Finite H-Systems with 3 Test Tubes are not Predictable

Lutz Priese
Computer Science Department, University of Koblenz, Germany,
priese@uni-koblenz.de

Yurii Rogojine
Academy of Sciences of Moldova, Kishinev, Moldova,
rogozhin@cc.acad.md

Maurice Margenstern
Institut Universitaire de Technologie, Metz, France,
margens@iut.univ-metz.fr

Finite H-systems with n test tubes are splicing systems of n test tubes over a common molecular alphabet, Σ, with a filter $F_i \subseteq \Sigma$ for each test tube. Initially, arbitrary many copies of molecules and enzymes (splicing rules) from a finite set of molecules and enzymes are given to the test tubes that produce new molecules by splicing and filtering. It is known that any formal language can be generated by a finite H-system with 9 test tubes and that the results of finite H-systems with 6 test tubes are unpredictable. Here we present a rather simple proof that the results of finite H-systems with only 3 test tubes are unpredictable and that 4 test tubes suffices to generate any formal language.

1 Introduction

Molecular computers have been attracting many people from chemistry, biology and computer science. A major break through was a concrete molecular computer by Adleman[1] that could solve instances of the travelling-salesman-problem from OR. In a remarkable paper Head[5] draw the connections between molecular computers and formal language theory. Head presented molecules as words over some alphabet and enzymes as so-called splicing rules. A splicing rule may be applicable to two molecules. It breaks both molecules at fixed locations, defined by the splicing rule, and recombines the initial string of one broken molecule with the final string of the second. Head's smooth connection to formal language theory brought this field to the attention of many people from formal language theory. E.g., Păun, Rozenberg and Salomaa[6] asked what classes of formal languages are derivable by molecular computers where initial molecules and enzymes from certain classes in formal language theory. One of such results says that any regular language is derivable from finitely many initial molecules with finitely many splicing rules. Csuhaj-Varjù, Kari, Păun[3] modified Head's concept slightly to systems of n test tubes. Here, any test tube is an H-system with an additional filter. In a single macro-step any test tube generates new molecules according to its set of starting molecules and its set of splicing rules. Afterwards, the outcome of all test tubes is poured into the filters of all test tubes. Those molecules that may pass the filter of test tubes $i, 1 \leq i \leq n$, form the new starting molecules for the i-th test tube for the next macro-step.

This new process, filtering results of one test tube into another, increases the computational capability of molecular computers. Let us call a system of n test tubes *finite* if initially any test tube contains (arbitrarily many) copies of molecules from a finite set of molecules and possesses only finitely many splicing rules. It is known [6,1] that a finite 1-test-tube-system generates only regular sets of molecules. However, finite 2-test-tube-system may generate more complicated non-regular sets [3]. Ferretti, Mauri, Zandron[4] have shown that any recursively enumerable (r.e.) set of molecules is derivable in a finite 9-test-tube-system (or in a finite 6-test-tube-system if one allows for a rather simple encoding of the molecules to be generated). These results have implications for molecular computers as r.e. languages have many undecidable properties. E.g., the membership problem is in general not decidable for r.e. languages. This means that there exists no algorithm $\mathcal{A}$ which can tell, when presenting a word w and an r.e. languages L to $\mathcal{A}$, whether w belongs to L or not. Further, there is a fixed language, U, such there exists no algorithms $\mathcal{A}$ which can tell, when presenting a word w to $\mathcal{A}$, whether w is an element of U or not. Thus, a trivial consequence of the result of[4] is that there exists no algorithm which can compute which molecules may be generated in a finite 6-test-tube-system. I.e., the results of a finite 6-test-tube-system cannot be algorithmically predicted in general. We will sharpen this result here by showing how to generate any r.e. language in a finite 3-test-tube-system. Thus, there is no way to predict the outcome of the reactions of only three test tubes starting with molecules and enzymes from finite set of molecules and enzymes.

2 Notations and Basic Concept

We use the following standard notations from formal language theory.

An *alphabet* is a finite, non-empty set whose elements are also called *letters*. A *word* (over some alphabet Σ) is a finite (possibly empty) concatenation of occurrences of letters (from Σ). The empty concatenation of letters is also called the *empty word* and is denoted by ε. Σ^* denotes the set of all words over Σ. A *language* (over Σ) is a set of words (over Σ).

A *formal grammar* G is a tuple $G = (N, T, R, S)$ of an alphabet N of so-called *non-terminal* letters, an alphabet T of so-called *terminal* letters, with

$N \cap T = \emptyset$, an *initial letter* S from N, and a finite set R of *rules* of the form $u \to v$ with $u, v \in (N \cup T)^*$. Any rule $u \to v \in R$ is a substitution rule allowing to substitute any occurrence of u in some word w by v.

Formally, we write $w \Rightarrow_G w'$ if there exist a rule $u \to v$ in R and words $w_1, w_2 \in (N \cup T)^*$ with $w = w_1 u w_2$ and $w' = w_1 v w_2$. $\Rightarrow_G^*$ denotes the reflexive and transitive closure of $\Rightarrow$. I.e., $w \Rightarrow_G^* w'$ means that there exists an integer n and words $w_1, \cdots, w_n$ with $w = w_1, w' = w_n$ and $w_i \Rightarrow_G w_{i+1}$ for all $i, 1 \leq i < n$. $n = 1$ is allowed, thus $w \Rightarrow_G^* w$ holds always. We often drop the index G and write $\Rightarrow$ instead of $\Rightarrow_G$ if G is clear from the context or not important.

The sequence $w_1 \Rightarrow w_2 \Rightarrow \cdots \Rightarrow w_n$ is also called a computation (from w_1 to w_n of length $n-1$), or similar. A *terminal word* is a word in T^*; all terminal words computable from the initial letter S form the language $L(G)$ generated by G. More formally, $L(G) := \{w \in T^*;\ S \Rightarrow^* w\}$. A language L is called *recursively enumerable*, or simply *r.e.*, if there exists some Turing machine generating L. It is a fundamental result of computer science that a language L is r.e. if and only if there exists some formal grammar G with $L = L(G)$.

An *(abstract) molecule* is in this paper simply a word over some alphabet. An *abstract enzyme*, or *splicing rule*, is a quadruple (u_1, u_2, u_1', u_2') of words, which is often written in a two dimensional way as $\frac{u_1 \mid u_2}{u_1' \mid u_2'}$.

A splicing rule $r = (u_1, u_2, u_1', u_2')$ is applicable to two molecules m_1, m_2 if there are words w_1, w_2, w_1', w_2' with $m_1 = w_1 u_1 u_2 w_2$ and $m_2 = w_1' u_1' u_2' w_2'$, and produces two new molecules $m_1' = w_1 u_1 u_2' w_2'$ and $m_2' = w_1' u_1' u_2 w_2$. In this case, we also write $\{m_1, m_2\} \vdash_r \{m_1', m_2'\}$. We also write $w \vdash_r w'$ if there exist w_1 and w_1' with $\{w, w_1\} \vdash_r \{w', w_1'\}$.

A *Head-splicing-system*, or *H-system*, is a triple $H = (\Sigma, M, E)$ of an alphabet Σ, a set $M \subseteq \Sigma^*$ of initial molecules over Σ, and a set $E \subseteq \Sigma^* \times \Sigma^* \times \Sigma^* \times \Sigma^*$ of splicing rules. H is called finite if M and E are finite sets. For any set $L \subseteq \Sigma^*$ of molecules we denote by $\sigma_H(L) := \{w \in \Sigma^*;\ \exists w_1, w_2 \in L : \exists w' \in \Sigma^* : \exists r \in E : \{w_1, w_2\} \vdash_r \{w, w'\}\}$ the set of all molecules derivable from L by one application of a splicing rule. Further, $\sigma_H^0(L) := L$, $\sigma_H^{i+1}(L) := \sigma_H^i(L) \cup \sigma_H(\sigma_H^i(L))$, $\sigma_H^*(L) := \bigcup_{i \geq 0} \sigma_H^i(L)$, $\sigma(H) := \sigma_H^*(M)$.

Thus, $\sigma(H)$ is the set of all molecules that can be generated in H starting with M as initial molecules by iteratively applying splicing rules to copies of the molecules already generated.

A *test tube* T is a tuple $T = (H, F)$ of an H-system $H = (\Sigma, M, E)$ and an alphabet $F \subseteq \Sigma$, called the *filter* for T or H.

An *H-system with n test tubes*, or simply an *n-tt*, H, is a tuple $H =$

$(\Sigma, T_1, \cdots, T_n)$ of an alphabet Σ and n test tubes $T_i = ((\Sigma, M_i, E_i), F_i)$, $1 \leq i \leq n$.

For languages $L_i, L_i', 1 \leq i \leq n$, over Σ one writes $(L_1, \cdots, L_n) \vdash_H (L_1', \cdots, L_n')$ if
$L_i' = \bigcup_{j=1}^{n} \left(\sigma_{T_j}^*(L_j) \cap F_i^*\right) \cup \left(\sigma_{T_i}^*(L_i) \cap \left(\Sigma^* - \bigcup_{j=1}^{n} F_j^*\right)\right)$
holds for $1 \leq i \leq n$.

I.e., to get L_i' one generates all results $\sigma_{T_j}^*(L_j)$ of all H-systems T_j starting with L_j as initial molecules and puts all those results $\sigma_{T_j}^*(L_j)$ into L_i' that pass the filter F_i. In addition, one keeps in L_i' all those molecules produced in T_i from L_i(i.e., $\sigma_{T_i}^*(L_i)$) that cannot pass any filter F_j into a further H-system T_j.

Again, $\vdash_H^*$ denotes the reflexive and transitive closure of $\vdash_H$.

The *result* $\rho(H)$ of H is all the possible contents of its first test tube. More formally,

$\rho(H) := \{L_1 \subseteq \Sigma^*; \exists L_2, \cdots, L_n \subseteq \Sigma^*\colon (M_1, \cdots, M_n) \vdash_H^* (L_1, \cdots, L_n)\}$
are all molecules derivable in the first test tube T_1 if one starts with the initial molecules M_i in T_i, $1 \leq i \leq n$.

An *extended n-tt* H is a tuple $H = (\Sigma, T_1, \cdots, T_n, ,)$ of an *n-tt* $(\Sigma, T_1, \cdots, T_n)$ and a *terminal* alphabet , $\subseteq \Sigma$. The *result* $\rho(H)$ of an extended *n-tt* is defined as $\rho(H) := \rho((\Sigma, T_1, \cdots, T_n)) \cap , ^*$, the set of all molecules over , * derivable in the first test tube.

3 An Example

A grammar $G = (N, T, R, S)$ is called *right-linear* if all of its rules $u \rightarrow v$ in R are of the form $x \rightarrow ay$, $x \rightarrow a$ or $x \rightarrow \epsilon$ with $x, y \in N$ and $a \in T$. Thus, any computation has the form $S \Rightarrow a_1 x_1 \Rightarrow a_1 a_2 x_2 \Rightarrow \cdots \Rightarrow a_1 a_2 \cdots a_n x_n \Rightarrow a_1 a_2 \cdots a_{n(+1)}$ with some $a_i \in T$, $x_i \in N$.

A language L is called *regular* if there exists a right-linear grammar G that generates L, i.e. $L = L(G)$. Obviously, regular languages are generated by computations of a very special form: there is never a substitution within a word but only at the right end of words. But such a "substitution at the right end" is easily expressed as a splicing: $wx \Rightarrow_G way$ is the result of breaking wx and a special word Zay before x and a, respectively, and recombining them into way and Zx.

Thus, the right-linear rule $x \rightarrow ay$ becomes the splicing rule $\dfrac{\varepsilon \mid x}{Z \mid ay}$.

The following right-linear grammar G_0 generates all words over $\{a, b\}$ with an even number of occurrences of the letter a and a number of occurrences of the letter b that can be divided by 3 : $G_0 = (N_0, T_0, R_0, S)$ with

- $N_0 := \{S, x_{0,0}, x_{0,1}, x_{0,2}, x_{1,0}, x_{1,1}, x_{1,2}\}$,
- $T_0 := \{a, b\}$
- R_0 is given by the rules

$$\begin{array}{llll}
S \rightarrow \varepsilon & x_{0,0} \rightarrow ax_{1,0} & x_{0,2} \rightarrow ax_{1,2} & x_{1,1} \rightarrow ax_{0,1} \\
S \rightarrow ax_{1,0} & x_{0,0} \rightarrow bx_{0,1} & x_{0,2} \rightarrow bx_{0,0} & x_{1,1} \rightarrow bx_{1,2} \\
S \rightarrow bx_{0,1} & x_{0,1} \rightarrow ax_{1,1} & x_{1,0} \rightarrow ax_{0,0} & x_{1,2} \rightarrow ax_{0,2} \\
x_{0,0} \rightarrow \varepsilon & x_{0,1} \rightarrow bx_{0,2} & x_{1,0} \rightarrow bx_{1,1} & x_{1,2} \rightarrow bx_{1,0}
\end{array}$$

An example of a computation in G_0 is
$S \Rightarrow ax_{1,0} \Rightarrow abx_{1,1} \Rightarrow abbx_{1,2} \Rightarrow abbax_{0,2} \Rightarrow abbabx_{0,0} \Rightarrow abbab$.

Obviously, this can easily be simulated by splicings. We regard the H-system $H = (\Sigma, M, E)$ with
– $\Sigma := N \cup T \cup \{Z\}$, with a new letter $Z \notin N \cup T$,
– $M := \{S,\ ZZ\} \cup \{Zax_{i,j};\ 0 \leq i \leq 1,\ 0 \leq j \leq 2\}$
$\cup \{Zbx_{i,j};\ 0 \leq i \leq 1,\ 0 \leq j \leq 2\}$,
– E is the following list of splicing rules

$$1: \begin{array}{c|c} \varepsilon & S \\ \hline ZZ & \varepsilon \end{array} \quad 2: \begin{array}{c|c} \varepsilon & S \\ \hline Z & ax_{1,0} \end{array} \quad 3: \begin{array}{c|c} \varepsilon & S \\ \hline Z & bx_{0,1} \end{array}$$

$$4: \begin{array}{c|c} \varepsilon & x_{0,0} \\ \hline ZZ & \varepsilon \end{array} \quad 5: \begin{array}{c|c} \varepsilon & x_{0,0} \\ \hline Z & ax_{1,0} \end{array} \quad 6: \begin{array}{c|c} \varepsilon & x_{0,0} \\ \hline Z & bx_{0,1} \end{array}$$

$$7: \begin{array}{c|c} \varepsilon & x_{0,1} \\ \hline Z & ax_{1,1} \end{array} \quad 8: \begin{array}{c|c} \varepsilon & x_{0,1} \\ \hline Z & bx_{0,2} \end{array} \quad 9: \begin{array}{c|c} \varepsilon & x_{0,2} \\ \hline Z & ax_{1,2} \end{array} \quad 10: \begin{array}{c|c} \varepsilon & x_{0,2} \\ \hline Z & bx_{0,0} \end{array}$$

$$11: \begin{array}{c|c} \varepsilon & x_{1,0} \\ \hline Z & ax_{0,0} \end{array} \quad 12: \begin{array}{c|c} \varepsilon & x_{1,0} \\ \hline Z & bx_{1,1} \end{array} \quad 13: \begin{array}{c|c} \varepsilon & x_{1,1} \\ \hline Z & ax_{0,1} \end{array} \quad 14: \begin{array}{c|c} \varepsilon & x_{1,1} \\ \hline Z & bx_{1,2} \end{array}$$

$$15: \begin{array}{c|c} \varepsilon & x_{1,2} \\ \hline Z & ax_{0,2} \end{array} \quad 16: \begin{array}{c|c} \varepsilon & x_{1,2} \\ \hline Z & bx_{1,0} \end{array}$$

One now easily simulates any computation of G_0. Let us regard the above example. How to simulate $S \Rightarrow ax_{1,0}$? In H we have the molecules S and $Zax_{1,0}$ and can apply splicing rule 2 to get $\{S, Zax_{1,0}\} \vdash_2 \{ZS, ax_{1,0}\}$, resulting in the new wanted molecule $ax_{1,0}$ plus some garbage ZS. We may continue to apply splicing rule 12 to the two molecules $ax_{1,0}$ and $Zbx_{1,1}$ to get $\{ax_{1,0}, Zbx_{1,1}\} \vdash_{12} \{Zx_{1,0}, abx_{1,1}\}$ with $abx_{1,1}$ plus some garbage $Zx_{1,0}$. One easily checks that $S \vdash_2 ax_{1,0} \vdash_{12} abx_{1,1} \vdash_{14} abbx_{1,2} \vdash_{15} abbax_{0,2} \vdash_{10} abbabx_{0,0} \vdash_4 abbab$ is a possible chain of reactions in H. The role of Z is to handle the garbage, some unwanted results. Suppose we operate without Z. Thus, rule 10 would have the form $\begin{array}{c|c} \varepsilon & x_{0,2} \\ \hline \varepsilon & bx_{0,0} \end{array}$. We might apply this rule 10' to $abbabx_{0,0}$ and the garbage $x_{0,2}$ (instead of $Zx_{0,2}$) resulting in a 'backward' computation $\{abbabx_{0,0}, x_{0,2}\} \vdash_{10'} \{abbax_{0,2}, bx_{0,0}\}$.

Those 'backward' computations are harmless for G_0, as G_0 has a so-called reversibility property (G_0 is "backward deterministic"). However, in general backward computations lead to disaster and are therefore eliminated by the help of the special symbol Z. (It might be mentioned that Bennett[2] has shown how to simulate any deterministic Turing machine by one which is additionally backward deterministic. But such a discussion on 'reversible' computations would leave the scope of this paper.)

With the technique of the example one easily proves that any regular language is the result of an extended *1-tt*; more results on this subject are found in [6].

4 *3-TT* Simulate any Grammar

In contrast to right-linear grammars a rule $u \to v$ of a formal grammar defines in general a substitution inside a word; $w_1uw_2 \Rightarrow w_1vw_2$. Whilst splicing rules trivially simulate the rules of right-linear grammars they cannot directly simulate a general substitution. In [4] and [3] a method is introduced to rotate words w_1uw_2 into w_2w_1u and apply the substitution solely at the end of a word, just as it is done in right-linear grammars. A further letter, B, marks the correct beginning of a word. Thus, w_2Bw_1u is a rotated version of Bw_1uw_2. For technical reasons, two further letters, X, Y are required that mark the first and final letter of all rotated words. Thus, a *representation* of a word $w \in (N \cup T)^*$ is any word of the form Xw_2Bw_1Y with $X, B, Y \notin N \cup T$ and $w = w_1w_2$. For any grammar $G = (N, T, R, S)$ we shall now design a *3-tt* such that for any word $w \in (N \cup T)^*$ with $S \Rightarrow_G^* w$ one finds all representations Xw_2Bw_1Y of w in test tube 1. Here, we follow quite closely the ideas in [4] and [3]. A rule $u \to v$ from R applicable to w_1uw_2 is simulated by a splicing rule $\frac{\varepsilon \mid uY}{Z \mid vY}$ applicable to the representation Xw_2Bw_1uY of w_1uw_2. To ensure that all representations of a derivable word w from S in G are found in test tube 1 we have to rotate the words between X and Y. This is done with the help of two more test tubes, 2 and 3, and an encoding of the letters in $N \cup T \cup \{B\}$. Let $N \cup T \cup \{B\} = \{l_1, \cdots, l_n\}$. We encode l_i as $\beta\alpha^i\beta$, where α^i is a sequence of i ocurrences of α. α and β are new letters. A splicing rule $\frac{\varepsilon \mid l_iY}{Z \mid \beta\alpha^i\beta Y}$ encodes the final letter l_i before Y into $\beta\alpha^i\beta$. If the new final letter before Y is β (or α), we change Y into Y_β (or Y_α) and delete this β (or α). No further reactions with the letter Y_α and Y_β are possible in test tube 1.

Words with a letter Y_β (or Y_α) may pass only the filter of test tube 2 (or test tube 3), where a new letter β (or α) is added as first letter behind X. If a complete encoding $\beta\alpha^i\beta$ is thus transformed from the end of a word to the beginning (behind X), a further splicing rule decodes $\beta\alpha^i\beta$ back into l_i: $\frac{X\beta\alpha^i\beta \mid \varepsilon}{Xl_i \mid Z}$.

Thus, we associate with any formal grammar $G = (N, T, R, S)$ the following *3-tt*, H_G:

$H_G = (\Sigma, T_1, T_2, T_3)$ with

- $\Sigma = N \cup T \cup \{B, X, Y, \alpha, \beta, X', Y_\alpha, Y_\beta\}$
- $T_1 = (M_1, E_1, F_1)$ with

  $F_1 = N \cup T \cup \{B, X, Y, \alpha, \beta\}$

  $M_1 = \{XSBY, XBSY, ZY_\alpha, ZY_\beta, X'Z\} \cup \{Z\beta\alpha^i\beta Y;\ 1 \le i \le n\} \cup \{ZvY;\ \exists u : u \to v \in R\} \cup \{Xl_iZ;\ 1 \le i \le n\}$ where $N \cup T \cup \{B\} = \{l_1, \cdots, l_n\}$, and

  E_1 consists of the following splicing rules,

  1: $\frac{\varepsilon \mid uY}{Z \mid vY}$ for $u \to v \in R$, $2: \frac{\varepsilon \mid l_iY}{Z \mid \beta\alpha^i\beta Y}$, $1 \le i \le n$,

  3: $\frac{\varepsilon \mid \beta Y}{Z \mid Y_\beta}$, $4: \frac{\varepsilon \mid \alpha Y}{Z \mid Y_\alpha}$, $5: \frac{X \mid \varepsilon}{X' \mid Z}$,

  6: $\frac{X\beta\alpha^i\beta \mid \varepsilon}{Xl_i \mid Z}$, $1 \le i \le n$,

- $T_2 = (M_2, E_2, F_2)$ with

  $F_2 = N \cup T \cup \{B, \alpha, \beta, X', Y_\beta\}$,

  $M_2 = \{ZY, X\beta Z\}$, and

  E_2 consists of the following splicing rules:

  7: $\frac{\varepsilon \mid Y_\beta}{Z \mid Y}$, $8: \frac{X' \mid \varepsilon}{X\beta \mid Z}$,

- $T_3 = (M_3, E_3, F_3)$ with

  $F_3 = N \cup T \cup \{B, \alpha, \beta, X', Y_\alpha\}$,

  $M_3 = \{ZY, X\alpha Z\}$, and

  E_3 consists of

  9: $\frac{\varepsilon \mid Y_\alpha}{Z \mid Y}$, $10: \frac{X' \mid \varepsilon}{X\alpha \mid Z}$.

Suppose $S \Rightarrow_G^* w_1uw_2 \Rightarrow_G w_1vw_2$ holds with $u \to v \in R$. In test tube 1 we have $XBSY$ as a molecule in M_1. Suppose that we have already generated all representations of w_1uw_2 in test tube 1. Thus, also Xw_2Bw_1uY is in test tube 1 and the following chain of reactions is valid:
test tube 1:
$\{Xw_2Bw_1uY, ZvY\} \vdash_1 \{Xw_2Bw_1vY, ZuY\}$,
let $v = v'l_i$ for some v', l_i; so we continue
$\{Xw_2Bw_1v'l_iY, Z\beta\alpha^i\beta Y\} \vdash_2 \{Xw_2Bw_1v'\beta\alpha^i\beta Y, Zl_iY\}$,
$\{Xw_2Bw_1v'\beta\alpha^i\beta Y, ZY_\beta\} \vdash_3 \{Xw_2Bw_1v'\beta\alpha^iY_\beta, Z\beta Y\}$
$\{Xw_2Bw_1v'\beta\alpha^iY_\beta, X'Z\} \vdash_5 \{X'w_2Bw_1v'\beta\alpha^iY_\beta, XZ\}$
test tube 2:
$\{X'w_2Bw_1v'\beta\alpha^iY_\beta, ZY\} \vdash_7 \{X'w_2Bw_1v'\beta\alpha^iY, ZY_\beta\}$
$\{X'w_2Bw_1v'\beta\alpha^iY, X\beta Z\} \vdash_8 \{X\beta w_2Bw_1v'\beta\alpha^iY, X'Z\}$
test tube 1:
$\{X\beta w_2Bw_1v'\beta\alpha^{i-1}\alpha Y, ZY_\alpha\} \vdash_4 \{X\beta w_2Bw_1v'\beta\alpha^{i-1}Y_\alpha, Z\alpha Y\}$
$\{X\beta w_2Bw_1v'\beta\alpha^{i-1}Y_\alpha, X'Z\} \vdash_5 \{X'\beta w_2Bw_1v'\beta\alpha^{i-1}Y_\alpha, XZ\}$
test tube 3:
$\{X'\beta w_2Bw_1v'\beta\alpha^{i-1}Y_\alpha, ZY\} \vdash_9 \{X'\beta w_2Bw_1v'\beta\alpha^{i-1}Y, ZY_\alpha\}$
$\{X'\beta w_2Bw_1v'\beta\alpha^{i-1}Y, X\alpha Z\} \vdash_{10} \{X\alpha\beta w_2Bw_1v'\beta\alpha^{i-1}Y, X'Z\}$

Continuing this way, we finally get $X\beta\alpha^i\beta w_2Bw_1v'Y$ in test tube 1, resulting in $\{X\beta\alpha^i\beta w_2Bw_1v'Y, Xl_iZ\} \vdash_6 \{Xl_iw_2Bw_1v'Y, X\beta\alpha^i\beta Z\}$, where the final letter l_i before Y is now rotated to be the first letter behind X. Using the same technique, we easily get any representation of w_1vw_2 in test tube 1. We also can describe the results of all three test tubes as follows. Let $c : \{l_1, \cdots, l_n\}^* \to 2^{\{l_1,\cdots,l_n,\alpha,\beta\}^*}$ – here 2^M denotes the set of all subsets of M – be defined by $c(l_i) := \{l_i, \beta\alpha^i\beta\}$ and $c(w_1w_2) := c(w_1)c(w_2)$. I.e., $l_2\beta\alpha\alpha\alpha\beta l_1\beta\alpha\beta \in c(l_2l_3l_1l_1)$. Further, $\rho : \{l_1, \cdots, l_n, \alpha, \beta\}^* \to 2^{\{l_1,\cdots,l_n,\alpha,\beta\}^*}$ is defined by $w_2w_1 \in \rho(w)$ if and only if $w = w_1w_2$, for some w_1, w_2, and $\tilde{c} := \rho \circ c$. I.e., $\alpha\beta l_1\beta\alpha\beta l_2\beta\alpha\alpha \in \tilde{c}(l_2l_3l_1l_1)$. Define

$$C_1 : = \{\mathcal{A}u\Omega;\ \mathcal{A} \in \{X, X'\}\ \&\ ((\Omega = Y\ \&\ \exists w : u \in \tilde{c}(Bw)\ \&\ S \Rightarrow^* w) \vee (\Omega = Y_\alpha\ \&\ \exists w : u\alpha \in \tilde{c}(Bw)\ \&\ S \Rightarrow^* w) \vee (\Omega = Y_\beta\ \&\ \exists w : u\beta \in \tilde{c}(Bw)\ \&\ S \Rightarrow^* w))\},$$

$$C_2 : = \{\mathcal{A}u\Omega;\ \mathcal{A} \in \{X', X\beta\}\ \&\ \Omega \in \{Y, Y_\beta\}\ \&\ \exists w : u\beta \in \tilde{c}(Bw)\ \&\ S \Rightarrow^* w\},$$

$$C_3 : = \{\mathcal{A}u\Omega;\ \mathcal{A} \in \{X', X\alpha\}\ \&\ \Omega \in \{Y, Y_\alpha\}\ \&\ \exists w : u\alpha \in \tilde{c}(Bw)\ \&\ S \Rightarrow^* w\}.$$

Further

$$G_1 : = \{ZuY;\ \exists v : u \to v \in R \vee v \to u \in R\} \cup \{Z\beta\alpha^i\beta Y,\ X\beta\alpha^i\beta Z,\ Zl_iY,\ Xl_iZ;\ 1 \le i \le n\} \cup \{Z\beta Y,\ ZY_\beta,\ Z\alpha Y,\ ZY_\alpha\ XZ,\ X'Z\},$$

$G_2 : = \{ZY,\, ZY_\beta,\, X\beta Z,\, X'Z\}$, and

$G_3 : = \{ZY,\, ZY_\alpha,\, X\alpha Z,\, X'Z\}$.

Then one easily proofs that the result ρ_i in test tube i is exactly $C_i \cup G_i$, $1 \leq i \leq 3$. Here, G_i denotes the "garbage" and C_i the wanted contents. To prove $\rho_i \supseteq C_i \cup G_i$ one proceeds as above: for any word in $C_i \cup G_i$ one inductively finds a chain of reactions producing this word. For "$\subseteq$" one simply notes that an application of a splicing rule in test tube i to two words from $C_i \cup G_i$ again results in two words in $C_i \cup G_i$. Thus, if we regard only the 'uncoded' words – i.e., $c(l_i) = l_i$ – we have already shown:

Theorem 1 *For any words w_1, $w_2 \in (N \cup T)^*$ there holds: H_G can produce Xw_2Bw_1Y in test tube 1 if and only if $S \Rightarrow_G^* w_1w_2$ holds.*

We say that a class $\mathcal{C}$ of *n-tts* is *predictable* if there exists an algorithm $\mathcal{A}$ which tells, given some *n-tt* H from $\mathcal{C}$ and some word w over the alphabet Σ of H as inputs to $\mathcal{A}$, whether w can be generated in the first test tube of H or not. Suppose now that the class of all *3-tts* is predictable. Then we could decide the membership problem of any grammar: Given $G = (N, T, R, S)$ and $w \in T^*$, consider H_G and $XBwY$. Test with the help of $\mathcal{A}$ if H_G can produce $XBwY$ in test tube 1. If yes, than $S \Rightarrow_G^* w$ holds, if not, than $S \Rightarrow_G^* w$ is false. As the membership problem for general formal grammars is undecidable, we conclude:

Corollary: Finite H-splicing-systems with 3 test tubes are not predictable.

5 Extended *3-tt* Can Generate Any 'Pure' Formal Language

An extended *n-tt* $(H, ,)$ generates a given language L if $L = \rho(H) \cap ,^*$ holds. Let G be a formal grammar, $L = L(G)$, then $\rho(H_G) \cap ,^* \neq L$ for our *3-tt* H_G from the previous chapter. We 'only' proved that $XBwY \in \rho(H_G)$ for $w \in T^*$ if and only if $w \in L(G)$. To produce the 'pure' words $w \in L(G)$ in test tube 1 one may use an extended *3-tt*, H'_G, with T as the terminal alphabet (i.e. $H'_G = (H_G, T)$) and try to get rid of the symbols X, B, and Y. A standard idea from formal language theory is to produce words $XBwY$ with $w \in T^*$ or $w \in \Sigma^*$ and simply guess that $w \in T^*$ may hold. Now one guesses to drop XB and Y and results with the pure terminal words. One might try and introduce two new splicing rules for test tube 1:

$1' : \dfrac{XB \mid \varepsilon}{\varepsilon \mid ZZ}$ and $2' : \dfrac{\varepsilon \mid Y}{ZZ \mid \varepsilon}$.

However, this would lead to chaos as is seen as follows. Suppose we have generated $XBwY$ with $w \in T^*$, thus $w \in L$. Applying $1'$ to $XBwY$ results in wY. We now may use the rules of all three test tubes to delete the final letters of w before Y without producing them (in a rotated form) in front behind X (as there is no more X). Thus, we could derive any $w'Y$ with $w = w''w'$ for some w'. By rule $2'$ we can produce also w', although this suffix w' of w does not have to belong to $L(G)$.

However, we still may follow the idea to guess an end of reactions and to drop XB and Y. But this has to be done with a more involved method where Y can only be dropped after Y 'has told X to do the same'.

When we guess to drop XB and Y we first transform Y into $\gamma\beta\beta Y$, where γ is a new letter. $\beta\beta$ are now rotated to the beginning, as before. Thus, we get $XBwY \vdash XBw\gamma\beta\beta Y \vdash^* X'\beta\beta Bw\gamma Y$. As γ is new, there is no rule for γY in H_G. We simply drop γY. However, $X'\beta\beta Bw$ now may enter test tubes 2 and 3 and new letters α, β may be produced after X. Fortunately, this leads only to further garbage not in T^* as we will drop X and B only together in the form of $X\beta\beta B$ by a rule $\frac{X\beta\beta B \mid \varepsilon}{\varepsilon \mid ZZ}$. Note, $\beta\beta$ encodes no letter l_i, only $\beta\alpha^i\beta$ with $i > 0$ encodes a letter. $X\beta\beta B$ is thus a unique message for X to become deleted.

Let $G = (N, T, R, S)$ be any formal grammar. H'_G denotes the extended *3-tt* $H'_G = (\Sigma, T'_1, T'_2, T'_3, ,)$ with

- $, = T$,
- $\Sigma = N \cup T \cup \{B, X, X', Y, \alpha, \beta, \gamma, Y_\alpha, Y_\beta\}$,
- $T'_1 = (M'_1, E'_1, F'_1)$ with

  $F'_1 = N \cup T \cup \{B, X, Y, \alpha, \beta, \gamma\}$,
  $M'_1 = \{XSBY, XBSY, ZY_\alpha, ZY_\beta, X'Z, ZZ\} \cup \{Z\beta\alpha^i\beta Y;\ 1 \le i \le n\} \cup \{ZvY;\ \exists u : u \to v \in R\} \cup \{Xl_iZ;\ 1 \le i \le n\}$,
  where $N \cup T \cup \{B\} = \{l_1, \cdots, l_n\}$,
  E_1 consists of the following splicing rules:

  1: $\frac{\varepsilon \mid uY}{Z \mid vY}$, for $u \to v \in R$, $\;2: \frac{\varepsilon \mid l_iY}{Z \mid \beta\alpha^i\beta Y}$, $1 \le i \le n$,

  3: $\frac{\varepsilon \mid \beta Y}{Z \mid Y_\beta}$, $\;4: \frac{\varepsilon \mid \alpha Y}{Z \mid Y_\alpha}$, $\;5: \frac{X \mid \varepsilon}{X' \mid Z}$,

  6: $\frac{X\beta\alpha^i\beta \mid \varepsilon}{Xl_i \mid Z}$, $1 \le i \le n$,

  7: $\frac{\varepsilon \mid Y}{Z \mid \gamma\beta\beta Y}$, $\;8: \frac{\varepsilon \mid \gamma Y}{ZZ \mid \varepsilon}$, $\;9: \frac{X\beta\beta B \mid \varepsilon}{\varepsilon \mid ZZ}$

– $T_2' = (M_2', E_2', F_2')$ with

$F_2' = V \cup T \cup \{B, \alpha, \beta, \gamma, X', Y_\beta\}$,
$M_2' = \{ZY, X\beta Z\}$,
and E_2' consists of

$$10 : \frac{\varepsilon \mid Y_\beta}{Z \mid Y}, \quad 11 : \frac{X' \mid \varepsilon}{X\beta \mid Z}$$

– $T_3' = (M_3', E_3', F_3')$ with

$F_3' = V \cup T \cup \{B, \alpha, \beta, \gamma, X', Y_\beta\}$,
$M_3' = \{ZY, X\alpha Z\}$,
and E_3' consists of

$$12 : \frac{\varepsilon \mid Y_\alpha}{Z \mid Y}, \quad 13 : \frac{X' \mid \varepsilon}{X\alpha \mid Z}$$

Now, using a proof as for theorem 1 with slightly more complicated sets $C_i, G_i, 1 \leq i \leq 3$, one easily gets: For $w \in T^*$:
$w \in \rho(H_G')$ if and only if $XBwY \in \rho(H_G)$ if and only if $w \in L(G)$.
Thus, $L(G) = \rho(H_G')$. This proves:

Theorem 2 *Any r.e. language can be generated by an extended* 3-tt.

Extended *n-tts* $(\Sigma, T_1, \cdots, T_n, ,)$ possess a terminal alphabet, , . This can be dropped in H_G' if we use a further test tube T_0 with $T_0 = (\emptyset, \emptyset, T)$. T_0 filters all words over the terminal alphabet T. As a trivial consequence we know:

Corollary: Any r.e. language can be generated by a 4-tt.

6 Summary

We presented a rather simple proof that any r.e. language is generated by a finite 4-test-tube-system or by a finite extended 3-test-tube-system. Further, there exists no algorithm that predictes the outcome of finite 3-test-tube-system. This question is still open for finite 2-test-tube-systems.

Acknowledgments

Several fundings created the conditions of a fruitfull collaboration between the three authors, which made them possible to do this joint work. In this respect, the first and third authors thank the *Institut Universitaire de Technologie* of Metz, and the second and third authors thank both the French Ministery for Education, Universities and Research and again the *Institut Universitaire de Technologie* of Metz.

1. L. M. Adleman *Molecular computation of solutions of combinatorial problems*, Science, 226, pp. 1021-1024, 1994.
2. C.H. Bennett, *Logical Reversibility of Computation*, IBM J. Res. Develop. 6, pp. 525–532, 1973.
3. E. Csuhaj-Varjù, L. Kari, G. Păun, *Test Tube distributed system based on splicing*, Computer and AI, 2–3, pp. 211-232, 1996.
4. C. Ferretti, G. Mauri, C. Zandron *Nine Test Tubes Generate any RE Language*, personal communication.
5. T. Head *Formal Language Theory and DNA: An Analysis of the Generative Capacity of Specific Recombinant Behaviors*, Bulletin of Mathematical Biology, Vol. 49, No. 6, pp. 737-759, 1987.
6. G. Păun, G. Rozenberg, A. Saloma *Computing by splicing*, TCS 168, pp. 321–336, 1996.