Theory Comput. Systems 32, 69–112 (1999)

Theory of Computing Systems
© 1999 Springer-Verlag New York Inc.

DNA Computing Based on Splicing: The Existence of Universal Computers*

R. Freund[1], L. Kari[2], and Gh. Păun[3]

[1]Institute for Computer Languages, Technical University Wien, Resselgasse 3, 1040 Wien, Austria
freund@logic.tuwien.ac.at

[2]Department of Computer Science, University of Western Ontario, London, Ontario, Canada N6A 5B7
lkari@csd.uwo.ca

[3]Institute of Mathematics of the Romanian Academy, PO Box 1-764, 70700 Bucureşti, Romania
gpaun@imar.ro

Abstract. We prove that *splicing systems* with finite components and certain controls on their work are *computationally complete* (they can simulate any Turing Machine); moreover, there are *universal splicing systems* (systems with all components fixed which can simulate any given splicing system, when an encoding of the particular system is added—as a program—to the universal system).

Splicing systems are based on the *splicing operation* which is a model for DNA recombination. Informally, a prefix of a word is catenated to a suffix of another word, thus yielding a (possibly) new word. Cutting occurs at specific sites which correspond to specific sequences in DNA strands as they can be recognized by restriction enzymes.

When no additional control is assumed, splicing systems with finitely many starting words (axioms) and finitely many splicing rules are known to characterize only regular languages (those recognized by *finite automata*). However, when a splicing rule is allowed to be used

(1) only in the presence of certain symbols ("catalyst") or
(2) only in the absence of certain symbols ("inhibitors"),

* This research was supported by the Academy of Finland, Project 11281, and Grants OGP0007877 and OGP0000243 of the National Science and Engineering Research Council of Canada. All correspondence to Gheorghe Păun.

then we can characterize the recursively enumerable languages (recognized by *Turing Machines*); the same result is obtained when counting the number of copies of (some of) the words used. From the proofs, we also infer the existence of universal (hence programmable) splicing systems.

1. Introduction

1.1. *Goals and General Framework*

The main goal of this paper is to find computability models based on DNA manipulations and equal in power to Turing Machines.

We reach this goal by considering *splicing systems*, language-generating devices using as the basic operation the *splicing operation*, introduced in Head (1987) as a formal model for DNA recombination under the influence of restriction enzymes and ligases. (The splicing operation has been investigated widely in the last years as an abstract operation on words and languages; see references, e.g., in Păun (1995a) and in Head et al. (1997).)

We have to stress the fact that we look for *theoretical models of computability*, not for, say, practical *computer designs*; we discuss the assumptions on which the constructions and the proofs are based and we try to keep these constructions as close as possible to biochemical reality, but still our models are quite liberal and idealized in their assumptions, at least compared with the current laboratory possibilities. Moreover, we are not interested in the efficiency of our models (time, space, or other complexity measures), but in their *competence*, comparing their power with the power of existing computing devices (Turing Machines and restricted variants of them). On the other hand, as one can see in the paper, without such assumptions we cannot obtain computational completeness and universality.

The topic of our work falls in a general research trend, aiming at developing new types of algorithms and designing new types of computability models (maybe also computers) which differ from the classical Turing/von Neumann notions of algorithms and computers in a fundamental way. Special attention has been paid to biological-like computing, as illustrated by the well-developed area of neural computation and that of genetic algorithms (Davis, 1991; Hertz et al., 1991; Koza and Rice, 1992.)

Such a rather new area is that of DNA computing, which is based on the observation that the incredibly complex structure of a living being results from applying a few simple operations (local mutations, copying and splicing/recombination/crossing over) to an initial DNA sequence. The complexity of the output implies that these operations are powerful and of a very fundamental nature.

These observations, together with the continuously increasing feasibility of DNA synthesis and of manipulating it in laboratory conditions, suggest the idea that any complex computation can be carried out by starting from an initial information, encoded in a DNA-like sequence, by performing the operations mentioned above.

Indeed, at the end of 1994, this idea started to be implemented: Adleman (1994) reports the way of solving (a small instance of) the Hamiltonian path problem in a test tube, just by handling DNA words. A series of enthusiastic (yet also less enthusiastic) reactions followed, see Gifford (1994), Hartmanis (1995), and Lipton (1994), (1995). Lipton

(1995) describes the way to solve the satisfiability problem by DNA computing. At the beginning of 1995, Adleman (1995) developed the idea, proposing a formal framework (a sort of test tube programming language for encoding molecular algorithms).

Yet two fundamental problems have still remained open (or at least not answered in a satisfying manner):

1. Are the classes of DNA algorithms *computationally complete*, in the sense that *every* algorithm (hence every Turing Machine or every equivalent mechanism) can be encoded as a DNA algorithm in a specified class?
2. Are there *universal and programmable DNA computers*, i.e., fixed "test tube computers" that are able to run any arbitrarily given program/algorithm?

These problems are already formulated in the papers quoted above: "It seems likely that a single molecule of DNA can be used to encode the instantaneous description of a Turing Machine and that currently available protocols and enzymes could (at least under idealized conditions) be used to induce successive sequence modifications, which would correspond to the execution of the machine" (Adleman, 1994). "If we were 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. A great hope is that as we begin to understand how biological systems compute, we will identify a naturally occurring universal computational system" (Gifford, 1994).

It is somewhat intriguing why the problems mentioned above were not investigated for the "programming language" in Adleman (1995) (this is also true for genetic algorithms (Goldberg, 1989).

In this paper we investigate a class of computing (language-generating) mechanisms based on an operation specific to DNA recombination, i.e. the operation of *splicing*. For various types of such mechanisms we affirmatively solve both the problems mentioned above:

(1) these mechanisms are computationally complete, and
(2) there are universal mechanisms for each class we consider.

1.2. *The Organization of the Paper*

In the following subsection we informally introduce the splicing operation, as a model of DNA recombination under the influence of restriction enzymes and ligases. The information given should be sufficient for the computer scientists to follow the remaining sections of the paper, without any need for further biochemical knowledge. For further details, the computer scientist can consult expository papers like Hunter (1993) or monographs like Li and Graur (1991).

Then (Section 1.4) we try to make familiar, especially to the molecular biologist, some fundamental notions we use, such as language-generating devices, splicing systems (or H systems), and universal computing mechanisms. Of course, the paper cannot be made self-contained from this point of view if still keeping its length in reasonable limits, but the significance of the results (maybe not their proofs) and, mainly, the assumptions on which they rely should be clear just on the basis of these explanations.

In Section 2 we enter the technical part of the paper; the reader interested in the technical development of our approach can start from there. Section 2 introduces the notion of (extended) H systems and of the generated language, exemplifying the definitions. Section 3 introduces the notion of a multiset, then the power of H systems based on multisets is investigated. Section 4 considers H systems with permitting and forbidding context conditions.

In all cases we prove that characterizations of recursively enumerable languages (hence of the power of Turing Machines (resp. of type-0 Chomsky grammars)) are obtained. On the basis of the proofs of these results, in Section 5 we infer the existence of *universal* (hence programmable) H systems. We close the paper with some discussions about the significance of the results obtained.

All proofs are supplemented with informal discussions about the idea behind the formal construction, and about the biological assumptions and restrictions, yet without entering into feasibility and efficiency details. Thus, the paper is addressed more to computer scientists than to molecular biologists; as we have said above, first we are interested in *theoretical models of computability* and only then in their implementation in the form of *computers*.

1.3. *The Splicing Operation*

The fundamental notion we investigate here is that of *splicing*. We introduce it below, starting from the operation of DNA recombination, whose formal model the splicing operation is.

A word of warning: biochemists use this term also for a quite different operation on RNA sequences. The operation considered here corresponds to the cross-overing, as used in genetic algorithms, but it is much less restrictive: we allow the cross-overing of words of different length, in positions which may be different from one word to another one (moreover, these positions are defined according to certain contexts). In theoretical computer science (formal language theory) the term *splicing* is already well established; e.g., it is settled in this way in the *Handbook of Formal Languages* (Rozenberg and Salomaa, 1997).

We do not repeat here all the details of the abstraction process which leads from the actual DNA recombination to the formal operation of splicing. We only give the idea of this development; for details, the reader might consult Head, (1987) or Head et al. (1997).

As one knows, a DNA molecule is a complex molecule composed of four nucleotides, *adenyne*, *cytosine*, *guanine*, and *tymine*, abbreviated by A, C, G, T, respectively. These four nucleotides are always present in pairs A–T and C–G (the so-called Watson–Crick complementarity), hence the DNA molecule is in fact a double-stranded structure. Thus, we may interpret a DNA molecule as a word over the alphabet consisting of four two-level symbols:

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

DNA molecules are also characterized by their specific spatial arrangement, i.e., their helicoidal structure, but this is not important for our approach, hence in the following we interpret DNA sequences as *linear words.*

We consider two such words:

$$\begin{array}{ll} CCCCCTCGACCCCC & AAAAAGCGCAAAAA \\ GGGGGAGCTGGGGG & TTTTTCGCGTTTTT \end{array}$$

On DNA sequences there can act the so-called *restriction enzymes*, chemical compounds able to *recognize* specific subsequences of DNA and to *cut* the DNA sequence at that place. The behavior of enzymes (the pattern they recognize) is well known, there are catalogs with such informations. For instance, the enzymes named *Taq*I and *Sci*NI are characterized by the recognition sequences

$$\begin{array}{ll} TCGA & GCGC \\ AGCT & CGCG \end{array}$$

respectively. Their effect on the two molecules above is that the molecules are cut into the following four fragments:

$$\begin{array}{llll} CCCCCT & CGACCCCC & AAAAAG & CGCAAAAA \\ GGGGGAGC & \quad TGGGGG & TTTTTCGC & \quad GTTTTT \end{array}$$

One can see that the first fragment and the last one fit together, as well as the second and the third one; through *ligation*, two new molecules may arise:

$$\begin{array}{ll} CCCCCTCGCAAAAA & AAAAAGCGACCCCC \\ GGGGGAGCGTTTTT & TTTTTCGCTGGGGG \end{array}$$

The ligation operation is performed by other enzymes, called *ligases*, which just "paste" the matching ends of DNA sequences produced by the restriction enzymes.

The operation described above is called *recombination*, or *cross-overing*. The formal model of it is the *splicing operation*, informally defined as follows.

Consider an abstract *alphabet* V, and two *words* x, y composed of *symbols* of V. The place where an enzyme E_1 can cut a word can be described by a pair (u_1, u_2), of words over V; such a pair is called a *context*. Assume that we have one more enzyme, E_2, characterized by the context (u_3, u_4). This means that when a word contains the subword u_1u_2 (or u_3u_4), then E_1 (resp. E_2) can cut that word between u_1 and u_2 (resp. between u_3 and u_4). If the two enzymes produce matching ends of the cut words, then we indicate this by putting the two contexts together, writing $((u_1, u_2), (u_3, u_4))$. This means that a ligation can occur.

Let us be a little bit more specific and assume that the words x, y above can be cut by the enzymes E_1, E_2, i.e., $x = x_1u_1u_2x_2$ and $y = y_1u_3u_4y_2$. The resulting fragments are

$$x_1u_1, \quad u_2x_2, \quad y_1u_3, \quad u_4y_2.$$

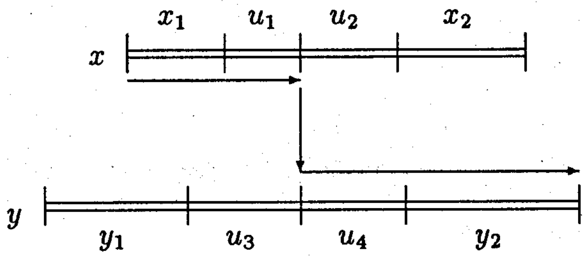

Fig. 1. The splicing operation.

If immediate recombination (ligation) is possible, then we can get any of the following four words:

$$x_1u_1u_2x_2,\quad x_1u_2u_4y_2,\quad y_1u_3u_2x_2,\quad y_1u_3u_4y_2.$$

The first and fourth words are identical with x and y again, but the second and third are possibly new words. We keep them as the result of the operation and we call this operation *splicing*. Specifically, we say that we have spliced x and y at the *sites* u_1u_2, u_3u_4, respectively, according to the *splicing rule* $((u_1, u_2), (u_3, u_4))$, obtaining $x_1u_2u_4y_2$, $y_1u_3u_2x_2$; the words x, y are also called the *terms* of the splicing.

Figure 1 illustrates this operation.

When passing from the natural DNA recombination to the formal splicing operation on words consisting of abstract symbols we, in fact, leave both DNA reality and terminology, generalizing the operation from several points of view:

1. We consider words, not double-stranded sequences (also having a spatial structure).
2. We work on arbitrary alphabets, not on the four-letter DNA alphabet.
3. The sites where the splicing is performed are identified by pairs of words on which we place no length restriction.
4. We cut the words only once and as a result of the splicing we keep the two possibly new words.
5. We ignore circular words.

However, these assumptions are not dramatically losing reality:

1. Due to the precise Watson–Crick complementarity, a single strand of a DNA sequence uniquely identifies the double-stranded sequence, hence there is no loss of information in point 1 above.
2. It is known that the symbols of any alphabet can be codified using two given symbols in such a way that many properties are preserved (unique deciphering, the place in the Chomsky hierarchy, etc.). In the DNA case we have four letters at our disposal, hence enough for such an encoding.
3. The actual restriction enzymes have recognizing patterns of small length, in most cases less than eight. From the proofs below, one can see that the longest context (u, u') used in splicing rules is of total length seven, and most of the contexts are of length two, three, or four. Again it seems that the generalization in the

definition is harmless if we return to DNA with the specific constructions used in the proofs.

4. If an enzyme can cut a DNA sequence in several places, then the action can take place nondeterministically in none, one, or in several places. However, we are interested here in finding *language-generating* devices which start from given words (axioms) and iteratively using given splicing rules produce all the words of languages we are interested in. Thus, on one hand, several cuttings of the same word are just iterations of a single-cutting operation; on the other hand, we look for *possible* derivations of the words of the language, thus discarding all the "wrong computations." This is the usual style of formal language theory. Moreover, in the DNA framework, due to the huge parallelism and nondeterminism of all processes taking place, we may assume that any *possible* operation actually takes place; furthermore, by specific filtering procedures, we can squeeze out the desired result from the output of a reaction, discarding the "garbage."
5. We may ignore circular words, on the one hand, because we may assume that linear words exist and can be selected, on the other hand, because the results we obtain can be generalized to languages containing both linear and circular words in a straightforward manner.

Besides these generalizations, in the constructions below we use several assumptions of a theoretical nature, which are no longer looking so "innocent"; we discuss them when introducing them.

On the positive side, as our models are so abstract, maybe they can be implemented not only on DNA supports, but also on other (bio)chemical structures; such a possibility has already been suggested in Adleman (1995).

1.4. *Splicing Systems; Looking for Universality*

The splicing operation can be used as a basic tool for building a language-generating mechanism.

The general framework we step in is formal language theory, whose main goal is to characterize the correctness (well-formedness) of the *phrases* of a *language* over a given *vocabulary*. As usual, we identify "vocabulary" with "alphabet" (a set of abstract *symbols*, which might denote any linguistic or nonlinguistic reality) and "phrases" with "words" (sequences of the considered abstract symbols). Thus, a language is a set (finite or not) of words composed of symbols from a given alphabet.

In order to identify the elements of a language one usually considers *grammars*, finite devices able to *generate* the words of the language. Besides the *alphabet* on which we work, a grammar should contain a set (possibly a singleton) of *axioms*, which are words considered to be correct "by definition," and a set of *production rules*, some elementary actions/tools, by the use of which the words can be transformed, preserving the well-formedness. Starting from axioms and iteratively applying the productions, a grammar identifies a set of correct words (a language). We say that this is the language *generated* by the grammar.

Of course, of special interest is the case when the components of the grammar, especially the sets of axioms and of production rules, are finite.

The basic grammars investigated in formal language theory are the *Chomsky* *grammars*. They have the additional feature that besides the alphabet of the generated language, they use an additional alphabet, of *auxiliary* symbols, only used in the process of derivation. The work of the grammar is accepted only when the produced word contains no such auxiliary symbol. This is why the auxiliary symbols are also called *nonterminals* and those appearing in the generated language are called *terminals*.

The unrestricted Chomsky grammars (the so-called type-0 grammars) characterize the family of recursively enumerable languages, which is also known as the family of languages recognized by Turing Machines.

For short, Turing Machines are devices consisting of a *finite memory*, a *read–write head*, and a potentially infinite *tape*; in each moment, the read–write head scans a *cell* of the tape. According to the *state* of the memory, the read–write head can rewrite the scanned symbol, can move one cell to the left or to the right. A sequence of such elementary operations, with the machine starting in a special *initial state* and finishing the work in a *terminal state*, is called a *computation*. Thus, a language can be associated to a Turing Machine, consisting of all words x for which there is a computation where starting with the word x on the tape, the machine goes from the initial state to a terminal state.

A Turing Machine which, given a word x on its tape, can only go from the left to the right on its tape, without modifying the scanned symbols, is called a *finite automaton*; finite automata characterize the family of regular languages.

According to the Turing–Church thesis, Turing Machines are the most general level of algorithmicity/calculability: any conceivable type of algorithms can be simulated by a Turing Machine (hence by a type-0 Chomsky grammar). In this paper we take the type-0 Chomsky grammars as the standard model of computability.

From the opposite viewpoint, the most restricted class of Chomsky grammars are the so-called regular ones, which generate the family of regular languages (which are also characterized by finite automata). Finite automata and Turing Machines are, in some sense, the two poles of computability and they play an essential role in the subsequent sections.

We return to the splicing operation. Having such an operation, we can design a language-generating mechanism in the usual way: take an alphabet, take a set of words over this alphabet (axioms), and take a set of splicing rules. Starting from the axioms, by iterated splicings we get a set of words. The mechanism described is a *splicing system*, the set of the words produced is the *language* generated by it. Following the model of Chomsky grammars (and of other well-known mechanisms in formal language theory, such as the Lindenmayer systems, see, e.g., Rozenberg and Salomaa (1980), (1997)), we can also consider auxiliary symbols, which can be used during the derivation process, but are not allowed to appear in the words produced. In this way we obtain the notion of an *extended splicing system*; the basic model is then called *nonextended*. The extended splicing systems (also called *extended H systems*) are the main subject of this paper: How powerful are they? Can they reach the power of Turing Machines? Under which conditions?

The extended splicing systems were explicitly introduced in Păun et al. (1996), but variants of them, in general nonextended, were much investigated in the last years.

In general, the H systems, extended or not, turned out to be very powerful generative mechanisms. Several characterizations of recursively enumerable languages in terms of various types of H systems were obtained in Păun (1996a) and Păun et al. (1996). Related results appeared in Yokomori and Kobayaski (1995), while Denninghoff and Gatterdam (1989) encoded the range of Turing Machines using iterated splicing on multisets (sets with multiplicities associated to their elements).

We improve these results here by obtaining the strongest characterization of recursively enumerable languages possible in this framework, i.e., the family of recursively enumerable languages is generated by H systems using finite sets of axioms and finite sets of splicing rules, supplemented with a certain control mechanism. Such a control mechanism consists of keeping track of the multiplicities of words (the number of available copies of a word at each moment), respectively of checking the presence of certain symbols in the words to be spliced.

This is the strongest result that can be obtained since, as was shown in Culik and Harju (1991) and Pixton (1995), if no control is used, then the H systems using finite sets of axioms and finite sets of rules only generate the family of regular languages. Our results can be reformulated by saying that the computational power of H systems of the types mentioned (using multisets or controlled splicing) equals the computational power of Turing Machines (resp. of type-0 Chomsky grammars). This is rather unexpected, taking into account that the nature of the splicing operation is quite different from the nature of the operations involved in the work of a Turing Machine or of a Chomsky grammar.

In formal language theory there are many characterizations of recursively enumerable languages. Usually, a class of mechanisms can lead to such a characterization if these mechanisms have (1) *context sensing* capabilities and (2) *erasing* capabilities. By the definition of the splicing operation, we have both these features introduced from the very beginning into H systems. The results in Culik and Harju (1991) and Pixton (1995) prove that they are not sufficient. One explanation is the fact that context sensing acts only locally, we cannot transmit information from a part of a word to another one. In our proofs we enlarge the context sensing capabilities. Basically, we introduce some possibilities to sense contexts at distance. The modifications look weak, but they turn out to fill in the missing strength: extended H systems with long distance context sensing (under the form of certain regulations for using the splicing rules) are computationally complete, they characterize the recursively enumerable languages.

However, this does not immediately imply that such systems can be used as *computers*. If for every problem (identified here by a language) we have to build a specific computer (identified here by an H system), then we do not obtain very much. We need *universal H systems*, in the sense already introduced by Turing (1936).

A Turing Machine, TM_u, is universal if all its components are fixed and given a specific Turing Machine TM_0 we can run TM_0 as a program on TM_u, in the following sense. Assume that starting from a word w, the machine TM_0 produces a word w'. There is a word w_0, the *code* of TM_0, such that starting from w_0w the machine TM_u produces the word w', and this is true for all pairs (w, w'), having w_0 fixed. Thus, w_0 is the "program" of the algorithm described by TM_0; receiving it, as well as some input *data* w, TM_u behaves exactly as TM_0 when working on the same data. Similarity with the actual computers is apparent. Turing (1936) already proved that there are universal Turing Machines. The notion of universality can be extended to any type of language generating (or computing) devices, in particular to type-0 Chomsky grammars (but, of course, not for all classes of mechanisms are there universal mechanisms inside the class).

For H systems it is both surprising and promising from a "practical" point of view to show that universal H systems exist. As suggested above, by a universal H system we mean a system with fixed sets of auxiliary symbols, of axioms, and of splicing rules that can behave like any given H system γ if we add an *encoding* of γ as an additional axiom to the universal system. We prove the existence of such a universal H system with multiplicities (resp. with checking the occurrence of certain subwords).

The interpretation of these results, from the point of view of genetic/molecular computing is that, theoretically, there exist *programmable universal DNA computers* which are based on the splicing operation. The only operations used in these "computers" are the iterated splicing (the splicing known from DNA recombination) and the squeezing operation of selecting only the produced words which contain no auxiliary symbol (in formal language terms, this amounts to the intersection with a regular set of the form T^*, with T a given alphabet; from biochemical point of view, this is a filtering procedure of a kind already used in laboratory—see, for instance, Adleman (1994) and Hunter (1993)). Therefore, for the case of using the splicing operation as a basic operation on DNA sequences, we (affirmatively) answer the problems formulated in Adleman (1994) and Gifford (1994).

In an optimistic way, one can think of an analogy between these results and the existence of universal Turing Machines proved in Turing (1936), which has laid the theoretical foundation for the design of electronic computers. However, we stress the *theoretical* character of our results. We only prove that DNA computers, programmable and universal, are *mathematically* possible. Will they become "personal computers" in, say, 50 years, as happened with electronic computers? Maybe yes, maybe no. Anyway, here we do not discuss the feasibility of actually building such computers, as this problem goes far beyond mathematics, stepping into practical DNA engineering.

2. Extended H Systems

In this section we formally introduce the notion which is fundamental to our study, i.e., that of an extended H system. First, we fix some formal language theory notations and terminology, then we define the extended H systems, we exemplify the definition, and we recall some previous results about the power of H systems.

2.1. *Formal Language Theory Prerequisites*

An alphabet is a finite and nonempty set of abstract symbols. Having an alphabet V, by V^* we denote the set of all words over V, including the empty one denoted by λ. The set of nonempty words over V, i.e., $V^* - \{\lambda\}$, is denoted by V^+. The length of a word $x \in V^*$ (the total number of symbol occurrences in x) is denoted by $|x|$.

A Chomsky grammar is a quadruple $G = (N, T, S, P)$, where N and T are disjoint alphabets, $S \in N$, and P is a finite set of pairs $(u, v), u, v \in (N \cup T)^*$, such that u contains at least one symbol from N.

The elements of N are called nonterminal symbols (they are used in the process of generating the language associated to G but are not allowed to appear in the words of this language), those of T are called terminal symbols (the generated language consists of words over this alphabet), S is the axiom, and the pairs (u, v) of P are called production rules (or simply productions, or rules) and are written in the form $u \rightarrow v$. This suggests the way they are used: u can be replaced (one also says rewritten) by v. Formally, for $x, y \in (N \cup T)^*$ we write

$$x \Rightarrow y \quad \text{if and only if} \quad x = x_1 u x_2, \quad y = x_1 v x_2, \quad \text{for some} \quad u \rightarrow v \in P.$$

We say that x directly derives y according to the rule $u \rightarrow v$ in P. By $\Rightarrow^*$ we denote the reflexive and transitive closure of the relation $\Rightarrow$ (thus, $\Rightarrow^*$ indicates zero or more direct derivation steps). Then the language generated by G is

$$L(G) = \{w \in T^* \mid S \Rightarrow^* w\}.$$

In what follows we do not consider the empty word, because it has no biological representation, and, moreover, the definitions can be given in a shorter way. Thus, we consider two languages identical if they differ by at most the empty word: $L_1 = L_2$ if $L_1 - \{\lambda\} = L_2 - \{\lambda\}$.

The family of languages generated by arbitrary Chomsky grammars is the family of recursively enumerable languages; we denote it by RE.

A grammar as above, with no restriction on the form of its rules, is said to be of type 0. If all rules are of the form $u \rightarrow v$ with $|u| \leq |v|$, then G is called monotone (of type 1, or context-sensitive). The grammar G is called context-free (of type 2) if all rules are of the form $A \rightarrow x$ for some $A \in N$. If all rules in a Chomsky grammar are of the forms

$$X \rightarrow aY, \quad X \rightarrow a, \qquad \text{for} \quad X, Y \in N, \quad a \in T,$$

then the grammar is called regular (or of type 3).

By *REG*, *CF*, *CS* we denote the families of regular, context-free, and context-sensitive languages, respectively, which are generated by the corresponding types of Chomsky grammars. By *FIN* we denote the family of finite languages. The following inclusions

$$FIN \subset REG \subset CF \subset CS \subset RE$$

are known as the Chomsky hierarchy and this is the usual reference for estimating the generative power of any newly introduced generative mechanisms.

As we have mentioned in Section 1.4, *RE* is also characterized by Turing machines and *REG* is also characterized by finite automata.

For general formal language theory prerequisites we refer to Harrison (1978), Salomaa (1973), and Rozenberg and Salomaa (1997). A friendly introduction to formal language theory for biologists can be also found in Searls (1993).

We only mention here the so-called *Kuroda normal form*, which is used below several times.

Lemma 1. *For each type-0 Chomsky grammar G there is a grammar $G' = (N, T, S, P)$ such that $L(G) = L(G')$ and P contains only rules of the following forms:*

1. *$A \rightarrow a$, for $A \in N, a \in T$,*

2. *$A \to BC$, for $A, B, C \in N$,*
3. *$A \to \lambda$, for $A \in N$,*
4. *$AB \to CD$, for $A, B, C, D \in N$.*

2.2. *Extended H Systems and Their Languages*

As we want to work in a general framework, making possible, for instance, the use of arbitrary sets of splicing rules, we encode the splicing rules not as quadruples $((u_1, u_2), (u_3, u_4))$ as above, but as strings, using the new symbols #, \$ as separators: instead of $((u_1, u_2), (u_3, u_4))$ we write $u_1\#u_2\$u_3\#u_4$.

Definition 1. An *extended H system* is a quadruple

$$\gamma = (V, T, A, R),$$

where V is an alphabet, $T \subseteq V$, $A \subseteq V^+$, and $R \subseteq V^*\#V^*\$V^*\#V^*$; #, \$ are special symbols not in V.

V is the *alphabet* of γ, T is the *terminal* alphabet, A is the set of *axioms*, and R is the set of *splicing rules*; the symbols in T are called *terminals* and those in $V - T$ are called *nonterminals*.

For $x, y, z, w \in V^+$ and $r = u_1\#u_2\$u_3\#u_4$ in R we define

$$(x, y) \vdash_r (z, w) \quad \text{if and only if} \quad \begin{array}{lll} x = x_1u_1u_2x_2, & y = y_1u_3u_4y_2, & \text{and} \\ z = x_1u_1u_4y_2, & w = y_1u_3u_2x_2, & \\ \text{for some} & x_1, x_2, y_1, y_2 \in V^*. & \end{array}$$

The words x, y are called the *terms* of the splicing. When r is understood, we write $\vdash$ instead of $\vdash_r$. Note that in splicing rules $u_1\#u_2\$u_3\#u_4$ we can have empty words u_1, u_2, u_3, u_4; for instance, a rule $a\#b\$c\#$ (with $u_4 = \lambda$) says that the first term of the splicing can be cut between a and b and the second one can be cut after c without any restriction concerning the symbols which follow. Of course, we never use splicing rules with both u_1, u_2 or both u_3, u_4 empty.

Definition 2. For an H system $\gamma = (V, T, A, R)$ and for any language $L \subseteq V^+$, we write

$$\sigma(L) = \{z \in V^+ \mid (x, y) \vdash_r (z, w) \text{ or } (x, y) \vdash_r (w, z), \text{ for some } x, y \in L,\ r \in R\},$$

and we define

$$\sigma^*(L) = \bigcup_{i \geq 0} \sigma^i(L),$$

where

$$\sigma^0(L) = L,$$
$$\sigma^{i+1}(L) = \sigma^i(L) \cup \sigma(\sigma^i(L)), \qquad \text{for} \quad i \geq 0.$$

The *language generated* by the H system γ is defined by

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

Then, for two families of languages, F_1, F_2, we denote

$$EH(F_1, F_2) = \{L(\gamma) \mid \gamma = (V, T, A, R), A \in F_1, R \in F_2\}.$$

Thus, $\sigma(L)$ is the set of strings obtained by splicing words in L according to splicing rules in R. Iterating the splicing, we get $\sigma^*(L)$, which can also be defined as the closure of L under splicing with respect to rules in R, or, equivalently, as the smallest language containing L and stable under splicing with respect to rules in R.

The language $L(\gamma)$ consists of all words which can be obtained from the axioms in A (including these axioms), by using arbitrarily many splicing operations, provided that no symbol outside T is still present in the word produced.

An H system $\gamma = (V, T, A, R)$ with $A \in F_1, R \in F_2$, for F_1, F_2 being two given families of languages, is said to be of type (F_1, F_2). (Note that R is a language, hence the definition makes sense.)

In the notion $EH(F_1, F_2)$, EH stands for "extended H" systems. A system as above having $V = T$ is called nonextended (all words are terminal, hence $L(\gamma) = \sigma^*(A)$). We do not discuss nonextended H systems in this paper, but incidentally they appear in certain proofs, hence we only specify that by $H(F_1, F_2)$ we denote the family of languages generated by nonextended H systems of type (F_1, F_2).

Remark 1. It is worth noting that the intersection with T^+, which is essential for defining the language generated by an extended H system, can already be realized practically. This is exactly the way the description of the Hamiltonian path is selected in Adleman (1994). (In fact, the practical procedure consists of removing from the test tube all the sequences containing a given subword; after the removing of all sequences containing an auxiliary symbol, codified in a specific way, what remains in the test tube is exactly the set of words over T, hence the desired intersection with T^+.)

Remark 2. From a mathematical point of view, for a splicing rule $r = u_1\#u_2\$u_3\#u_4$ it is enough to define the ternary relation $(x, y) \vdash_r z$, for $x = x_1u_1u_2x_2$, $y = y_1u_3u_4y_2$, and $z = x_1u_1u_4y_2$. Considering also the word $w = y_1u_3u_2x_2$ amounts to considering the *symmetric rule* $u_3\#u_4\$u_1\#u_2$, too. This is the style of many papers on splicing. Here we work in the restricted set-up of Definition 1 for "practical" reasons, in order not to introduce unnecessary assumptions which cannot be met in practice. On the other hand, a *reflexivity assumption* should also be considered: together with $r = u_1\#u_2\$u_3\#u_4$, also the rules $u_1\#u_2\$u_1\#u_2$ and $u_3\#u_4\$u_3\#u_4$ are implicitly present (the restriction enzyme defining the cut at $u_1\#u_2$ acts on this site irrespective of whether or not some other word containing the site $u_3\#u_4$ is present; moreover, there might also be a ligase recombining the two ends u_1 and u_2). We do not consider these additional rules in our constructions, but in remarks placed after the main proofs we check whether they would modify the generated languages.

u_1	u_2
u_3	u_4

Fig. 2. Tabular view of a splicing rule.

2.3. *Examples*

For the reader not accustomed with the splicing operation it might be difficult to "feel" the way an H system works. Hence it might be of some help to see a splicing rule $u_1\#u_2\$u_3\#u_4$ as in Figure 2, which remembers the way the splicing operation proceeds (see also Figure 1).

In order to increase readability, the place where the cutting of the splicing terms is done is specified below by a vertical bar:

$$(x_1u_1|u_2x_2,\ y_1u_3|u_4y_2) \vdash (x_1u_1u_4y_2,\ y_1u_3u_2x_2).$$

Figure 3 illustrates the way of building the output words using parts of the input words.

Bearing in mind these representations, we examine some **examples**.

The first one is very simple: consider the extended H system

$$\gamma_1 = (\{a, b, c\}, \{a\}, \{a^nb, ca^m\}, \{a\#b\$c\#a\}),$$

where n, m are two given positive integers. The only splicing rule, $a\#b\$c\#a$, can be applied to the two existing axioms, a^nb, ca^m, and we get

$$(a^n|b,\ c|a^m) \vdash (a^{n+m},\ cb).$$

None of the words obtained can enter new splicings. Therefore, for $A = \{a^nb, ca^m\}$, we have

$$\sigma^0(A) = \{a^nb, ca^m\},$$
$$\sigma^1(A) = \sigma^0(A) \cup \{a^{n+m}, cb\},$$
$$\sigma^i(A) = \sigma^1(A), \qquad i \geq 2,$$

therefore

$$\sigma^*(A) = \sigma^0(A) \cup \sigma^1(A) = \{a^nb, ca^m, a^{n+m}, cb\}$$

and

$$L(\gamma_1) = \{a^{n+m}\}.$$

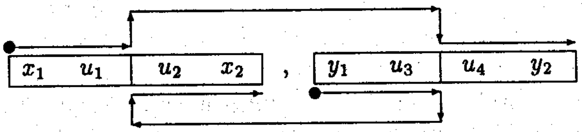

Fig. 3. Building the output of a splicing operation.

The extended H system γ_1 in some sense "computes" the addition of the integers n and m.

Note that we can consider a simpler splicing rule,

$$\gamma_1' = (\{a, b, c\}, \{a\}, \{a^n b, ca^m\}, \{\#b\$c\#\}),$$

and the generated language is the same:

$$(a^n|b, c|a^m) \vdash (a^{n+m},\ cb),$$
$$(a^n|b, c|b) \vdash (a^n b,\ cb),$$
$$(c|b, c|a^m) \vdash (ca^m,\ cb).$$

Thus, the new splicings where the word cb is involved produce no new word, hence

$$\sigma^*(A) = \sigma^0(A) \cup \sigma^1(A) \cup \sigma^2(A) = \{a^n b, ca^m, a^{n+m}, cb\},$$

hence $L(\gamma_1) = L(\gamma_1') = \{a^{n+m}\}$.

Now consider a more complex H system (it is, in fact, a reformulation of an example in Mateescu et al. (1995)); take

$$\gamma_2 = (\{a, b, c\}, \{a, b, c\}, \{abaca, acaba\}, R),$$

with R containing the two splicing rules

$$r_b = b\#\$b\#, \qquad r_c = c\#\$c\#.$$

For instance, we can perform

$$(abac|a, ac|aba) \vdash_{r_c} (abacaba, aca),$$
$$(abacab|a, ab|aca) \vdash_{r_b} (abacabaca, aba),$$

or, in general, for $n \geq 1$,

$$((abac)^n|a, ac|aba) \vdash_{r_c} ((abac)^n aba, aca),$$
$$((abac)^n ab|a, ab|aca) \vdash_{r_b} ((abac)^{n+1} a, aba).$$

In a similar way,

$$((acab)^n|a, ab|aca) \vdash_{r_b} ((acab)^n aca, aba),$$
$$((acab)^n ac|a, ac|aba) \vdash_{r_c} ((acab)^{n+1} a, aca).$$

Therefore,

$$\{abac\}^+\{a\} \cup \{abac\}^*\{aba\} \cup \{acab\}^+\{a\} \cup \{acab\}^*\{aca\} \subseteq L(\gamma_2).$$

(The extension plays no role in this case, γ_2 is of the form (T, T, A, R), hence it is a nonextended system.)

The reader can also verify that the converse inclusion holds (hint: when splicing two words of one of the forms $(abac)^n a$, $(acab)^n a$—initially we have $n = 1$—or of the forms $(abac)^n aba$, $(acab)^n aca$, we identify either a subword aba or a subword aca of them, hence the words obtained are of the same forms).

Now we consider a still more complex example, namely, an extended H system of type (FIN, REG) generating the language $\{a^{2^n} \mid n \geq 0\}$ (one can say that this H system computes the power of 2). Besides the aim of familiarizing the reader with the way the splicing operation and the H systems work, this example also involves an idea used in the constructions in Section 4.

Consider the system

$$\gamma_3 = (V, \{a\}, A, R),$$

with

$$V = \{X, X', Y, Y', Y'', B, a\},$$
$$A = \{XBaY,\ X'aY',\ X'Y''\},$$

and R containing the following groups of rules:

1. $Xa^nBa^m\#aY\$X'a\#Y'$, for $n, m \geq 0$,
2. $X'a^2\#Y\$X\#a^nBa^mY'$, for $n, m \geq 0$,
3. $X'a^nBa^m\#Y'\$X\#Y$, for $n, m \geq 0$,
4. $X\#Y\$X'\#a^nBa^mY$, for $n, m \geq 0$,
5. $Xa^n\#BY\$X'\#Y''$, for $n \geq 1$,
6. $X'B\#Y\$X\#a^nY''$, for $n \geq 1$,
7. $X'Ba^n\#Y''\$X\#Y$, for $n \geq 1$,
8. $\#XY\$XB\#a^nY$, for $n \geq 1$,
9. $a^n\#Y\$XY\#$, for $n \geq 1$.

Here is the way of producing the word a^4:

$$(XB|aY, X'a|Y') \vdash_1 (XBY', X'a^2Y),$$
$$(X'a^2|Y, X|BY') \vdash_2 (X'a^2BY', XY),$$
$$(X'a^2B|Y', X|Y) \vdash_3 (X'a^2BY, XY'),$$
$$(X|Y, X'|a^2BY) \vdash_4 (Xa^2BY, X'Y),$$
$$(Xa^2|BY, X'|Y'') \vdash_5 (Xa^2Y'', X'BY),$$
$$(X'B|Y, X|a^2Y'') \vdash_6 (X'Ba^2Y'', XY),$$
$$(X'Ba^2|Y'', X|Y) \vdash_7 (X'Ba^2Y, XY''),$$
$$(X|Y, X'|Ba^2Y) \vdash_4 (XBa^2Y, X'Y),$$
$$(XBa|aY, X'a|Y') \vdash_1 (XBaY', X'a^2Y),$$
$$(X'a^2|Y, X|BaY') \vdash_2 (X'a^2BaY', XY),$$
$$(X'a^2Ba|Y', X|Y) \vdash_3 (X'a^2BaY, XY'),$$
$$(X|Y, X'|a^2BaY) \vdash_4 (Xa^2BaY, X'Y),$$
$$(Xa^2B|aY, X'a|Y') \vdash_1 (Xa^2BY', X'a^2Y),$$
$$(X'a^2|Y, X|a^2BY') \vdash_2 (X'a^4BY', XY),$$

$$
\begin{aligned}
&(X'a^4B|Y', X|Y) \vdash_3 (X'a^4BY, XY'),\\
&(X|Y, X'|a^4BY) \vdash_4 (Xa^4BY, X'Y),\\
&(Xa^4|BY, X'|Y'') \vdash_5 (Xa^4Y'', X'BY),\\
&(X'B|Y, X|a^4Y'') \vdash_6 (X'Ba^4Y'', XY),\\
&(X'Ba^4|Y'', X|Y) \vdash_7 (X'Ba^4Y, XY''),\\
&(X|Y, X'|Ba^4Y) \vdash_4 (XBa^4Y, X'Y),\\
&(|XY, XB|a^4Y) \vdash_8 (a^4Y, XBXY),\\
&(a^4|Y, XY|) \vdash_9 (a^4, XYY).
\end{aligned}
$$

One sees how rules in groups 1–4 cut a symbol a from the right-hand end of a word of the form $Xa^nBa^mY, n \geq 0, m \geq 1$, and add a^2 to the left-hand word a^n. The pairs of nonterminals (X, Y), (X, Y'), (X', Y), (X', Y') control these operations precisely. In a similar way, rules in groups 5–7 and 4 cut a symbol B from words of the form Xa^nBY and move it to the left-hand end, producing the word XBa^nY. The process can be iterated. Between a moving of B from the right-hand end to the left-hand end and another moving of B, all occurrences of a are doubled. The symbol B can be removed only in the presence of both X and Y, namely, from words of the form XBa^nY. This ensures the fact that n is of the form 2^i. Then Y can also be removed. (If Y is removed before removing B, then B can never be removed.) All rules in groups 1–7 can be used only in the presence of both X and Y, or of their primed variants.

Using these remarks, the reader can prove not only the rather easy inclusion $\{a^{2^n} \mid n \geq 0\} \subseteq L(\gamma_3)$, but also the converse.

2.4. *The Power of Extended H Systems*

For the readers' convenience and as a reference framework for our study here, we recall some known results on the generative power of extended H systems.

Specifically, the results in Table 1 hold, where at the intersection of the row marked with a family F_1 with the column marked with a family F_2 we have indicated the corresponding family $EH(F_1, F_2)$. The most important results in this table are the equalities:

1. *EH(FIN, FIN) = REG,*
2. *EH(FIN, REG) = RE.*

Using a finite set of splicing rules and a finite (or regular) set of axioms leads to regular languages only. The inclusion *EH(FIN, FIN)* $\subseteq$ *REG* was first proved in Culik and Harju (1991); Pixton (1996) gives a simpler proof, whereas Pixton (1995) and Head

Table 1. The size of families $EH(F_1, F_2)$.

	FIN	*REG*	*CF*	*CS*	*RE*
FIN	*REG*	*RE*	*RE*	*RE*	*RE*
REG	*REG*	*RE*	*RE*	*RE*	*RE*
CF	*CF*	*RE*	*RE*	*RE*	*RE*
CS	*RE*	*RE*	*RE*	*RE*	*RE*
RE	*RE*	*RE*	*RE*	*RE*	*RE*

et al. (1997) contain a general result in terms of abstract families of languages, which also implies the inclusion above. The opposite inclusion is proved in Păun et al. (1996). It is worth noting that using a finite set of splicing rules also preserves the context-freeness. (The inclusion $EH(CF, FIN) \subseteq CF$ is proved in Pixton (1995).)

However, if we make the step from finite sets of rules to regular sets, then we spectacularly jump to the full power of Turing Machines/type-0 Chomsky grammars. The inclusion $RE \subseteq EH(FIN, REG)$ is proved in Păun (1996b), the opposite one follows from the Turing–Church thesis.

The proof in Păun (1996b) makes essential use of the fact that we are allowed to use an infinite set of splicing rules, a feature which makes the result of less interest from a practical point of view. We cannot handle an infinite set of splicing rules (hence with arbitrarily long splicing sites), even when this set is described by a regular language. However, according to the results mentioned in Culik and Harju (1991) and Pixton (1995), finite sets of rules can only cover the competence of finite automata (resp. of regular grammars). In order to get a larger power, we have to supplement the model with something else and this is suggested by the very proof in Păun (1996b): the splicing sites are arbitrarily long just because they check the presence of certain symbols at the ends of the spliced words. In the proofs in Section 4 we see how this is done, hence we do not go into detail here.

3. Computational Completeness Using Multisets

In this section we give one of the main results of the paper: if we are able to keep track of the number of copies of the words used, then extended H systems can characterize the recursively enumerable languages.

In fact, what we need is to be able to keep track of the descendants of one axiom only, which is initially present in one copy; after splicing it together with another axiom with arbitrary nonzero multiplicity, we get two words whose multiplicity is known, namely, equal to one; then these two distinguished words are spliced together, resulting in one word of multiplicity one. The process is iterated, hence in any moment we should be able to know exactly how many copies are present for one or two words only. All the other words are supposed to be present in an arbitrary number of copies, large enough for the needs of the splicing process.

The mathematical framework for such operations with sets whose elements appear in a specified number of copies is the theory of multisets. In the next subsection we fix some notation and terminology and we define the extended H systems with multiplicities (shortly, extended mH systems). In Section 3.2 we prove that extended mH systems characterize RE.

3.1. *Multisets and mH Systems*

In the definition above of the splicing operation, after splicing two words x and y and obtaining two words z and w, we may use x or y again as a term of a splicing, possibly the second one being z or w, they are not "consumed" by splicing; also the new words are supposed to appear in arbitrarily many copies. Probably more realistic is the assumption that at least part of the words are available in a limited number of copies. This leads us to consider *multisets*, i.e., sets with multiplicities associated to their elements.

In the style of Eilenberg (1974), a multiset over a set X is a function $M\colon X \to \mathbf{N} \cup \{\infty\}$; $M(x)$ is the number of copies of $x \in X$ in the multiset M. The set $\{w \in X \mid M(w) > 0\}$ is called the support of M and is denoted by $\mathrm{supp}(M)$. A usual set $S \subseteq X$ is interpreted as the multiset defined by $S(x) = 1$ for $x \in S$, and $S(x) = 0$ for $x \notin S$.

For two multisets M_1, M_2 over X we define their *union* by $(M_1 \cup M_2)(x) = M_1(x) + M_2(x)$, and their *difference* by $(M_1 - M_2)(x) = M_1(x) - M_2(x)$, $x \in X$, provided $M_1(x) \geq M_2(x)$ for all $x \in X$. Usually, a multiset with finite support, M, is presented as a set of pairs $(x, M(x))$, for $x \in supp(M)$.

For instance, $M_1 = \{(ab, 3), (abb, 1), (aa, \infty)\}$ is a multiset over $\{a, b\}^*$ with the support consisting of three words, ab, abb, aa; the first one appears in three copies, the second one appears in only one copy, whereas aa appears in arbitrarily many copies. If we also take $M_2 = \{(ab, 1), (abb, 1), (aa, 17)\}$, then the difference $M_1 - M_2$ is defined and is equal to $\{(ab, 2), (aa, \infty)\}$.

Definition 3. An *extended mH system* is a quadruple

$$\gamma = (V, T, A, R),$$

where V, T, R are as in an extended H system (Definition 1), and A is a multiset over V^+.

For such an mH system and two multisets M_1, M_2 over V^+ we define

$$\begin{array}{lll} M_1 \Rightarrow_\gamma M_2 & \text{iff} & \text{there are } x, y, z, w \in V^* \text{ such that} \\ & \text{(i)} & M_1(x) \geq 1,\ (M_1 - \{(x, 1)\})(y) \geq 1, \\ & \text{(ii)} & x = x_1u_1u_2x_2,\ y = y_1u_3u_4y_2, \\ & & z = x_1u_1u_4y_2,\ w = y_1u_3u_2x_2, \\ & & \text{for } x_1, x_2, y_1, y_2 \in V^*,\ u_1\#u_2\$u_3\#u_4 \in R, \\ & \text{(iii)} & M_2 = (((M_1 - \{(x, 1)\}) - \{(y, 1)\}) \cup \{(z, 1)\}) \cup \{(w, 1)\}. \end{array}$$

At point (iii) we have operations with multisets. The writing above is also meant to cover the case when $x = y$ (then we must have $M_1(x) \geq 2$ and we must subtract 2 from $M_1(x)$) or $z = w$ (then we must add 2 to $M_2(z)$).

The *language generated* by an extended mH system γ is

$$L(\gamma) = \{w \in T^+ \mid w \in supp(M) \text{ for some } M \text{ such that } A \Rightarrow_\gamma^* M\},$$

where $\Rightarrow_\gamma^*$ is the reflexive and transitive closure of $\Rightarrow_\gamma$.

For two families of languages, F_1, F_2, by $EH(mF_1, F_2)$ we denote the family of languages $L(\gamma)$ for $\gamma = (V, T, A, R)$ being an extended mH system with $supp(A) \in F_1$, $R \in F_2$.

In plain words, when passing from a multiset M_1 to a multiset M_2, according to γ, the multiplicity of two elements of M_1, x and y, is diminished by one, then the multiplicity of the resulting words, z and w, is augmented by one. The multiplicity of all the other elements in $supp(M_1)$ is not changed. The multiset obtained is M_2.

The language generated by γ consists of all words containing only terminal symbols and whose multiplicity is at least once greater than one during the work of γ (all the words which can be reached from the axioms by iterated splicings, using the "bounded resources" of the mH system).

An extended H system as in Definition 1 can be interpreted as an extended mH system working with multisets of the form $M(x) = \infty$ for all x such that $M(x) \neq 0$. Such multisets are called ω-multisets; hence in literature the corresponding H systems are also called ωH systems and the corresponding families are also denoted by $EH(\omega F_1, F_2)$. Observe that according to the definitions given above, for arbitrary F_1, F_2 we have $EH(\omega F_1, F_2) = EH(F_1, F_2)$.

An mH system $\gamma = (V, T, A, R)$ with $V = T$ is called *nonextended*; the family of languages generated by such systems, corresponding to $EH(mF_1, F_2)$, is denoted by $H(mF_1, F_2)$. The family of languages generated by mH systems $\gamma = (V, T, A, R)$ with $card(supp(A)) \leq k$, for a given k, and $R \in F_2$ is denoted by $EH(m[k], F_2)$. In a similar way, we denote the family of languages generated by extended H systems with at most k axioms by $EH([k], F_2)$. Again observe that $EH(\omega[k], F_2) = EH([k], F_2)$.

Remark 3. We want to emphasize an important point, whose misunderstanding can cause wrong interpretations of the model. According to the style of Eilenberg (1974), we write $M(x) = \infty$ for a word which is present in the multiset M in an *unbounded* number of copies. This however does not mean that we *actually* dispose of infinitely many copies of that word x. This only means that *if at any moment of the work of γ we need one further copy of x, then we can have it.* In the DNA framework, this means that if we need further copies of a given sequence, then we can produce them, for instance, devising amplifications via the so-called PCR (polymerase chain reaction) techniques, see, e.g., Hunter (1993) and Li and Graur (1991).

This remark essentially applies to the constructions in the following subsection. We shall return to this important point after presenting those constructions.

3.2. *Computational Completeness of mH Systems*

This subsection is devoted to the proof of the following result (stating the computational completeness of extended mH systems).

Theorem 1. *$REG = EH(m[1], FIN) \subset EH(m[2], FIN) = EH(mF_1, F_2) = RE$, for all families F_1, F_2 such that $FIN \subseteq F_1 \subseteq RE$, $FIN \subseteq F_2 \subseteq RE$.*

To stress the significance of this result, we reformulate (part of) it in different terms: every Turing Machine (resp. type-0 Chomaky grammar) can be simulated by an extended mH system with a finite set of rules; two axioms are sufficient (and necessary: systems with one axiom only generate regular languages).

We separate the proof of this theorem into four lemmas. The first one establishes the most important (and most unexpected) of the relations stated above.

Lemma 2. $RE \subseteq EH(mFIN), FIN)$.

Proof. Consider a type-0 Chomsky grammar $G = (N, T, S, P)$, with the rules in P of the form $u \to v$ with $1 \leq |u| \leq 2$, $0 \leq |v| \leq 2$, $u \neq v$ (for instance, we can take G in Kuroda normal form; Lemma 1 guarantees that such a grammar exists). Also assume that the rules in P are labeled in a one-to-one manner with elements of a set Lab; we write $r\colon u \to v$, for r being the label of $u \to v$. By U we denote the set $N \cup T$ and we construct the extended mH system

$$\gamma = (V, T, A, R),$$

where

$$V = N \cup T \cup \{X_1, X_2, Y, Z_1, Z_2\} \cup \{(r), [r] \mid r \in Lab\},$$

the multiset A contains the word

$$w_0 = X_1^2 Y S X_2^2,$$

with the multiplicity $A(w_0) = 1$, and the following words with infinite multiplicity:

$$\begin{aligned}
w_r &= (r)v\,[r], && \text{for} \quad r\colon u \to v \in P,\\
w_\alpha &= Z_1 \alpha Y Z_2, && \text{for} \quad \alpha \in U,\\
w'_\alpha &= Z_1 Y \alpha Z_2, && \text{for} \quad \alpha \in U,\\
w_t &= YY.
\end{aligned}$$

The set R contains the following splicing rules:

1. $\delta_1\delta_2 Yu\#\beta_1\beta_2\$(r)v\#[r]$, for $r\colon u \to v \in P$, $\beta_1, \beta_2 \in U \cup \{X_2\}$, $\delta_1, \delta_2 \in U \cup \{X_1\}$,
2. $Y\#u\,[r]\$(r)\#v\alpha$, for $r\colon u \to v \in P$, $\alpha \in U \cup \{X_2\}$,
3. $\delta_1\delta_2 Y\alpha\#\beta_1\beta_2\$Z_1\alpha Y\#Z_2$, for $\alpha \in U$, $\beta_1, \beta_2 \in U \cup \{X_2\}$, $\delta_1, \delta_2 \in U \cup \{X_1\}$,
4. $\delta\#Y\alpha Z_2\$Z_1\#\alpha Y\beta$, for $\alpha \in U$, $\delta \in U \cup \{X_1\}$, $\beta \in U \cup \{X_2\}$,
5. $\delta\alpha Y\#\beta_1\beta_2\beta_3\$Z_1 Y\alpha\#Z_2$, for $\alpha \in U$, $\beta_1 \in U$, $\beta_2, \beta_3 \in U \cup \{X_2\}$, $\delta \in U \cup \{X_1\}$,
6. $\delta\#\alpha Y Z_2\$Z_1\#Y\alpha\beta$, for $\alpha \in U$, $\delta \in U \cup \{X_1\}$, $\beta \in U \cup \{X_2\}$,
7. $\#YY\$X_1^2 Y\#w$, for $w \in \{X_2^2\} \cup T\{X_2^2\} \cup T^2\{X_2\} \cup T^3$,
8. $\#X_2^2\$Y^3\#$.

The idea behind this construction is the following. The rules in groups 1 and 2 simulate rules in P, but only in the presence of the symbol Y. The rules in groups 3 and 4 move the symbol Y to the right, the rules in groups 5 and 6 move the symbol Y to the left. The "main axiom" is w_0. All rules in groups 1–6 involve a word derived from

w_0 and contain such a symbol Y introduced by this axiom, in the sense that they can use only one axiom different from w_0. In any moment, we have two occurrences of X_1 at the beginning of a word and two occurrences of X_2 at the end of a word (maybe the same word). The rules in groups 1, 3, and 5 separate words of the form $X_1^2 z X_2^2$ into two words $X_1^2 z_1$ and $z_2 X_2^2$, each one with multiplicity one; the rules in groups 2, 4, and 6 bring together these words, leading to a word of the form $X_1^2 z' X_2^2$. The rules in groups 7 and 8 remove the auxiliary symbols X_1, X_2, Y. If the remaining word is terminal, then it is an element of $L(G)$. The symbols (r), $[r]$ are associated with labels in *Lab*, Z_1 and Z_2 are associated with *moving* operations.

Using these explanations, the reader can easily verify that each derivation in G can be simulated in γ, hence we have $L(G) \subseteq L(\gamma)$. (An induction argument on the length of the derivation can be used, but the details are straightforward and tedious; we avoid such a strategy here. Moreover, the discussion below implicitly shows how to simulate a terminal derivation in G by splicing operations in γ.)

We consider in some detail the opposite inclusion. We claim that if $A \Rightarrow_\gamma^* M$ and $w \in T^+$, $M(w) > 0$, then $w \in L(G)$.

As we have pointed out above, by a direct check we can see that we cannot splice two of the axioms $w_r, w_\alpha, w'_\alpha, w_t$ (for instance, the symbols δ, β in rules in groups 4 and 6 prevent the splicing of $w_\alpha, w'_\alpha, \alpha \in U$). In the first step, we have to start with w_0, $w_0 = X_1^2 Y S X_2^2$, $A(w_0) = 1$. Now, assume that we have a word $X_1^2 w_1 Y w_2 X_2^2$ with multiplicity one (w_0 is of this form). If w_2 starts with the left-hand member of a rule in P, then we can apply a rule of type 1 to it. Assume that this is the case, the word is $X_1^2 w_1 Y u w_3 X_2^2$ for some $r\colon u \to v \in P$. Using the axiom $(r)v[r]$ from A we obtain

$$(X_1^2 w_1 Y u | w_3 X_2^2,\ (r)v|[r]) \vdash (X_1^2 w_1 Y u[r],\ (r)v w_3 X_2^2).$$

No rule from groups 1 and 3–8 can be applied to the words obtained. From group 2, the rule $Y\#u[r]\$(r)\#v\alpha$ can be applied, involving both these words, which leads to

$$(X_1^2 w_1 Y | u[r],\ (r)| v w_3 X_2^2) \vdash (X_1^2 w_1 Y v w_3 X_2^2,\ (r)u\,[r]),$$

where the word $(r)u\,[r]$ can never enter a new splicing, because in the rule $r\colon u \to v$ from P we have assumed $u \neq v$. The multiplicity of $X_1^2 w_1 Y u[r]$ and $(r)v w_3 X_1^2$ has been reduced to zero again (hence these words are no longer available), the multiplicity of $X_1^2 w_1 Y v w_3 X_2^2$ is one. In this way, we have passed from $X_1^2 w_1 Y u w_3 X_2^2$ to $X_1^2 w_1 Y v w_3 X_2^2$, both having the multiplicity one, which corresponds to using the rule $r\colon u \to v$ in P. Moreover, we see that at each moment there is only one word containing X_1^2 and only one word (maybe the same) containing X_2^2 in the current multiset.

If to a word $X_1^2 w_1 Y \alpha w_3 X_2^2$ we apply a rule of type 3, then we get

$$(X_1^2 w_1 Y \alpha | w_3 X_2^2,\ Z_1 \alpha Y | Z_2) \vdash (X_1^2 w_1 Y \alpha Z_2,\ Z_1 \alpha Y w_3 X_2^2).$$

No rule from groups 1–3 and 5–8 can be applied to the words obtained. By using a rule from group 4 we obtain

$$(X_1^2 w_1 | Y \alpha Z_2,\ Z_1 | \alpha Y w_3 X_2^2) \vdash (X_1^2 w_1 \alpha Y w_3 X_2^2,\ Z_1 Y \alpha Z_2).$$

The first of the words obtained has replaced $X_1^2 w_1 Y \alpha w_3 X_2^2$, which now has multiplicity zero (hence we have interchanged Y with α), the second one is an axiom.

In the same way, one can see that using a rule from group 5 must be followed by using the corresponding rule of type 6, which results in interchanging Y with its left-hand neighbour.

Consequently, in each moment we have a multiset with either one word $X_1^2 w_1 Y w_2 X_2^2$ or two words $X_1^2 z_1$, $z_2 X_2^2$, each one with multiplicity one. Only in the first case, provided $w_1 = \lambda$, can we remove $X_1^2 Y$ by using a rule from group 7; then we can also remove X_2^2 by using the rule in group 8. This is the only way to remove these nonterminal symbols. If the word obtained is not terminal, then it cannot be processed any more, because it does not contain the symbol Y. In conclusion, we can only simulate derivations in G and move Y freely in the word of multiplicity one, hence $L(\gamma) \subseteq L(G)$. □

Remark 4. The proof above also remains valid when the reflexivity assumption in Remark 2 is considered, which can be seen as follows: Rules of type $u_1\#u_2\$u_1\#u_2$ for rules from groups 1–6 and 8, and rules of type $u_3\#u_4\$u_3\#u_4$ for rules from groups 2, 4, and 6–8 involve words whose multiplicities are at most one, hence these rules cannot be applied. The remaining rules of these types are applied to axioms of infinite multiplicity and what we obtain are always new copies of the axioms we used, because there is a unique position where we can splice.

Remark 5. Let us estimate the number of copies necessary for each axiom. We have said that $A(w_0) = 1$ (and this is essential for the correctness of the simulation of G by γ above). For all w of type w_r, w_α, w'_α, w_t we have said that $A(w) = \infty$. Actually, one sees that for each $r \in P$ we need as many copies of w_r for as many times rule r is used in a derivation in G. Then w_α and w'_α are necessary for each operation of moving Y to the left or to the right. The word w_t is used only once, by a rule of type 7, at the end of the work of γ. Thus, we might take $A(w_t) = 1$, too.

Moreover, we have seen above that in each moment there are exactly one or exactly two words whose multiplicity is controlled, namely equal to one. Thus, we do not have to "count," say, distinguishing between n and $n + 1$ copies of a given word, for large n. It is enough to distinguish between zero and one, and that for at most two words; this distinction is made automatically, by the way the system above works, our only care is to prevent making copies of these distinguished words.

This fact, plus the possibility of obtaining new copies of certain words, via PCR techniques, makes the construction above realistic—from these points of view.

What is nonrealistic, maybe fatal for the practical feasibility of the model, are the following two observations:

1. Having two words, each one with multiplicity one, and splicing them, is practically an event of probability equal to zero. Theoretically, the two words can arrive close enough to each other in order to let a ligase paste them. How to ensure this in a realistic interval of time (not to speak about an *efficient* time) is another story. One can imagine ways of binding the two words to a solid support (such techniques are relatively well understood, see, e.g., Zhang and Seeman

(1992)), in order to keep them closer and to increase the probability of splicing, but the extent to which such an operation is feasible and efficient is a matter of bioengineering beyond the scope of this paper.

2. The construction above involves a large number of splicing rules: of type 1 only we have used

   $card(P)(card(N \cup T) + 1)^4$

   rules! A splicing rule means two restriction enzymes. (In fact, one enzime for each rule of type 1 is sufficient: each splicing by a rule of type 1 uses an axiom $(r)v[r]$ and we can produce the fragments $(r)v$, $[r]$ of these axioms in advance, with sticky ends prepared as requested by the corresponding splicing rules. This decreases by a factor of two the number of enzymes necessary during the work of γ.) In general, each enzyme acts in specific temperature, salt concentration, and other reaction conditions, hence it is rather difficult to use different enzymes at the same time. In the work of γ above we never use the rules in parallel, hence we do not actually need to use concomitantly more than two enzymes, those corresponding to a rule. (Moreover, we may assume that we first cut a word by one enzyme, then the other word by the second enzyme, and after that we recombine the words obtained.) Thus, we may suppose that the reaction conditions are changed from a step to another one as requested by the enzyme currently used. This again makes the process theoretically and practically *possible*, but completely uninteresting as far as the efficiency is concerned: we cannot wait for the result of the computing if we have to change the temperature of the test tube for each splicing rule! (This reminds us of the way of computing in the framework of the "programming language" imagined by Lipton (1995) and Adleman (1995), by handling test tubes.)

Concerning the feasibility of the model, please also note that the longest splicing site we used is of length seven, in rules of group 1 ($|\delta_1\delta_2 Y u \beta_1 \beta_2| = 7$ when $|u| = 2$). In rules of groups 3, 5, and 7 we use sites of length six, each rule of groups 2, 4, and 6 has a site of length four, whereas the rule of type 8 has the longest site of length three.

Lemma 3. *EH(mFIN, FIN)* $\subseteq$ *EH(m*[2]*, FIN)*.

Proof. Take an extended mH system $\gamma = (V, T, A, R)$, with finite $supp(A)$. Let $w_1, w_2, \ldots, w_n$ be the words of $supp(A)$ such that $A(w_i) < \infty, 0 \leq i \leq n$, and let $z_1, \ldots, z_m$ be the words in $supp(A)$ with $A(z_i) = \infty, 0 \leq i \leq m$. We construct the extended mH system

$$\gamma' = (V \cup \{c, d_1, d_2\}, T, A', R'),$$

where A' contains the word

$$w = (w_1 c)^{A(w_1)} (w_2 c)^{A(w_2)} \cdots (w_n c)^{A(w_n)},$$

with multiplicity one, and the word

$$z = d_1cz_1cz_2c \cdots cz_mcd_2,$$

with infinite multiplicity. If $n = 0$, then w does not appear, if $m = 0$, then $z = d_1cd_2$. Moreover,

$$R' = R \cup \{\#c\$d_2\#,\ \#d_1\$c\#\}.$$

The word z can be used for cutting each w_i and each z_j from w and z, respectively. For instance, in order to obtain z_j we splice z with z using $\#d_1\$c\#$ for the occurrence of c to the left hand of z_j, that is,

$$(d_1cz_1c \cdots z_{j-1}c|z_jc \cdots cz_mcd_2,\ |z) \vdash (d_1cz_1c \cdots cz_{j-1}cz,\ z_jc \cdots cz_mcd_2),$$

then we splice the second word with z again using $\#c\$d_2\#$, and we get

$$(z_j|c \ldots cz_mcd_2,\ z|) \vdash (z_j,\ zcz_{j+1} \ldots cz_mcd_2).$$

Arbitrarily many words z_j can be produced, because $A'(z) = \infty$.

In order to produce the words w_i, $1 \leq i \leq n$, we start from the left-hand end of w, applying $\#c\$d_2\#$ to w and z; we get w_1 and $zc(w_1c)^{A(w_1)-1}(w_2c)^{A(w_2)} \cdots (w_nc)^{A(w_n)}$, both with multiplicity one. Using the rule $\#d_1\$c\#$ for z and the second word, we obtain zcz and $(w_1c)^{A(w_1)-1}(w_2c)^{A(w_2)} \cdots (w_nc)^{A(w_n)}$, again both with multiplicity one. From the first word we can separate axioms z_j, $1 \leq j \leq m$, but this is not important, because these axioms appear with infinite multiplicity in A. From the second word we can continue as above, cutting again a prefix w_1. In this way, exactly $A(w_1)$ copies of w_1 will be produced; we can proceed in a similar way for the other axioms $w_2, \ldots, w_n$ in order to obtain exactly the $A(w_i)$ copies of w_i, $i = 2, \ldots, n$.

The use of the nonterminals c, d_1, and d_2 guarantees that only the axioms of γ with multiplicity ∞ can be generated in an arbitrary number of copies by the splicing rules in $R' - R$, whereas for each axiom w_i of γ with finite multiplicity $A(w_i)$ we can only obtain $A(w_i)$ copies of w_i. If a rule of R is used for splicing words of the form x_1cx_2, i.e., containing the nonterminal c, we will finally have to cut such a word by using the rules in $R' - R$ in order to obtain a terminal word. As we start from the axioms of γ, separated by c, and with the correct multiplicities (guaranteed by the mode of constructing the words w and z), this also corresponds to a correct splicing in γ. Consequently, $L(\gamma') = L(\gamma)$. □

Remark 6. If the reflexivity assumption does not change the language generated by γ, then this is also true for γ', because the additional rules imposed by reflexivity are

$$\#c\$\#c,\quad c\#\$c\#,\quad d_2\#\$d_2\#,\quad \#d_1\$\#d_1.$$

The last two rules can be used only on unique places, hence they reproduce the terms of the splicing. The first two rules can be used for splicing parts of w, parts of z, or parts of

w with parts of z, but this is always done without changing the subwords w_i, $1 \leq i \leq n$, and z_j, $1 \leq j \leq m$, and without modifying the number of copies of each w_i, $1 \leq i \leq n$.

Lemma 4. *$EH(m[1], FIN) \subseteq REG$.*

Proof. Take an extended mH system $\gamma = (V, T, A, R)$ with $supp(A) = \{w\}$. If $A(w) < \infty$, then $L(\gamma)$ obviously is a finite language (every word in $L(\gamma)$ has a length not greater than $|w| \cdot A(w)$).

If $A(w) = \infty$, then $L(\gamma) \in EH([1], FIN) \subseteq EH(FIN, FIN)$. In Culik and Harju (1991) and Pixton (1995) it is proved that $H(FIN, FIN) \subseteq REG$. As REG is closed under intersection and $L(\gamma) = L(\gamma') \cap T^+$, for $\gamma' = (V, V, \{w\}, R)$, we obtain $L(\gamma) \in REG$. Hence we conclude $EH(m[1], FIN) \subseteq REG$. □

Lemma 5. *$REG \subseteq EH(\omega[1], FIN)$.*

Proof. Let $G = (N, T, S, P)$ be a regular Chomsky grammar, that is, with all the rules in P of the forms $X \to a$ or $X \to aY$, for $X, Y \in N$ and $a \in T$. Construct the ωH system

$$\gamma = (N \cup T \cup \{E, F\}, T, A, R),$$

with

$$\begin{aligned} A = &\{(SE, \infty)\} \\ &\cup \{(XaYE, \infty) \mid X \to aY \in P, X, Y \in N, a \in T\} \\ &\cup \{(XaF, \infty) \mid X \to a \in P, X \in N, a \in T\}, \end{aligned}$$

and R contains the following splicing rules:

1. $\#XE\$X\#aYE$, for $X \to aY \in P$, $X, Y \in N$, $a \in T$,
2. $\#XE\$X\#aF$, for $X \to a \in P$, $X \in N$, $a \in T$,
3. $\#F\$XXE\#$, for $X \in N$.

Starting with the axiom SE, we can simulate the derivations in G by using rules in the first group (and axioms $XaYE$), then a rule in the second group (and the corresponding axiom XaF), and finally applying the rule of type 3. Indeed, the following splicings are possible in γ for all $w \in T^+$ (the numbers below the derivation sign $\vdash$ indicate the group the splicing rule we apply is taken from):

(i) $(w|XE, X|aYE) \vdash_1 (waYE, XXE)$, for $X \to aY \in P$,
(ii) $(w|XE, X|aF) \vdash_2 (wF, XXE)$, for $X \to a \in P$,
(iii) $(w|F, XXE|) \vdash_3 (w, XXEF)$.

Therefore, $L(G) \subseteq L(\gamma)$.

Conversely, each word in A is nonterminal, and each rule in R involves words containing occurrences of E and/or of F. Let us examine the result of splicing operations different from the "legal" ones of the forms (i)–(iii) above. This means that $w \notin T^*$, hence the first term is one of XXE, $XXEF$ obtained as above, or $XaYE$, XaF in A, or other similar words obtained by splicing. We have

$$\begin{aligned}
&(X|XE, X|aYE) \vdash_1 (XaYE, XXE), && \text{for } X \to aY \in P,\\
&(X|XE, X|aF) \vdash_2 (XaF, XXE), && \text{for } X \to a \in P,\\
&(X|XE, XXE|) \vdash_3 (X, XXEXE),\\
&(X|XEF, X|aYE) \vdash_1 (XaYE, XXEF), && \text{for } X \to aY \in P,\\
&(X|XEF, X|aF) \vdash_2 (XaF, XXEF), && \text{for } X \to a \in P,\\
&(X|XEF, XXE|) \vdash_3 (X, XXEXEF).
\end{aligned}$$

We obtain words already considered, or axioms, or words never entering a splicing (like X), or new words composed of X, E, F. Moreover,

$$\begin{aligned}
&(Xa|YE, Y|bZE) \vdash_1 (XabZE, YYE), && \text{for } Y \to bZ \in P,\\
&(Xa|YE, Y|bF) \vdash_2 (XabF, YYE), && \text{for } Y \to b \in P.
\end{aligned}$$

Words $XxZE$, $|x| \geq 2$, $x \in T^+$, $X, Z \in N$, can be used only as the first term for a rule in group 1, but the leftmost X can never be removed, hence no terminal word can be produced in this way. The same holds true for XxF, $x \in T^+$, $X \in N$: the symbol F can be removed by rule 3, but then X cannot be removed.

The remarks above apply to the new words $XXEXE$, $XXEXEF$, too: in all cases we obtain "legal" words (axioms), or old or new words composed of X, E, F, or words starting with X and also containing terminals, but never turning to terminal words. Consequently, no terminal word outside $L(G)$ can be produced, hence $L(\gamma) \subseteq L(G)$. Therefore we conclude that $L(G) = L(\gamma)$.

Now, using the same construction as in the proof of Lemma 3, we can combine all axioms from A with multiplicity ∞ into only one axiom z, hence we obtain $REG \subseteq EH(\omega[1], FIN)$, which completes the proof. □

We are now able to give a complete proof for Theorem 1 stated at the beginning of this section, i.e.,

$REG = EH(m[1], FIN) \subset EH(m[2], FIN) = EH(mF_1, F_2) = RE$,
for all families F_1, F_2 *such that* $FIN \subseteq F_1 \subseteq RE$, $FIN \subseteq F_2 \subseteq RE$.

Proof of Theorem 1. For the readers' convenience, we recall the relations proved in the four lemmas above:

Lemma 2: $RE \subseteq EH(mFIN, FIN)$,
Lemma 3: $EH(mFIN, FIN) \subseteq EH(m[2], FIN)$,
Lemma 4: $EH(m[1], FIN) \subseteq REG$,
Lemma 5: $REG \subseteq EH(\omega[1], FIN)$.

Now, from the definitions we have

$$\begin{aligned}
&EH(\omega[k], F_2) \subseteq EH(m[k], F_2), \qquad k \geq 1,\\
&EH(\omega F_1, F_2) \subseteq EH(mF_1, F_2),
\end{aligned}$$

for all F_1, F_2. Thus, from Lemmas 4 and 5 we obtain

$$REG = EH(m[1], FIN).$$

Lemmas 2 and 3 imply

$$RE \subseteq EH(m[2], FIN).$$

The inclusions

$$EH(m[2], FIN) \subseteq EH(mF_1, F_2),$$

for all F_1, F_2 such that $FIN \subseteq F_1 \subseteq RE$ and $FIN \subseteq F_2 \subseteq RE$, hold by the definitions. Because of the Turing–Church thesis we also have

$$EH(mF_1, F_2) \subseteq RE,$$

for all $F_1, F_2 \subseteq RE$. Consequently,

$$RE = EH(m[2], FIN),$$

which completes the proof. □

4. Computational Completeness by Checking Context Conditions

The aim of this section is to prove that we can characterize the recursively enumerable languages without also counting the copies of the words used as in the mH systems above, but by restricting the use of the splicing rules: with each splicing rule we associate sets of words; each rule can be applied for splicing two words only when (1) all the associated words appear as subwords of the spliced words, or when (2) none of the associated words appears in the spliced words. As we show below, each of these two variants of context conditions allows us to define extended H systems which turn out to characterize the family of recursively enumerable languages; in each case we only check the presence of certain symbols in one of the spliced words.

In the following subsection we introduce the extended H systems with permitting contexts and then we prove their computational completeness; in Section 4.3 we introduce the extended H systems with forbidding contexts and prove their computational completeness.

4.1. *Extended H Systems with Permitting Contexts*

From now on we deal with H systems as in Section 2, without multiplicities associated to words (ωH systems, in the terminology of Section 3), but with the splicing regulated by context conditions. We start with "positive" context conditions.

Definition 4. An extended H system *with permitting contexts* is a quadruple

$$\gamma = (V, T, A, R),$$

where V, T, A are as in Definition 1, and R is a set of triples (we call them rules with permitting contexts) of the form

$$p = (r;\ C_1, C_2),$$

where $r = u_1\#u_2\$u_3\#u_4$ is a splicing rule over V and C_1, C_2 are finite subsets of V^+. To such a triple p we associate the word

$$\rho(p) = u_1\#u_2\$u_3\#u_4@w_1\&\cdots\&w_k@v_1\&\cdots\&v_m,$$

where $C_1 = \{w_1, \ldots, w_k\}$, $C_2 = \{v_1, \ldots, v_m\}$, $k, m \geq 0$, and $@, \&$ are new special symbols. We denote $\rho(R) = \{\rho(p) \mid p \in R\}$.

For $x, y, z, w \in V^*$ and $p \in R$ as above, we define $(x, y) \vdash_p (z, w)$ if and only if $(x, y) \vdash_r (z, w)$, every word contained in C_1 appears as a subword in x, and every word contained in C_2 appears as a subword in y (of course, when $C_1 = \emptyset$ or $C_2 = \emptyset$, then this imposes no restriction on the use of rule p).

The language generated by γ is defined in the natural way, and the family of languages $L(\gamma)$, for $\gamma = (V, T, A, R)$ as above, with $A \in F_1$ and with $\rho(R) \in F_2$, is denoted by $EH(F_1, pF_2)$; instead of F_1 we put $[k]$ when systems with at most k axioms are used. The words over V in the sets C_1, C_2 control the use of splicing rules in the same way as permitting contexts in random context grammars and in semiconditional grammars do (van derWalt, 1971; Dassow and Păun, 1989).

Remark 7. The reason for considering the type of γ defined above, using the language $\rho(R)$, is to avoid the possibility of introducing complex features in systems which can be considered simple at a superficial sight. Examine, for instance, the following system:

$$\gamma = (\{a, b, c\}, \{a, b\}, \{bac, bb\}, R),$$

with R containing the rules

$$\begin{aligned} p_1 &= (a\#c\$b\#a;\ \emptyset, \emptyset), && \\ p_{2,n} &= (ba^n\#c\$b\#b;\ \{b, c\}, \emptyset), && n \geq 1, \quad n \in \{2^i \mid i \geq 0\}, \\ p_{3,n} &= (ba^n\#c\$b\#b;\ \{b\}, \{c\}), && n \geq 1, \quad n \notin \{2^i \mid i \geq 0\}. \end{aligned}$$

One can see that

$$L(\gamma) = \{ba^{2^n}b \mid n \geq 1\} \cup \{bb\}$$

(by rule p_1 we can produce all words $ba^m c$, $m \geq 1$, whereas $p_{2,n}$ replaces c by b in such words having $m = 2^n$; rules $p_{3,n}$ can never be used). The language $L(\gamma)$ is not regular (in fact, it is not context-free) and this is directly due to rules of type $p_{2,n}$. However, if we consider the type of γ according to the set of splicing rules only, then γ is of type (FIN, REG), which is misleading:

$$\begin{aligned}\{a\#c\$b\#a\} &\cup \{ba^n\#c\$b\#b \mid n \geq 1, n = 2^i, i \geq 0\} \\ &\cup \{ba^n\#c\$b\#b \mid n \geq 1, n \neq 2^i, i \geq 0\} \\ &= \{a\#c\$b\#a\} \cup \{ba^n\#c\$b\#b \mid n \geq 1\} \in REG.\end{aligned}$$

The same result is obtained if we do not separate the symbols in the conditions associated with rules $p_{2,n}$ and $p_{3,n}$; specifically, we get the regular language

$$\{a\#c\$b\#a@\} \cup \{ba^n\#cb\#b@bc \mid n \geq 1\}.$$

4.2. *The Power of Extended H Systems with Permitting Contexts*

A result similar to Theorem 1 also holds for H systems with permitting contexts, with the difference that the hierarchy on the number of axioms now collapses to a hierarchy with one level only. Specifically, the following result holds.

Theorem 2. *$EH([1], pFIN) = EH(F_1, pF_2) = RE$, for all families F_1, F_2 such that $FIN \subseteq F_1 \subseteq RE$, $FIN \subseteq F_2 \subseteq RE$.*

This subsection is devoted to proving this theorem. The following two lemmas are used to achieve this aim.

Lemma 6. *$RE \subseteq EH(FIN, pFIN)$.*

Proof. Consider a type-0 Chomsky grammar $G = (N, T, S, P)$ like in the proof of Lemma 2, in the Kuroda normal form; let Lab denote the set of labels of the rules in P and denote $U' = U \cup \{B\}$, $U = N \cup T$, for B being a new symbol. We now construct the extended H system with permitting contexts

$$\gamma = (V, T, A, R),$$

where

$$\begin{aligned}V &= N \cup T \cup \{B, E, E', F, F', X, X', Y, Z\} \cup \{Y_\alpha \mid \alpha \in U' \cup Lab\}, \\ A &= \{F'Z, XBSY, XZ, ZE, ZE', ZF, ZY\} \\ &\quad \cup \{ZY_\alpha, X'\alpha Z \mid \alpha \in U\} \cup \{ZY_r, X'vZ \mid r\colon u \to v \in P\}\end{aligned}$$

and R contains the following rules with permitting contexts:

1. $(\#uY\$Z\#Y_r; \{X\}, \emptyset)$, for $r: u \to v \in P$,
2. $(X\#\$X'v\#Z; \{Y_r\}, \emptyset)$, for $r: u \to v \in P$,
3. $(\#Y_r\$Z\#Y; \{X'\}, \emptyset)$, for $r: u \to v \in P$,
4. $(X'\#\$X\#Z; \{Y\}, \emptyset)$,
5. $(\#\alpha Y\$Z\#Y_\alpha; \{X\}, \emptyset)$, for $\alpha \in U'$,
6. $(X\#\$X'\alpha\#Z; \{Y_\alpha\}, \emptyset)$, for $\alpha \in U'$,
7. $(\#Y_\alpha\$Z\#Y_\alpha; \{X'\}, \emptyset)$, for $\alpha \in U'$,
8. $(\#Y\$Z\#F; \{X\}, \emptyset)$,
9. $(XB\#\$F'\#Z; \{F\}, \emptyset)$,
10. $(\#F\$ZE\#; \{F'\}, \emptyset)$,
11. $(F'\#\$\#ZE'; \{F'\}, \emptyset)$.

The idea behind this construction is the following. The rules from groups 1–4 allow us to simulate rules from P on a suffix of the first term of the splicing. A rule in group 1 cuts the left-hand side u of the production $r: u \to v \in P$ from the right-hand end of the word and the associated symbol Y_r memorizes the label of this rule; in the presence of Y_r a rule from group 2 will introduce the right-hand side v of the rule with label r on the left-hand end of the word together with X' instead of X; then Y_r is again replaced by Y (by using the appropriate rule from group 3), and X' is again replaced by X (by using the rule from group 4).

However, we must be able to simulate the application of a rule from P at an arbitrary position of the underlying sentential form, not only at the right-hand end of the word. To this aim, the rules in groups 5–7 and 4 allow us to "rotate" the word: a rule in group 5 cuts a symbol α from the right-hand end of the word, Y_α memorizes this symbol, in its presence a rule from group 6 will introduce α in the left-hand end (together with X'), then Y_α is again replaced by Y (by using the appropriate rule from group 7), and X' is again replaced by X (by using the rule from group 4). Any circular permutation can be obtained in this way.

In a quite similar way, the rules from groups 8–11 finally allow us to remove the markers X and Y by first replacing Y by F and X by F' and then by removing F and F'.

We now look in some detail at the way the ideas mentioned above work.

When simulating derivation steps in G, we start from $XBSY$, and at every step the marker X and its variant X' as well as Y and its variants Y_β, $\beta \in U' \cup Lab$, are present to indicate the ends of the word. Moreover, in each moment the symbol B tells us where the beginning of the word whose permutation we consider is.

All the splicing rules with permitting contexts contained in R require an occurrence of the symbol Z in the second term of the splicing; in fact, these words are meant to be taken from A. If we start with rule 1 applied to $XBSY$ and ZvY, for some $S \to v \in P$, this starts the simulation of a derivation in G. In general, having a word Xx_1Bx_2uY and $r: u \to v \in P$, we can obtain Xvx_1Bx_2Y by using the associated rules in groups 1–4. This corresponds to a derivation step $x_2ux_1 \Rightarrow x_2vx_1$ in G. The corresponding steps in γ have the following form:

1. $(Xw|uY, Z|Y_r) \vdash_p (XwY_r, ZuY)$, for $p = (\#uY\$Z\#Y_r; \{X\}, \emptyset)$,
   where $r: u \to v \in P$ and $w \in U^*\{B\}U^*$;

2. $(X|wY_r, X'v|Z) \vdash_p (XZ, X'vwY_r)$, for $p = (X\#\$X'v\#Z; \{Y_r\}, \emptyset)$, where $r\colon u \to v \in P$ and $w \in U^*\{B\}U^*$;
3. $(X'vw|Y_r, Z|Y) \vdash_p (X'vwY, ZY_r)$, for $p = (\#Y_r\$Z\#Y; \{X'\}, \emptyset)$, where $r\colon u \to v \in P$ and $w \in U^*\{B\}U^*$;
4. $(X'|vwY, X|Z) \vdash_p (X'Z, XvwY)$, for $p = (X'\#\$X\#Z; \{Y\}, \emptyset)$, where $r\colon u \to v \in P$ and $w \in U^*\{B\}U^*$.

As additional results (that cannot already be found in the set of axioms A) of the splicing derivations listed above we obtain the words

ZuY, for $r\colon u \to v \in P$,

and also the word $X'Z$ if $v \neq \lambda$ for all $r\colon u \to v \in P$.

To each word $Xw\alpha Y$, $\alpha \in U$, $w \in U^*\{B\}U^*$ (resp. $Xw\alpha Y$, $\alpha = B$, $w \in U^*$) we can also apply the appropriate rule from group 5 and then proceed by applying the appropriate rules from groups 6 and 7; finally, by using the rule in group 4 again, we obtain the word $X\alpha wY$. The symbol α has been moved from the right-hand end to the left-hand end of the word, which is exactly what we need for rotating the underlying sentential form:

1. $(Xw|\alpha Y, Z|Y_\alpha) \vdash_p (XwY_\alpha, Z\alpha Y)$, for $p = (\#\alpha Y\$Z\#Y_\alpha; \{X\}, \emptyset)$, where $\alpha \in U$ and $w \in U^*\{B\}U^*$ or $\alpha = B$ and $w \in U^*$;
2. $(X|wY_\alpha, X'\alpha|Z) \vdash_p (XZ, X'\alpha wY_\alpha)$, for $p = (X\#\$X'\alpha\#Z; \{Y_\alpha\}, \emptyset)$, where $\alpha \in U$ and $w \in U^*\{B\}U^*$ or $\alpha = B$ and $w \in U^*$;
3. $(X'\alpha w|Y_\alpha, Z|Y) \vdash_p (X'\alpha wY, ZY_\alpha)$, for $p = (\#Y_\alpha\$Z\#Y; \{X'\}, \emptyset)$, where $\alpha \in U$ and $w \in U^*\{B\}U^*$ or $\alpha = B$ and $w \in U^*$;
4. $(X'|\alpha wY, X|Z) \vdash_p (X'Z, X\alpha wY)$, for $p = (X'\#\$X\#Z; \{Y\}, \emptyset)$, where $\alpha \in U$ and $w \in U^*\{B\}U^*$ or $\alpha = B$ and $w \in U^*$.

As additional results (that cannot already be found in the set of axioms A) of the splicing derivations listed above we obtain the words

$Z\alpha Y$, for $\alpha \in U'$,

and, if $v \neq \lambda$ for all $r\colon u \to v \in P$, then again $X'Z$ too.

Notice again that every word obtained from $XBSY$ so far and not containing the symbol Z is of the form $\alpha_1 x_1 B x_2 \alpha_2$, with (α_1, α_2) being one of the pairs (X, Y), (X, Y_α), (X', Y_α), (X', Y), $\alpha \in U' \cup Lab$, hence these symbols appearing as permitting contexts in the splicing rules of R control the work of γ precisely.

In order to obtain a terminal word we have to use rules from groups 8–11, in this order (Y must be present when using rule 8); it is easy to see that in fact we have to start with a word of the form $XBwY$ with $w \in U^*$:

1. $(Xw'|Y, Z|F) \vdash_p (Xw'F, ZY)$, for $p = (\#Y\$Z\#F; \{X\}, \emptyset)$, where $w' \in U^*\{B\}U^*$;
2. $(XB|wF, F'|Z) \vdash_p (XBZ, F'wF)$, for $p = (XB\#\$F'\#Z; \{F\}, \emptyset)$, where $w \in U^*$;

3. $(F'w|F, ZE|) \vdash_p (F'w, ZEF)$, for $p = (\#F\$ZE\#; \{F'\}, \emptyset)$, where $w \in U^*$;
4. $(F'|w, |ZE') \vdash_p (F'ZE', w)$ for $p = (F'\#\$\#ZE'; \{F'\}, \emptyset)$, where $w \in U^*$.

As additional results (that cannot already be found in the set of axioms A) of the splicing derivations listed above we obtain the words

$$XBZ,\ ZEF,\ F'ZE'.$$

In sum, in γ we can produce every terminal word that can be produced by G, i.e. $L(\gamma) \supseteq L(G)$.

Conversely, no parasitic terminal words can be generated in γ, i.e., $L(\gamma) \subseteq L(G)$:

All the words generated in addition to the axioms represent a sentential form of a derivation in G (maybe in a rotated version) or contain the symbol Z. Only the words containing this symbol Z can be used as second terms of a splicing in γ, and, except for the word $F'ZE'$, all words that really can be used already appear in the set of axioms A; with $F'ZE'$ we can also do the following derivations involving words of the forms $F'^l ZE'$ as well as $F'^k w$ and $F'^k wF$, for $l, k \geq 0$: If $k = 0$, then we obtain w (resp. wF), where the word wF cannot be processed any more, because the rule in group 10 needed for eliminating the symbol F requires the occurrence of the symbol F'; if the word w obtained is not terminal, then no further step can be done. If $k \geq 2$, then we can only obtain similar words like from $F'wF$. The words $F'^l ZE'$ for $l \geq 1$ can be used freely in the same way as the word ZE' itself.

1. $(F'^m|F'^k ZE', F'^l|ZE') \vdash_p (F'^m ZE', F'^l F'^k ZE')$,
   for $p = (F'\#\$\#ZE'; \{F'\}, \emptyset)$,
   where $m, k, l \geq 0$ and $m + k \geq 1$;
2. $(F'^m|F'^k w, F'^l|ZE') \vdash_p (F'^m ZE', F'^l F'^k w)$,
   for $p = (F'\#\$\#ZE'; \{F'\}, \emptyset)$,
   where $m, k, l \geq 0$ and $m + k \geq 1$ and $w \in U^*$;
3. $(F'^m|F'^k wF, F'^l|ZE') \vdash_p (F'^m ZE', F'^l F'^k wF)$,
   for $p = (F'\#\$\#ZE'; \{F'\}, \emptyset)$,
   where $m, k, l \geq 0$ and $m + k \geq 1$ and $w \in U^*$;
4. $(F'^k w|F, ZE|) \vdash_p (F'^k w, ZEF)$, for $p = (\#F\$ZE\#; \{F'\}, \emptyset)$,
   where $k \geq 1$ and $w \in U^*$.

Moreover, using the word ZEF does not yield new results, because $(F'^k w|F, ZE|F) \vdash_p (F'^k wF, ZEF)$ for $p = (\#F\$ZE\#; \{F'\}, \emptyset)$, where $k \geq 1$ and $w \in U^*$.

In sum, we obtain $L(\gamma) = L(G)$. □

Remark 8. Note that in the rules $(p; C_1, C_2)$ of the H system with permitting contexts constructed in the proof of Lemma 6, the pairs (C_1, C_2) of permitting contexts are of the special form $(\{X\}, \emptyset)$ for some nonterminal X, i.e., we only check the occurrence of one nonterminal in the first term of the splicing. This can be viewed as a normal form result for our systems. Also observe that when considering the rules $(u_1\#u_2\$u_1\#u_2; C_1, C_1)$, $(u_3\#u_4\$u_3\#u_4; C_2, C_2)$ for each rule $(u_1\#u_2\$u_3\#u_4; C_1, C_2)$ in the previous proof, the generated language is the same: the place of using such rules is unique, hence we always have $(x, y) \vdash_p (x, y)$.

Moreover, the longest sites appear in rules of type 2, and they are of length four: for $|v| = 2$ we have $|X'vZ| = 4$. All other rules have sites of length two, but for rules in group 1 we have $|uY| = 3$ for $|u| = 2$.

Remark 9. A permitting context splicing rule as in the proof of Lemma 6, i.e., with only one checked symbol which should appear at an end of the string, might be implemented in the following way.

The restriction enzymes work only on double-stranded sequences. We melt the solution in order to obtain single-stranded sequences (raising the temperature, the two strands of a double-stranded DNA sequence are separated, resulting in single-stranded sequences of nucleotides). Add a primer which contains the complement of the permitting symbol. (A primer is a short single-stranded sequence of nucleotides, complementary to a subword of a given single-stranded sequence; due to the complementarity, the primer anneals to the sequence and this activates the polymerization process by which the whole sequence will become double-stranded—provided the environment contains a sufficient amount of the single nucleotides requested by the doubling.) Hence, this primer will be attached only to single-stranded sequences containing the designated symbol (at an end of it). Only these single-stranded sequences will enter the polymerization reaction, leading to double-stranded sequences, hence the enzyme will act on them only.

The construction in the previous proof is an extension of the idea used in the third example in Section 2.3, and it also resembles the way of simulating Turing Machines by Post TAG systems in Minsky (1961). The same idea is also used in the proof of the inclusion $RE \subseteq EH(FIN, REG)$ from Păun (1996b).

Lemma 7. $EH(FIN, pFIN) \subseteq EH([1], pFIN)$.

Proof. Consider an H system $\gamma = (V, T, A, R)$ with permitting contexts, and suppose that $A = \{w_1, w_2, \ldots, w_n\}, n \geq 2$. Take the new symbols $c_0, c_1, \ldots, c_n, c_{n+1}$ and construct the new H system with permitting contexts

$$\gamma' = (V \cup \{c_0, c_1, \ldots, c_n, c_{n+1}\}, T, \{w\}, R'),$$

where

$$w = c_0 c_1 w_1 c_2 w_2 \cdots c_n, c_{n+1} c_0,$$
$$R' = R \cup R''$$

with R'' containing the following rules with permitting contexts:

1. $(c_0 \# c_1 \$ c_i \# w_i c_{i+1}; \emptyset, \emptyset)$, for $1 \leq i \leq n$,
2. $(c_0 w_i \# c_{i+1} \$ c_{n+1} \# c_0; \emptyset, \emptyset)$, for $1 \leq i \leq n$,
3. $(\# c_0 c_1 \$ c_0 \# w_i c_0; \emptyset, \emptyset)$, for $1 \leq i \leq n$,
4. $(w_i \# c_0 \$ c_{n+1} c_0 \#; \emptyset, \emptyset)$, for $1 \leq i \leq n$.

It is easy to see that by splicing w with itself using a rule of type 1 we obtain the word $c_0 w_i c_{i+1} w_{i+1} c_{i+2} \cdots c_n w_n c_{n+1} c_0$ (and $c_0 c_1 w_1 \cdots c_i c_1 w_1 \cdots c_n w_n c_{n+1} c_0$). By splicing this word with w using the rule in group 2 with the suitable value of i we get $c_0 w_i c_0$ (and $c_0 c_1 w_1 \cdots c_{n+1} c_{i+1} w_{i+1} \cdots c_n w_n c_{n+1} c_0$). Using rules from groups 3 and 4, we can now remove the front c_0 and then the back c_0. All the axioms of γ can be detached from w; now applying rules in R we can produce each word of $L(\gamma)$, hence $L(\gamma) \subseteq L(\gamma')$.

Conversely, if we apply a rule of R to two copies of w (possibly on subwords w_i, w_j which cannot be spliced correctly when they are separated, due to the restrictions imposed by the permitting contexts), producing words of the form $z_1 c_i z_2 c_j z_3$, with z_2 in V^*, then z_2 can be separated from the neighboring c_i, c_j if and only if $j = i + 1$ and $z_2 = w_i$, hence no new word can be produced in this way. The same holds if the word $z_1 c_i z_2 c_j z_3$ is obtained after several splicings, possibly also using words of the form $c_0 w_i c_{i+1} \cdots c_n w_n c_{n+1} c_0$, etc., or $c_0 w_i c_0$, obtained after using rules in groups 1 and 2. On the other hand, splicing copies of w or of $c_0 w_i c_{i+1} \cdots c_n w_n c_{n+1} c_0$, etc., and $c_0 w_i c_0$ as above using rules in R, we can obtain only words starting and ending with c_0, hence we can separate subwords $c_i z c_j$, $z \in V^*$, out of them if and only if $j = i + 1$ and $z = w_i$.

If a rule in R is applied to two words of the form $w_i c_0$, $w_j c_0$ obtained after using rules of type 3, then instead of v and w we obtain $v c_0$ and $w c_0$, where v and w are correct splicing results. If a rule in R is applied to a word $w_i c_0$ and to another word of a different form (hence also starting with a symbol c_0), then we either obtain a word both starting and ending with c_0 (when $w_i c_0$ is the second term of the splicing), and, as above, we can separate from it only subwords of the form $c_j w_j c_{j+1}$), or we obtain a word $z_1 c_j z_2$, $z_1 \in V^*$, with w_i contributing to z_1, and z_1 cannot be separated from c_j. Therefore, the rules in R can be used successfully only for splicing the result of using rules in groups 1–3 (and 4), and this corresponds to the work of γ, starting from the axioms in A. Therefore, $L(\gamma') \subseteq L(\gamma)$ is also true. □

Remark 10. When considering rules associated to rules in R'' by the reflexivity assumption, the language generated is not modified, because these rules can be applied only on places defined by symbols $c_0, c_1, \ldots, c_{n+1}$, hence what we obtain always consists of complete words w_i, $1 \le i \le n$, and symbols $c_0, c_1, \ldots, c_{n+1}$.

We now prove the relations in Theorem 2, i.e.,

$EH([1], pFIN) = EH(F_1, F_2) = RE$,
for all families F_1, F_2 *such that* $FIN \subseteq F_1 \subseteq RE$, $FIN \subseteq F_2 \subseteq RE$.

Proof of Theorem 2. In the previous two lemmas we have proved:

Lemma 6 : $RE \subseteq EH(FIN, pFIN)$,
Lemma 7 : $EH(FIN, pFIN) \subseteq EH([1], pFIN)$.

Combining these relations, we get

$RE \subseteq EH([1], pFIN)$.

The inclusions

$$EH([1], pF_2) \subseteq EH(F_1, pF_2),$$

for all F_1, F_2 such that $FIN \subseteq F_1 \subseteq RE, FIN \subseteq F_2 \subseteq RE$, hold by the definitions. Because of the Turing–Church thesis, we have

$$EH(F_1, pF_2) \subseteq RE$$

for all $F_1, F_2 \subseteq RE$. Consequently,

$$RE = EH([1], pFIN),$$

which completes the proof. □

4.3. *Extended H Systems with Forbidding Contexts*

Instead of controlling the applicability of a splicing rule by using permitting contexts, i.e., by checking the occurrence of specific subwords (symbols) in the underlying words, we can also control the applicability of a splicing rule by using forbidding contexts, i.e., by forbidding the occurrence of specific subwords (symbols) in the underlying words.

These forbidding contexts can be interpreted as *inhibitors* of the associated rules.

Definition 5. An extended H system *with forbidding contexts* is a quadruple

$$\gamma = (V, T, A, R),$$

where V, T, A are as in Definition 1, and R is a set of triples (we call them rules with forbidding contexts) of the form

$$p = (r;\ D_1, D_2),$$

where $r = u_1\#u_2\$u_3\#u_4$ is a splicing rule over V and D_1, D_2 are finite subsets of V^+. As in Definition 4, the triple p can also be represented by the word

$$\rho(p) = u_1\#u_2\$u_3\#u_4@w_1\&\cdots\&w_k@v_1\&\cdots\&v_m,$$

where $D_1 = \{w_1, \dots, w_k\}$ and $D_2 = \{v_1, \dots, v_m\}$, $k, m \geq 0$. We denote $\rho(R) = \{\rho(p) \mid p \in R\}$.

For $x, y, z, w \in V^+$ and $p \in R$ as above, we define $(x, y) \vdash_p (z, w)$ if and only if $(x, y) \vdash_r (z, w)$, no word contained in D_1 appears as a subword in x, and no word contained in D_2 appears as a subword in y (of course, when $D_1 = \emptyset$ or $D_2 = \emptyset$, then this imposes no restriction on the use of rule p).

The language generated by γ is defined in the natural way, and the family of languages $L(\gamma)$, for $\gamma = (V, T, A, R)$ as above, with $A \in F_1$ and $\rho(R) \in F_2$, is denoted by $EH(F_1, fF_2)$. Instead of F_1 we put $[k]$ when systems with at most k axioms are used.

Theorem 3. *$EH([1], fFIN) = EH(F_1, fF_2) = RE$, for all families F_1, F_2 such that $FIN \subseteq F_1 \subseteq RE$, $FIN \subseteq F_2 \subseteq RE$.*

The relations above are consequences of the following two lemmas, of the Turing–Church thesis, and of the obvious inclusion $EH([1], fFIN) \subseteq EH(F_1, fF_2)$, F_1, F_2 as in the theorem above.

Lemma 8. *$RE \subseteq EH(FIN, fFIN)$.*

Proof. Consider a type-0 grammar $G = (N, T, S, P)$ like in the proof of Lemma 2, let Lab be the set of labels of the rules in P, and denote $U' = N \cup T \cup \{B\}$. We now construct the extended H system with forbidding contexts

$$\gamma = (V, T, A, R),$$

where V, T, and A are as in the proof of Lemma 6 and R contains the following rules with forbidding contexts:

1. $(\#uY\$Z\#Y_r;\ Q - \{X, Y\}, \emptyset)$, for $r\colon u \to v \in P$,
2. $(X\#\$X'v\#Z;\ Q - \{X, Y_r\}, \emptyset)$, for $r\colon u \to v \in P$,
3. $(\#Y_r\$Z\#Y;\ Q - \{X', Y_r\}, \emptyset)$, for $r\colon u \to v \in P$,
4. $(X'\#\$X\#Z;\ Q - \{X', Y\}, \emptyset)$,
5. $(\#\alpha Y\$Z\#Y_\alpha;\ Q - \{X, Y\}, \emptyset)$, for $\alpha \in U'$,
6. $(X\#\$X'\alpha\#Z;\ Q - \{X, Y_\alpha\}, \emptyset)$, for $\alpha \in U'$,
7. $(\#Y_\alpha\$Z\#Y_r;\ Q - \{X', Y_\alpha\}, \emptyset)$, for $\alpha \in U'$,
8. $(\#Y\$Z\#F;\ Q - \{X, Y\}, \emptyset)$,
9. $(XB\#\$F'\#Z;\ Q - \{F, X\}, \emptyset)$,
10. $(\#F\$ZE\#;\ Q - \{F, F'\}, \emptyset)$,
11. $(F'\#\$\#ZE';\ Q - \{F'\}, \emptyset)$,

where

$$Q = \{X, X', Y, Z, F, F'\} \cup \{Y_\alpha \mid \alpha \in U'\} \cup \{Y_r \mid r : u \to v \in P\}$$

is the set of all symbols occurring in the permitting contexts used by the extended H system in the proof of Lemma 6. It is easy to see that the permitting contexts in the rules of the extended H system with permitting contexts constructed in the proof of Lemma 6 correspond in a precise manner to the forbidding contexts in the H system with forbidding contexts constructed as above. (All words produced by these systems start and end with symbols in Q, except for the words in the last steps of the process, when rules 8–11 are used. The presence of X, for instance, means precisely that neither X' nor F' is present in the leftmost position of the first term of the splicing. Similar arguments hold true for the rightmost position of the first term of the splicing, where Y and its variants are present.) Hence, according to the arguments given in the proof of Lemma 6, we conclude $L(\gamma) = L(G)$. □

Remark 11. Note that in the rules $(p_1; D_1, D_2)$ of the extended H system with forbidding contexts constructed in the proof of Lemma 8, the pairs (D_1, D_2) of forbidding contexts are of the special form $(M, \emptyset)$ with $M \subseteq V - T$, i.e., we only check the nonoccurrence of some nonterminals in the first term of the splicing. In the same way as for the construction in Lemma 6, we can show that the reflexivity assumption does not change the language generated.

Lemma 9. *$EH(FIN, fFIN) \subseteq EH([1], fFIN)$.*

Proof. For an extended H system with forbidding contexts $\gamma = (V, T, A, R)$ with $A = \{w_1, w_2, \dots, w_n\}$, we construct an extended H system

$$\gamma' = (V \cup \{c\}, T, \{w\}, R'),$$

where

$$w = ccw_1cw_2c \cdots cw_ncc$$

and

$$R' = \{(p; F_1 \cup \{c\}, F_2 \cup \{c\}) \mid (p; F_1, F_2) \in R\}$$
$$\cup \{(\#cc\$c\#; \emptyset, \emptyset), (\#c\$cc\#; \emptyset, \emptyset)\}.$$

The splicing rule $\#cc\$c\#$ applied to two copies of w allows us to cut one w at the symbol c in front of some axiom w_i, i.e., as one new word we obtain $w_ic \cdots w_ncc$, and the splicing rule $\#c\$cc\#$ applied to this word and to another copy of w then allows us to separate the axiom w_i.

As the rules from R can only be applied to words not containing the nonterminal c any more, we obviously obtain $L(\gamma') = L(\gamma)$. □

Another way of controlling the work of splicing rules is to impose sequencing restrictions (like in programmed grammars or in graph controlled grammars, etc., see Dassow and Păun (1989)). Probably other control mechanisms used in formal language theory for regulating the work of context-free grammars can also be used as in Theorems 2 and 3 above, but we do not follow this direction further here. It is a matter of practical feasibility to choose one or the other of these regulating mechanisms, according to the possibility of implementing them. Theorems 1–3 prove in a convincing manner that the full power of Turing Machines can be reached by considering extended H systems with finite components (axiom set and sets of rules), provided such control mechanisms are involved.

5. Universal H Systems

The results in the previous sections prove that *finite* H systems of the types considered are computationally complete, but this does not mean that *programmable* computers based on splicing can be constructed. To this aim, it is necessary to find *universal H systems* as described in Section 1.4, i.e., systems with all components but one (the set of axioms) fixed, able to behave as any given H system γ, when a code of γ is introduced in the set of axioms of the universal system.

Surprisingly enough, universal H systems exist for all the types of extended H systems we have considered in the previous sections, and this is a consequence of the very proofs of Lemmas 2, 6, and 8.

Definition 6. Given an alphabet T and two families of languages, F_1, F_2, a construct

$$\gamma_U = (V_U, T, A_U, R_U),$$

where V_U is an alphabet, $A_U \subseteq V_U^*$, $A_U \in F_1$, and $R_U \subseteq V_U^*\#V_U^*\$V_U^*\#V_U^*$, $R_U \in F_2$, is said to be a *universal* H system of type (F_1, F_2), if for every $\gamma = (V, T, A, R)$ of type (F_1, F_2) there is a language A_γ such that $A_U \cup A_\gamma \in F_1$ and $L(\gamma) = L(\gamma'_U)$, where $\gamma'_U = (V_U, T, A_U \cup A_\gamma, R_U)$.

The particularizations of this definition to mH systems or to H systems with permitting (resp. forbidding) contexts are obvious.

The restriction to a given terminal alphabet cannot be avoided, but this is anyway imposed by the fact that the DNA alphabet has only four letters. Moreover, in order to aim at obtaining universal H systems, we must also be able to bound the number of nonterminal symbols used. Fortunately, this can indeed be done.

Lemma 10. *Given an extended H system $\gamma = (V, T, A, R)$ of type (F_1, F_2), for F_1, F_2 families of languages closed under λ-free morphisms, we can construct an extended H system $\gamma' = (\{c_1, c_2\} \cup T, T, A', R')$ of the same type (F_1, F_2), such that $L(\gamma) = L(\gamma')$. This is also true for γ being an mH system or an H system with permitting (resp. forbidding) contexts.*

Proof. If $V - T = \{Z_1, \dots, Z_n\}$, then we consider the morphism $h\colon V^+ \to (\{c_1, c_2\} \cup T)^+$, defined by

$$h(Z_i) = c_1 c_2^i c_1, \qquad 1 \le i \le n,$$
$$h(a) = a, \qquad a \in T.$$

Then

$$A' = h(A),$$
$$R' = \{h(u_1)\#h(u_2)\$h(u_3)\#h(u_4) \mid u_1\#u_2\$u_3\#u_4 \in R\}.$$

The equality $L(\gamma) = L(\gamma')$ is obvious, because due to the form of the axioms in A'

and of the rules in R', the blocks $c_1c_2^ic_1$, $1 \leq i \leq n$, are never broken by splicing, they behave in the same way as the corresponding symbols Z_i do.

For the case when γ is an mH or an H system with permitting (resp. forbidding) contexts, the definitions of the corresponding components of γ' are obvious, transferring them from γ to γ' by the morphism h. □

Note that in the proof of this lemma it is important to work with strings as permitting or forbidding contexts, not with symbols, as in the proofs of Lemmas 6 and 8. This, however, does not introduce a significant additional difficulty in checking these conditions, because the way of checking the presence of a symbol in a string by using a primer, as described in Remark 9, works in a similar way when we have to check the presence of a substring. This is true at least for short strings. Moreover, we have obtained a system γ' with only two auxiliary symbols. Increasing the number of nonterminals allowed (but still having this number bounded), we can decrease the length of the codings of the nonterminals of the system γ we have started with, thus decreasing the length of the necessary primer, hence the difficulty of checking the context conditions. The balance of the number of nonterminals and of the length of the mentioned codings is a matter of practical interest, hence it should be investigated under specific circumstances.

From a practical point of view, the main results of this paper are the following two theorems:

Theorem 4. *For every given alphabet T there exists an extended mH system of type $(m[2], FIN)$ which is universal for the class of extended mH systems with the terminal alphabet T.*

Proof. Consider an alphabet T and two different symbols c_1, c_2 not in T.

For the class of type-0 Chomsky grammars with given terminal alphabet, there are universal grammars, i.e., constructs $G_U = (N_U, T, -, P_U)$ such that for any given grammar $G = (N, T, S, P)$ there is a word $w(G) \in (N_U \cup T)^+$ (the "code" of G) such that $L(G'_U) = L(G)$ for $G'_U = (N_U, T, w(G), P_U)$. (The language $L(G'_U)$ consists of all terminal words z such that $w(G) \Longrightarrow^* z$ using the rules in P_U.) This follows from the existence of universal Turing Machines (Turing, 1936) and the way of passing from Turing Machines to type-0 grammars and conversely, or it can be proved directly (an effective construction of a universal type-0 grammar can be found in Calude and Păun (1981)).

For a given universal type-0 grammar $G_U = (N_U, T, -, P_U)$, we follow the construction in the proof of Lemma 2, obtaining an extended mH system $\gamma_1 = (V_1, T, A_1, R_1)$, where the axiom (with multiplicity one) $w_0 = X_1^2YSX_2^2$ is not considered. Note that all other axioms in A_1 (all having infinite multiplicity) and the rules in R_1 depend on N_U, T, and P_U only, hence they are fixed.

As in the proof of Lemma 3, we now pass from γ_1 to $\gamma_2 = (V_2, T, A_2, R_2)$, with at most two axioms in A_2. In fact, as A_1 contains only axioms with infinite multiplicity, A_2 consists of only one word (that one denoted by z in the proof of Lemma 3), which has infinite multiplicity.

We now follow the proof of Lemma 10, codifying all symbols in $V_2 - T$ by words over $\{c_1, c_2\}$; the system obtained,

$$\gamma_U = (\{c_1, c_2\} \cup T, T, A_U, R_U),$$

is the universal mH system we are looking for.

Indeed, take an arbitrary extended mH system $\gamma_0 = (V, T, A, R)$. From the Turing–Church thesis we know that $L(\gamma_0) \in RE$, hence there is a type-0 grammar $G_0 = (N_0, T, S_0, P_0)$ such that $L(\gamma_0) = L(G_0)$ (in fact, the grammar G_0 can be constructed directly and in an effective way). Construct the code of G_0, $w(G_0)$, as imposed by the definition of universal type-0 grammars one uses, consider the word

$$w_0' = X_1^2 Y w(G_0) X_2^2,$$

corresponding to the axiom w_0 in the proof of Lemma 2, then codify w_0' over $\{c_1, c_2\} \cup T$ as we have done above with the axioms of γ_2. Denote the obtained word by $w(\gamma_0)$. Then $L(\gamma_U') = L(\gamma_0)$, for $\gamma_U' = (\{c_1, c_2\} \cup T, T, \{(w(\gamma_0), 1)\} \cup A_U, R_U)$.

This can be seen easily: In the proof of Lemma 2, the system γ simulates the work of G, starting from the axiom S of G, bracketed as in $X_1^2 Y S X_2^2$. If instead of S we put an arbitrary word x over the alphabet of G, then in γ we obtain exactly the language of terminal words y such that $x \Longrightarrow^* y$ in G. If we start from a universal grammar G_U and S is replaced by the code $w(G_0)$ of a type-0 grammar G_0 equivalent with γ_0, then the system γ_U, associated as above with the universal grammar G_U, will simulate the work of G_U, starting from $w(G_0)$. Hence $L(\gamma_U') = L(G_U') = L(G_0) = L(\gamma_0)$, for $G_U' = (N_U, T, w(G_0), P_U)$. □

Theorem 5. *For every given alphabet T, there is an extended H system of type* $([1], FIN)$ *with permitting (resp. forbidden) contexts that is universal for the class of extended H systems with permitting (resp. forbidding) contexts and with the terminal alphabet* T.

Proof. This can be proved in the same way as in Theorem 4 above, by using the constructions given in the proofs of Lemmas 6, 8, and 10. □

Note that the universal H systems γ_U furnished by the previous proofs contain only one axiom and that in order to enable them to simulate any given extended H system γ, we only have to add to γ_U one more axiom (of multiplicity one in the case of mH systems).

The "computers" based on γ_U seem to be quite economical, as far as their "programming" is concerned.

6. Conclusions

The fact that the splicing operation is very powerful (as a formal operation on words and languages) has been proved in various places. The usual way to do this is to characterize the family of recursively enumerable languages using the splicing operation and other "weak" prerequisites (other operations, special forms of splicing rules (Păun, 1996a, 1996b; Păun et al., 1996; Păun and Salomaa, 1996), or additional languages such as Dyck languages, palindrome languages, etc. (Yokomori and Kobayaski, 1995)). Our results in Sections 3 and 4 are the strongest possible of this type, because we only use the splicing, the intersection with a language of the form T^+ (which according to Păun et al. (1996), cannot be avoided), and systems with finite sets of axioms and finite sets of splicing rules. While it is true that we use the additional control mechanism of multiplicity counting or permitting (resp. forbidding) contexts, such features are essential and cannot be removed; indeed, in view of Culik and Harju (1991) and Pixton (1995), ordinary finite splicing systems can produce regular languages only.

It is rather interesting to notice in this context the tremendous influence that certain apparently minor changes in the structure of an extended H system have on its generative power. For instance, in the case of mH systems, the transition from one axiom to two axioms causes an increase from the power of finite automata/regular grammars to the full power of Turing Machines/type-0 Chomsky grammars (Theorem 1). Similarly in the case of H systems the addition of permitting contexts, one for each rule, strengthens the generative power to that of Turing Machines/type-0 Chomsky grammars (Lemma 6).

However, as we have already pointed out, the most significant of the results we obtained is the existence of universal H systems of various types. This theoretically proves the feasibility of designing universal and programmable DNA computers, where a program consists of a single word to be added to the axiom set of the universal computer. In the particular case of mH systems, these program axioms have multiplicity one, while an unbounded number of copies of all the other axioms is available. The "fixed" axioms of the computer can be interpreted as a sort of nonerasable stored information available for free.

As a closing remark, note that the proofs of Theorems 4 and 5 rely on one hand on the lemmas in Sections 3 and 4, and on the other hand on the existence of universal type-0 grammars and on the possibility of commuting from an H system to a type-0 grammar and conversely. This reduces the problem of the existence of universal H systems to the existence of universal Chomsky grammars (or Turing Machines). However, this quite indirect way, while theoretically useful, is inconvenient from a practical point of view. The *open problem* that remains is the effective construction of a universal H system that is as simple as possible. As the task seems to be a difficult one, it is perhaps better to look for a construction which at the same time meets the practical requirements raised by a possible implementation of such a universal H system. In short, we leave this task to a joint team of language theorists and practitioners of DNA computing.

Acknowledgments

Very useful discusssions with Arto Salomaa, Gabriel Thierrin, Helmut Jürgensen, Tom Head, and Dennis Pixton are gratefully acknowledged. Thanks are also due to two anonymous referees for many remarks about an earlier version of this paper, allowing us to improve the readability of the text significantly.

References

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

L. M. Adleman, On constructing a molecular computer, in *DNA Based Computers* (R. J. Lipton and E. B. Baum, eds.), Proc. of a DIMACS Workshop, Princeton, 1995, American Mathematical Society, Providence, RI, 1996, pp. 67–74.

C. Calude and Gh. Păun, Global syntax and semantics for recursively enumerable languages, *Fundamenta Informaticae*, 4(2) (1981), 245–254.

K. Culik II and T. Harju, Splicing semigroups of dominoes and DNA, *Discrete Applied Mathematics*, 31 (1991), 261–277.

J. Dassow and Gh. Păun, *Regulated Rewriting in Formal Language Theory*, Springer-Verlag, Berlin, 1989.

L. Davis, *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, New York, 1991.

K. L. Denninghoff and R. W. Gatterdam, On the undecidability of splicing systems, *International Journal of Computer Mathematics*, 27 (1989), 133–145.

S. Eilenberg, *Automata, Languages and Machines*, Vol. A, Academic Press, New York, 1974.

D. K. Gifford, On the path to computation with DNA, *Science*, 226 (Nov. 1994), 993–994.

D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, Reading, MA, 1989.

J. Hartmanis, On the weight of computation, *Bulletin of the EATCS*, 55 (1995), 136–138.

M. A. Harrison, *Introduction to Formal Language Theory*, Addison-Wesley, Reading, MA, 1978.

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

T. Head, Gh. Păun, and D. Pixton, Language theory and molecular genetics. Generative mechanisms suggested by DNA recombination, in *Handbook of Formal Languages*, Vol. 2 (G. Rozenberg and A. Salomaa, eds.), Springer-Verlag, Berlin, 1997, pp. 295–360.

J. Hertz, A. Krogh, and R. G. Palmer, *Introduction to the Theory of Neural Computation*, Addison-Wesley, Reading, MA, 1991.

L. Hunter, Molecular biology for computer scientists, in *Artificial Intelligence and Molecular Biology* (L. Hunter, ed.), AAAI Press/ MIT Press, Menlo Park, CA, 1993, pp. 1–46.

J. H. Koza and J. P. Rice, *Genetic Algorithms: The Movie*, MIT Press, Cambridge, MA, 1992.

W.-H. Li and D. Graur, *Fundamentals of Molecular Evolution*, Sinauer, Sunderland, MA, 1991.

R. J. Lipton, Speeding up computations via molecular biology, in *DNA Based Computers* (R. J. Lipton and E. B. Baum, eds.), Proc. of a DIMACS Workshop, Princeton, 1995, American Mathematical Society, Providence, RI, 1996, pp. 67–74.

R. J. Lipton, DNA solution of hard computational problems, *Science*, 268 (April 1995), 542–545.

A. Mateescu, Gh. Păun, G. Rozenberg, and A. Salomaa, Simple splicing systems, Technical Report 95–09, Department of Computer Science, Leiden University, 1995, and *Discrete Applied Mathematics*, 84 (1998), 145–163.

M. L. Minsky, Recursive unsolvability of Post's problem of "tag" and other topics in the theory of Turing machines, *Annals of Mathematics*, 74(3) (1961), 437–455.

Gh. Păun, Splicing. A challenge for formal language theorists, *Bulletin of the EATCS*, 57 (1995a), 183–194.

Gh. Păun, On the power of the splicing operation, *International Journal of Computer Mathematics*, 59 (1995b), 27–35.

Gh. Păun, On the splicing operation, *Discrete Applied Mathematics*, 70 (1996a), 57–79.

Gh. Păun, Regular extended H systems are computationally universal, *Journal of Automata, Languages, and Combinatorics*, 1(1) (1996b), 27–36.

Gh. Păun, G. Rozenberg, and A. Salomaa, Computing by splicing, *Theoretical Computer Science*, 168(2) (1996), 321–336.

Gh. Păun and A. Salomaa, DNA computing based on the splicing operation, *Mathematica Japonica*, 43(3) (1996), 17–32.

D. Pixton, Linear and circular splicing systems, *Proc. 1st IEEE Internat. Symp. on Intelligence in Neural and Biological Systems*, Herndon, 1995, pp. 38–45.

D. Pixton, Regularity of splicing languages, *Discrete Applied Mathematics*, 69 (1996), 101–124.

G. Rozenberg and A. Salomaa, *The Mathematical Theory of L Systems*, Academic Press, New York, 1980.

G. Rozenberg and A. Salomaa, *Handbook of Formal Languages*, Springer-Verlag, Berlin, 1997.

A. M. Turing, On computable numbers, with an application to the Entscheidungsproblem, *Proceedings of the London Mathematical Society*, Ser. 2, 42 (1936), 230–265; a correction, 43 (1936), 544–546.

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

D. B. Searls, The computational linguistics of biological sequences, in *Artificial Intelligence and Molecular Biology* (L. Hunter, ed.), AAAI Press/MIT Press, Menlo Park, CA, 1993, pp. 47–120.

A. P. J. van der Walt, Random context grammars, *Proc. IFIP Congress* 1970, North-Holland, Amsterdam, 1971, pp. 66–68.

T. Yokomori and S. Kobayashi, DNA evolutionary linguistics and RNA structure modelling: a computational approach, *Proc.* 1*st IEEE Internat. Symp. on Intelligence in Neural and Biological Systems*, Herndon, 1995, pp. 38–45.

Y. Zhang and N. C. Seeman, A solid-support methodology for the construction of geometrical objects from DNA, *Journal of the American Chemical Society*, 114 (1992), 1656–1663.

Received August 1997, *and in final form March* 1998.