_i are finite sets of splicing rules, and V_i are sets of symbols. The components work separately, on their *contents*, which initially are the sets A_i, according to the splicing schemes $\sigma_i = (V, R_i)$ (V is the alphabet of the system). This means an iterated splicing of the type σ_i^*. The communication is done in the WAVE style: in each moment, each string x produced by a component i is transmitted to any component j for which we have $x \in V_j^*$ (we say that x passes the filter defined by V_j). Copies of x are sent to all components j for which $x \in V_j^*$. The contents of a designated component contribute to the language generated by the system.

Let us denote by CDS_n the family of languages generated by such systems with at most $n, n \geq 1$, components (all components being finite).

Theorem 6. $CDS_1 \subset REG \subset CDS_2$, $CDS_3 - CF \neq \emptyset$, CDS_6 *contains non-recursive languages*, $CDS_{10} = RE$.

The equality $CDS_{10} = RE$ has been proved in [27]; in [3] it is only proved that $RE = \bigcup_{n \geq 1} CDS_n$.

Another distributed H system is introduced in [16] (*synchronized distributed H systems*), with the components having two types of axioms and of current strings – *active* and *non-active* – and two types of splicing rules – *internal* and *external*. The external rules act, with priority over the internal rules, on active strings only. We do not enter into details, but we only mention that denoting by SDS_n the family of languages generated by such systems with at most $n, n \geq 1$, components, we get

Theorem 7. $SDS_3 = RE$.

From the proofs of all these results we again obtain the existence of universal (hence programmable) H systems of the mentioned types.

3 Insertion/deletion systems

It is known that evolution is determined not only by recombination (crossovering), but also by local mutations, insertions and deletions of symbols or short strings in (from) the DNA sequences. Such operations are well-known in formal language theory; see details, for instance, in [20]. Using them, interesting generative devices can be defined. We present them in the form considered in [11].

An *insertion-deletion* (shortly, *insdel*) *system* is a construct

$$\gamma = (V, T, A, I, D),$$

where V is an alphabet, $T \subseteq V$, A is a finite subset of V^*, and I, D are finite subsets of $V^* \times V^* \times V^*$.

The alphabet T is the terminal alphabet of γ, A is the set of axioms, I is the set of insertion rules, and D is the set of deletion rules. An insertion/deletion rule is given in the form (u, z, v).

For $x, y \in V^*$ we write $x \Longrightarrow y$ iff one of the following two cases holds:

1. $x = x_1uvx_2$, $y = x_1uzvx_2$, for $x_1, x_2 \in V^*$, $(u, z, v) \in I$ (an insertion step);

2. $x = x_1uzvx_2$, $y = x_1uvx_2$, for $x_1, x_2 \in V^*$, $(u, z, v) \in D$ (a deletion step).

Denoting by $\Longrightarrow^*$ the reflexive and transitive closure of the relation $\Longrightarrow$, the language generated by γ is defined by

$$L(\gamma) = \{w \in T^* \mid x \Longrightarrow^* w, \text{ for some } x \in A\}.$$

An insdel system $\gamma = (V, T, A, I, D)$ is said to be of *weight* (n, m, p, q) if

$$\begin{aligned}
&\max\{|z| \mid (u, z, v) \in I\} = n,\\
&\max\{|u| \mid (u, z, v) \in I \text{ or } (v, z, u) \in I\} = m,\\
&\max\{|z| \mid (u, z, v) \in D\} = p,\\
&\max\{|u| \mid (u, z, v) \in D \text{ or } (v, z, u) \in D\} = q.
\end{aligned}$$

We denote by $INS_n^m DEL_p^q$, $n, m, p, q \geq 0$, the family of languages $L(\gamma)$ generated by insdel systems of weight (n', m', p', q') such that $n' \leq n$, $m' \leq m$, $p' \leq p$, $q' \leq q$. When one of the parameters n, m, p, q is not bounded, we replace it by ∞. Thus, the family of all insdel languages is $INS_\infty^\infty DEL_\infty^\infty$. Rules of the form (u, λ, v) are of no use, hence we ignore them. Thus, when $n = 0$ $(p = 0)$, then $m = 0$ $(q = 0)$.

In [12] one proves that:

(i). Each language $L \in RE$ can be written in the form $L = g(h^{-1}(L'))$, for a morphism h, a weak coding g, and $L \in INS_4^7 DEL_0^0$.

(ii). $RE = INS_3^2 DEL_3^0$.

In fact, in [12] no attention is paid to the length of contexts or of the inserted/deleted strings. By carefully observing such restrictions, in [11], one proves:

Theorem 8. $RE = INS_1^2 DEL_1^1 = INS_2^1 DEL_2^0 = INS_1^2 DEL_2^0$.

In the molecular computing framework we deal with a restricted number of symbols Moreover, the insertion/deletion of some of these symbols is easier

than for other symbols. Thus it is of practical interest to consider insdel systems with rules of the form (u, z, v) with $z \in c^*$ for a specified symbol c. In [26] one explicitly asks whether or not such systems can simulate any given Turing machine. Of course, because we can no longer introduce or erase symbols different from c, we must start with them introduced from the very beginning as a sort of workspace, whereas the result of a rewriting (we may also call it a *computation*) should be also embedded in a specified workspace. In such conditions, it is proved in [11] that the problem in [26] has a positive answer. Moreover, we can work with only one symbol different from c.

Specifically, a *restricted insdel system* is a construct

$$\gamma = (V, \{a, c\}, A, I, D, h),$$

where V is an alphabet, a, c are specified symbols (not necessarily from V), A is a finite subset of $\{a, c\}^*$, I, D are finite subsets of $\{a, c\}^* \times c^* \times \{a, c\}^*$, and $h : V^* \longrightarrow \{a, c\}^*$ is a morphism. (Note that all insertion-deletion rules are of the form $(u, c^i, v), u, v \in \{a, c\}^*, i \geq 0$.) The relation $\Longrightarrow$ is defined in the usual way, over $\{a, c\}^*$. Then, the language generated by γ is

$$L(\gamma) = h^{-1}(\{w \in \{a, c\}^* \mid z(aca)^n \Longrightarrow^* (aca)^m w, \text{ for some } n, m \geq 0, z \in A\}.$$

In words, we start from an axiom $z \in A$, prolonged with an arbitrary number of "empty spaces" aca, we use arbitrarily many insertion/deletion rules, we discard the "spaces" aca placed to the left hand end of the obtained string, and we map by h^{-1} the remaining string into a string in V^*. In this way, strings w for which $h^{-1}(w)$ is not defined are removed, hence we can ensure the termination of the derivation in the same way as when using a specified terminal alphabet.

We denote by $1INSDEL$ the family of languages generated by restricted insdel systems of arbitrary weight; because we work here with a codification of strings over V as strings over $\{a, c\}$, we cannot keep bounded (independent of the cardinality of V, for instance) the weight of the used systems.

Expected from the point of view of Theorem 7 and encouraging from DNA/RNA computing point of view, we have the following result.

Theorem 9. $RE = 1INSDEL$.

On the basis of the proofs of theorems above, universal insdel systems can be found, in the natural way.

4 Concluding remarks

We cannot enter here into details concerning the formal definitions and the proofs of the results mentioned above, or into a discussion concerning the

biochemical feasibility or unfeasibility of these models. Many features involved in these models look realistic: the hybridization and the prolongation operations involved in the matching systems are well controlled operations, the length of sites where the splicing is performed can be bounded by two, checking permitting conditions can be done by using primers which start the hybridization of single stranded DNA sequences making possible the enzyme action on the obtained double stranded sequence, intersection with T^* is a *separate* operation already used in laboratory, new copies of a sequence can be produced by PCR amplification, insertion can be realized by mismatching hybridization, etc. Other features (in general, the control of operations considered above) are far from the present day lab possibilities.

Acknowledgement: Research supported by the Academy of Finland, Project 11281

References

[1] L. M. Adleman, Molecular computation of solutions to combinatorial problems, *Science*, 226 (Nov. 1994), 1021 – 1024.

[2] E. Csuhaj-Varju, J. Dassow, J. Kelemen, Gh. Păun, *Grammar Systems. A Grammatical Approach to Distribution and Cooperation*, Gordon and Breach, London, 1994.

[3] E. Csuhaj-Varju, L. Kari, Gh. Păun, Test tube distributed systems based on splicing, *Computers and AI*, 15, 2-3 (1996), 211 – 232.

[4] E. Csuhaj-Varju, J. Kelemen, Gh. Păun, Grammar systems with WAVE-like communication, *Computers and AI*, to appear.

[5] K. Culik II, T. Harju, Splicing semigroups of dominoes and DNA, *Discrete Appl. Math.*, 31 (1991), 261 – 277.

[6] J. Dassow, V. Mitrana, Splicing grammar systems, *Computers and AI*, 15, 2-3, (1996), 109 – 122.

[7] R. Freund, L. Kari, Gh. Păun, DNA computing based on splicing: The existence of universal computers, *Technical Report 185-2/FR-2/95*, TU Wien, 1995.

[8] T. Head, Formal language theory and DNA: an analysis of the generative capacity of specific recombinant behaviors, *Bull. Math. Biology*, 49 (1987), 737 – 759.

[9] T. Head, Gh. Păun, D. Pixton, Language theory and molecular genetics. Generative mechanisms suggested by DNA recombination, in *Handbook of Formal Languages* (G. Rozenberg, A. Salomaa, eds.), Springer-Verlag, Berlin, Heidelberg, in preparation.

[10] L. Kari, Gh. Păun, G. Rozenberg, A. Salomaa, S. Yu, DNA computing by using matching systems, submitted, 1996.

[11] L. Kari, Gh. Păun, G. Thierrin, S. Yu, Characterizing RE using insertion-deletion systems, submitted, 1996.

[12] C. Martin-Vide, Gh. Păun, A. Salomaa, Characterizations of recursively enumerable languages by means of insertion grammars, submitted, 1996.

[13] Gh. Păun, Splicing. A challenge to formal language theorists, *Bulletin EATCS*, 57 (1995), 183 – 194.

[14] Gh. Păun, On the power of splicing grammar systems, *Ann. Univ. Buc., Matem.-Inform. Series*, 45, 1 (1996).

[15] Gh. Păun, Regular extended H systems are computationally universal, *J. Aut., Languages, Combinatorics*, 1, 1 (1996), 27 – 36.

[16] Gh. Păun, Computationally universal distributed systems based on the splicing operation, submitted, 1995.

[17] Gh. Păun, Universal DNA computing models based on the splicing operation, *Second Annual Meeting on DNA Based Computers*, Princeton, 1996, 67 – 86.

[18] Gh. Păun, Splicing systems with targets are computationally universal, *Inform. Processing Letters*, to appear.

[19] Gh. Păun, On the splicing operation, *Discrete Applied Math.*, 70 (1996), 57 – 79.

[20] Gh. Păun, *Contextual Grammars. From Natural Languages to Formal Languages and Back*, forthcoming (1997).

[21] Gh. Păun, G. Rozenberg, A. Salomaa, Computing by splicing, *Theor. Computer Sci.*, to appear.

[22] Gh. Păun, A. Salomaa, DNA computing based on the splicing operation, *Mathematica Japonica*, 43, 3 (1996), 607 – 632.

[23] Gh. Păun, L. Sântean (now Kari), Parallel communicating grammar systems: the regular case, *Ann. Univ. Buc., Matem.-Inform. Series*, 38 (1989), 55 – 63.

[24] D. Pixton, Regularity of splicing languages, *Discrete Appl. Math.*, 69 (1996), 101 – 124.

[25] A. Salomaa, *Jewels of Formal Language Theory*, Computer Science Press, Rockwille, Maryland, 1981.

[26] W. Smith, A. Schweitzer, DNA computers in vitro and in vivo, manuscript in circulation, March 1995.

[27] Cl. Zandrou, Distributed test tube systems versus RE, manuscript, 1996.

Turku Centre for Computer Science
Lemminkäisenkatu 14
FIN-20520 Turku
Finland

http://www.tucs.abo.fi

University of Turku
- **Department of Mathematical Sciences**

Åbo Akademi University
- **Department of Computer Science**
- **Institute for Advanced Management Systems Research**

Turku School of Economics and Business Administration
- **Institute of Information Systems Science**