Turing Computability and Membrane Computing

Yurii Rogozhin and Artiom Alhazov

Institute of Mathematics and Computer Science
Academy of Sciences of Moldova
Academiei 5, Chişinău MD-2028 Moldova
{rogozhin,artiom}@math.md

Abstract. Alan Turing began a new area in science; he discovered that there are universal computers, which in principal are very simple. Up to now this is the basis of a modern computing theory and practice. In the paper one considers Turing computability in the frame of P (membrane) systems and other distributive systems. An overview of the recent results about small universal P and DNA systems and some open problems and possible directions of investigation are presented.

1 Introduction

In the paper several very small universal computing devices inspired by molecular biology are presented. Alan Turing [43] discovered that there are universal computing devices, which in principal are very simple. Claude Shannon [42] suggested to find universal Turing machine of smallest size (he considered a descriptional complexity of universal programs). Current state of the art in solving Shannon's task is presented in [30]. In the paper one applies the Shannon's task to other computing models, especially to modern computing models inspired by molecular biology, namely for Membrane computing, DNA computing and some others computing models. Before proceeding, we outline selected small universal systems.

1.1 Selected Small Universal Systems

Turing Machines

We should mention references A. Turing [43]; C. Shannon [42]; M. Minsky [28]; R. Robinson [38]; M. Margenstern [21,22]; L. Pavlotskaya [31]; M. Margenstern and L. Pavlotskaya [23]; Yu. Rogozhin [39]; T. Neary and D. Woods [30]. The best known results are Turing machines with 24 rules, simulating Tag systems or cyclic Tag systems.

Circular Post Machines and Tag Systems

We should mention references E. Post [36]; J. Cocke and M. Minsky [9]; A. Alhazov, M. Kudlek and Yu. Rogozhin [5]; L. De Mol [29]; T. Neary and D.

E. Csuhaj-Varjú et al. (Eds.): CMC 2012, LNCS 7762, pp. 56–77, 2013.

Woods [30]; A. Alhazov, A. Krassovitskiy and Yu. Rogozhin [3]. Circular Post machines are a one-way variant of Turing machines that can insert and delete cells. Tag systems are a restricted model of Post normal systems, i.e., systems rewriting strings by removing a prefix and appending a suffix.

Besides being a tool for Turing machines, Tag systems have been used to obtain small universal devices in a number of splicing-based models, presented in this paper.

Circular Post machines have been used to obtain small universal Tag P systems and also small universal obligatory hybrid networks of evolutionary processors.

Cellular Automata

The most famous cellular automaton is the Conway's Game of Life. In two dimensions, cellular automata are known to be universal with two states, even with the von Neumann neighborhood (the center cell and 4 neighbours). In one dimension, there exists an intrinsically universal cellular automaton with 4 states. We are not discussing the elementary cellular automata in this paper, since the details of the universality notion already become a separate topic.

With more states, universal cellular automata have been obtained having radius-1/2 neighborhood (the center cell and only 1 neighbour), or having additional properties, such as number conservation, reversibility or symmetries.

Register Machines and Counter Automata

The smallest known universal register machines are constructed by I. Korec in 1996. The main result is a machine with 32 instructions (or 22 if the decrement and zero-test are counted as one instruction) and 8 registers. Its flowchart has 13 branchings.

This result has been used to obtain small universal spiking neural P systems and small universal P colonies, as well as a small universal antiport P system, presented in this paper. The latter is equivalent to maximally parallel multiset rewriting.

1.2 Computing Models Based on Splicing or Multiset Rewriting

Head splicing systems (H systems) [17] were one of the first theoretical models of biomolecular computing (DNA-computing). The molecules from biology are replaced by words over a finite alphabet and the chemical reactions are replaced by the *splicing* operation. An H system specifies a set of rules used to perform a splicing and a set of initial words or axioms. The computation is done by applying iteratively the rules to the set of words until no more new words can be generated. This corresponds to a bio-chemical experiment where one has

enzymes (splicing rules) and initial molecules (axioms) which are put together in a tube and one waits until the reaction stops.

From the formal language theory point of view, the computational power of the obtained model is rather limited, only regular languages can be generated. Various additional control mechanisms were proposed in order to "overcome" this obstacle and to generate all recursively enumerable languages. An overview of such mechanisms can be found in [34].

One of the goals of this work is to present several of small size universal systems based on splicing. Like in [40,6] we consider the number of rules as a measure of the size of the system. This approach is coherent with investigations related to small universal Turing machines, e.g. [39].

One of the first ideas to increase the computational power of splicing systems is to consider them in a distributed framework. Such a framework introduces test tubes, corresponding to H systems, which are arranged in a communicating network. The computation is then performed as a repeated sequence of two steps: computation and communication. During the computational step, each test tube evolves as an ordinary H system in an independent manner. During the communication step, the words at each test tube are redistributed among other tubes according to some communication protocol.

Test tube systems based on splicing, introduced in [10], communicate through redistribution of the contents of the test tubes via filters that are simply sets of letters (in a similar way to the *separate* operation of Lipton-Adleman [20,1]). These systems, with finite initial contents of the tubes and finite sets of splicing rules associated to each component, are computationally complete, they characterize the family of recursively enumerable languages. The existence of universal splicing test tube distributed systems was obtained on this basis, hence the theoretical proof of the possibility to design universal programmable computers with the structure of such a system. After a series of results, the number of tubes sufficient to achieve this result was established to be 3 [37]. The computational power of splicing test tube systems with two tubes is still an open question. The descriptional complexity for such kind of systems was investigated in [4] where it was shown that there exist universal splicing test tube systems with 10 rules. The best known result shows that there exist universal splicing test tube system with 8 rules [7] and this result is presented in this paper.

A simple possibility to turn splicing-based systems into computationally complete devices are time-varying distributed H systems (TVDH systems). Such systems work like H systems, but on each computational step the set of active rules is changed in a cycle. These sets are called *components*. It was shown [34] that 7 components are enough for the computational completeness; further this number was reduced to 1 [24,26]. This last result shows a fine difference between the definitions of a computational step in H systems. If one iterates the splicing operation while keeping all generated strings, then such systems are regular. If only the result of each splicing step is kept, then the resulting systems are computationally complete. An overview of results on TVDH systems may be found in [27]. Recently one constructed very small universal TVDH systems with two

components and 15 rules and with one component and 17 rules [4]. These results also are presented in the paper.

Another extension of H systems was done using the framework of P systems [32], see also [16] and [35]. In a formal way, splicing P systems can be considered like a graph, whose nodes contain sets of strings and sets of splicing rules. Every rule permits to perform a splicing and to send the result to some other node. Since splicing P systems generate any recursively enumerable language, it is clear that there are universal splicing P systems. Like for small universal Turing machines, we are interested in such universal systems that have a small number of rules. A first result was obtained in [40] where a universal splicing P system with 8 rules was shown. Recently a new construction was presented in [6] for a universal splicing P system with 6 rules. The best known result [7] shows that there exists a universal splicing P system with 5 rules and this result is presented in this paper. Notice, that this result (5 rules) is the best known for "classical" approach to construct small universal devices. Similar investigations for P systems with symbol-objects were done in [11,8] and the latter article constructs a universal antiport P system with 23 rules. This result also is presented in the paper.

We also consider a class of H systems which can be viewed as a counterpart of the matrix grammars in the regulated rewriting area. These systems are called double splicing extended H systems [34]. In [7] one obtains an unexpected result: 5 rules are enough for such kind of H systems in order to be universal.

The following series of results claim existence of universal devices of very small size is presented in the paper. We only present the constructions with some important explanations. Thus, there exist the following universal devices:

- A double splicing extended H system with 5 rules [7],
- An extended splicing test tube system with 3 tubes with 8 rules [7],
- A TVDH system with two components and 15 rules [4],
- A TVDH system with one component and 17 rules [4],
- A splicing P system with 5 rules [7],
- An antiport P system with 23 rules [8].

2 Definitions

In this section, we recall some very basic notions and notations we use throughout the paper. We assume the reader to be familiar with the basics of formal language theory. For more details, we refer to [41].

We denote the empty word by λ and finite alphabets by V and U. A morphism is a mapping $h : V \to U^*$, extended to $h : V^* \to U^*$ by $h(\lambda) = \{\lambda\}$ and $h(xy) = h(x)h(y)$, $x, y \in V^*$. An inverse morphism, denoted as h^{-1} is defined as $h^{-1}(y) = \{x \mid h(x) = y\}$. A weak coding is a morphism $\xi : V \to U \cup \{\lambda\}$, *i.e.*, it can only rename or erase.

Register Machines

A deterministic *register machine* is the following construction:

$$M = (Q, R, q_0, q_f, P),$$

where Q is a set of states, $R = \{R_1, \ldots, R_k\}$ is the set of registers, $q_0 \in Q$ is the initial state, $q_f \in Q$ is the final state and P is a set of instructions (called also rules) of the following form:

1. $(p, [R_k P], q) \in P$, $p, q \in Q, p \neq q, R_k \in R$ (being in state p, increase register R_k and go to state q).
2. $(p, [R_k M], q) \in P$, $p, q \in Q, p \neq q, R_k \in R$ (being in state p, decrease register R_k and go to state q).
3. $(p, \langle R_k \rangle, q, s) \in P$, $p, q, s \in Q, R_k \in R$ (being in state p, go to q if register R_k is not zero or to s otherwise).
4. $(p, \langle R_k Z M \rangle, q, s) \in P$, $p, q, s \in Q, R_k \in R$ (being in state p, decrease register R_k and go to q if successful or to s otherwise).
5. $(q_f, STOP)$ (may be associated only to the final state q_f).

We note that for each state p there is only one instruction of the types above.

A configuration of a register machine is given by the $(k+1)$-tuple $(q, n_1, \cdots, n_k)$, where $q \in Q$ and $n_i \in \mathbb{N}, 1 \leq i \leq k$, describing the current state of the machine as well as the contents of all registers. A transition of the register machine consists in updating/checking the value of a register according to an instruction of one of types above and by changing the current state to another one. We say that the machine stops if it reaches the state q_f. We say that M computes a value $y \in \mathbb{N}$ on the *input* $x \in \mathbb{N}$ if, starting from the initial configuration $(q_0, x, 0, \cdots, 0)$, it reaches the final configuration $(q_f, y, 0, \cdots, 0)$.

It is well-known that register machines compute all partial recursive functions and only them, [28]. For every $n \in \mathbb{N}$, with every register machine M having n registers, an n-ary partial recursive function Φ_M^n is associated. Let $\Phi_0, \Phi_1, \Phi_2, \cdots$, be a fixed admissible enumeration of the set of unary partial recursive functions. Then, a register machine M is said to be *strongly universal* if there exists a recursive function g such that $\Phi_x(y) = \Phi_M^2(g(x), y)$ holds for all $x, y \in \mathbb{N}$.

We also note that the power and the efficiency of a register machine M depends on the set of instructions that are used. In [19] several sets of instructions are investigated. In particular, it is shown that there are strongly universal register machines with 22 instructions of form $[RiP]$ and $\langle RiZM \rangle$. Moreover, these machines can be effectively constructed.

Figure 1 shows this special universal register machine (more precisely in [19] only a machine with 32 instructions of type $[R_k P]$, $[R_k M]$ and $\langle R_k \rangle$ is constructed, and the machine below may be simply obtained from that one).

Here is the list of rules of this machine.

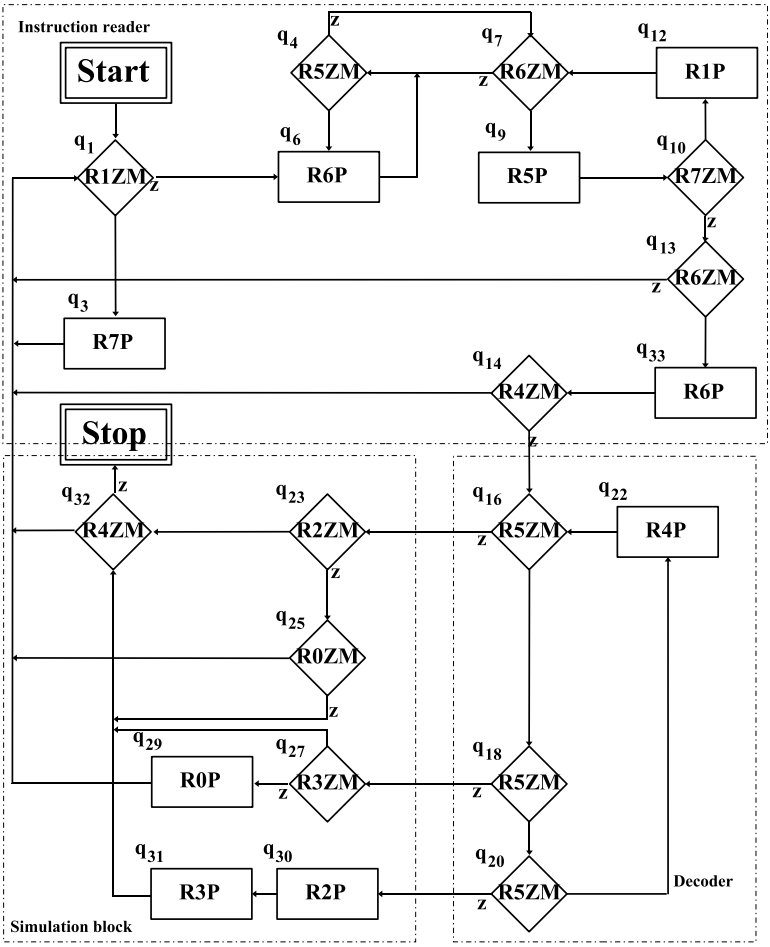

Fig. 1. Flowchart of the strongly universal machine U_{22}

$(q_1, \langle R_1 ZM \rangle, q_3, q_6)$ $(q_3, [R_7 P], q_1)$ $(q_4, \langle R_5 ZM \rangle, q_6, q_7)$
$(q_6, [R_6 P], q_4)$ $(q_7, \langle R_6 ZM \rangle, q_9, q_4)$ $(q_9, [R_5 P], q_{10})$
$(q_{10}, \langle R_7 ZM \rangle, q_{12}, q_{13})$ $(q_{12}, [R_1 P], q_7)$ $(q_{13}, \langle R_6 ZM \rangle, q_{33}, q_1)$
$(q_{33}, [R_6 P], q_{14})$ $(q_{14}, \langle R_4 ZM \rangle, q_1, q_{16})$ $(q_{16}, \langle R_5 ZM \rangle, q_{18}, q_{23})$
$(q_{18}, \langle R_5 ZM \rangle, q_{20}, q_{27})$ $(q_{20}, \langle R_5 ZM \rangle, q_{22}, q_{30})$ $(q_{22}, [R_4 P], q_{16})$
$(q_{23}, \langle R_2 ZM \rangle, q_{32}, q_{25})$ $(q_{25}, \langle R_0 ZM \rangle, q_1, q_{32})$ $(q_{27}, \langle R_3 ZM \rangle, q_{32}, q_1)$
$(q_{29}, [R_0 P], q_1)$ $(q_{30}, [R_2 P], q_{31})$ $(q_{31}, [R_3 P], q_{32})$
$(q_{32}, \langle R_4 ZM \rangle, q_1, q_f)$

Tag Systems

A *tag system* of degree $m > 0$, see [9] and [28], is the triplet $T = (m, V, P)$, where $V = \{a_1, \ldots, a_{n+1}\}$ is an alphabet and where P is a set of productions of the form $a_i \to P_i, 1 \leq i \leq n$, $P_i \in V^*$. For every a_i, $1 \leq i \leq n$, there is exactly one production in P. The symbol a_{n+1} is called the halting symbol. A configuration of the system T is a word $w \in V^*$. If $|w| < m$ or $w = a_{n+1}a_{i_2} \ldots a_{i_m}w'$, with $w' \in V^*$, then w is a halting configuration. We pass from a non-halting configuration $w = a_{i_1}a_{i_2} \ldots a_{i_m}w'$ to the next configuration z by erasing the first m symbols of w and by adding P_{i_1} to the end of the word: $w \Rightarrow z$, if $z = w'P_{i_1}$.

The computation of T over the word $x \in V^*$ is a (finite or infinite) sequence of configurations $x = x_0 \Rightarrow x_1 \Rightarrow \cdots \Rightarrow x_r \Rightarrow ...$ such that for each $j \geq 0$, x_{j+1} is the next configuration of x_j. In the case of a finite sequence $x = x_0 \Rightarrow x_1 \Rightarrow \cdots \Rightarrow x_r$, with x_r being a halting configuration we say that x_r is the result of the computation of T over x.

We say that T recognizes the language L if there exists a recursive coding ϕ such that for all $x \in L$, T halts on $\phi(x)$, and T halts only on words from $\phi(L)$.

Tag systems of degree 2 are able to recognize the family of recursively enumerable languages [9,28]. Moreover, the construction in [9] has non-empty productions and halts only by reaching the symbol a_{n+1} in the first position.

In what follows, for convenience, we consider that the halting symbol is a_1 and $P = \{a_i \to P_i \mid 2 \leq i \leq n\}$.

H Systems

Now we briefly recall the basic notions concerning the splicing operation and related constructs [18,34].

A *splicing rule* (over an alphabet V) is a 4-tuple (u_1, u_2, u_3, u_4) where u_1, u_2, $u_3, u_4 \in V^*$ and it is frequently written as $u_1\#u_2\$u_3\#u_4$, $\{\$, \#\} \notin V$. Strings u_1u_2 and u_3u_4 are called splicing *sites*.

We say that a word x *matches* rule r if x contains an occurrence of one of the two sites of r. We also say that x and y are *complementary* with respect to a rule r if x contains one site of r and y contains the other one. In this case we also say that x or y may *enter* rule r. When x and y can enter a rule $r = u_1\#u_2\$u_3\#u_4$, *i.e.*, $x = x_1u_1u_2x_2$ and $y = y_1u_3u_4y_2$, it is possible to define the application of r to the couple x, y. The result of this application are w and z, where $w = x_1u_1u_4y_2$ and $z = y_1u_3u_2x_2$. We also say that x and y are spliced and w and z are the result of this splicing. We write this as follows: $(x, y) \vdash_r (w, z)$ or

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

The pair $h = (V, R)$, where V is an alphabet and R is a set of splicing rules, is called a *splicing scheme* or an *H scheme*.

For a splicing scheme $h = (V, R)$ and for a language $L \subseteq V^*$ we define

$$\sigma_h(L) \stackrel{\text{def}}{=} \{w, z \in V^* \mid \exists x, y \in L, \exists r \in R : (x, y) \vdash_r (w, z)\}.$$

Now we can introduce the iteration of the splicing operation.

$$\begin{aligned}\sigma_h^0(L) &= L,\\ \sigma_h^{i+1}(L) &= \sigma_h^i(L) \cup \sigma(\sigma^i(L)),\ i \geq 0,\\ \sigma_h^*(L) &= \cup_{i\geq 0}\sigma_h^i(L).\end{aligned}$$

It is known that the iterated splicing preserves the regularity of a language [34].

A *Head-splicing-system* [17,18], or H *system*, is a construct $H = (h, A) = ((V, R), A)$, which consists of an alphabet V, a set $A \subseteq V^*$ of initial words over V, the *axioms*, and a set $R \subseteq V^* \times V^* \times V^* \times V^*$ of splicing rules. System H is called finite if A and R are finite sets.

The language generated by an H system H is defined as $L(H) \stackrel{\text{def}}{=} \sigma_h^*(A)$.

An *extended* H system is the quadruple $H = (V, T, A, R)$, where $H' = ((V, R), A)$ is an H system and $T \subseteq V$ is a terminal alphabet. The language generated by the extended H system H is defined as $L(H) = L(H') \cap T^*$.

We now consider a class of H systems which can be viewed as a counterpart of the matrix grammars in the regulated rewriting area. They require that the work of an H system proceeds in a couple of steps: the two strings obtained after a splicing immediately enter a second splicing. The rules used in the two steps are not prescribed or dependent in any way on each other.

Consider an extended H system $\Gamma = (V, T, A, R)$. For $x, y, w, z, u, v \in V^*$ and $r_1, r_2 \in R$ we write

$$(x, y) \vdash_{r_1,r_2} (w, z) \text{ iff } (x, y) \vdash_{r_1} (u, v) \text{ and } (u, v) \vdash_{r_2} (w, z) \text{ or } (v, u) \vdash_{r_2} (w, z).$$

For a language $L \subseteq V^*$ we define

$$\begin{aligned}\sigma_d(L) &= \{w, z \mid (x, y) \vdash_{r_1,r_2} (w, z) \ \text{ for } x, y \in L, r_1, r_2 \in R\},\\ \sigma_d^*(L) &= \bigcup_{i\geq 0} \sigma_d^i(L), \text{ where}\\ \sigma_d^0(L) &= L,\\ \sigma_d^{i+1}(L) &= \sigma_d^i(L) \cup \sigma_d(\sigma_d^i(L)),\ i \geq 0.\end{aligned}$$

Then, we associate with Γ the language

$$L_d(\Gamma) = \sigma_d^*(A) \cap T^*.$$

We say that $L_d(\Gamma)$ is the language generated by the *double splicing extended H system* Γ.

By $EH_2(FIN)$ we denote the family of languages $L_d(\Gamma)$ generated as above by double splicing extended H systems. It is known that $RE = EH_2(FIN)$ [34].

We say that $\Gamma = (V, T, A, R)$ *computes* $L \subseteq V^*$ on input w if $L = L_d(\Gamma')$, where $\Gamma' = (V, T, A \cup \{w\}, R)$ and we denote this as $L_d(\Gamma, w)$.

Splicing Test Tube Systems

There are several variants of splicing test tube systems, called also *communicating distributed H systems*. We consider the (historically) first variant introduced in [10] and also described in [34].

A *splicing test tube* T is a couple $T = (H, F)$ consisting of an H system $H = (h, A) = ((V, R), A)$ and an alphabet $F \subseteq V$, called the *filter* for T.

A *splicing test tube system* with n test tubes is a tuple $\Delta = (V, T_1, \cdots, T_n)$, where V is an alphabet and $T_i = (H_i, F_i) = (((V, \mathcal{R}_i), A_i), F_i)$, $1 \leq i \leq n$, are n splicing test tubes.

The *computation* in Δ is a sequence of two subsequent steps, a computation step and a communication step, which are repeated iteratively and change the configuration of the system. By a *configuration* of Δ, above, we mean an n-tuple $(\mathcal{L}_1, \ldots, \mathcal{L}_n)$, where $\mathcal{L}_i \in V^*$, $1 \leq i \leq n$. The initial configuration of Δ is $(A_1, \ldots, A_n)$.

The *computation* step consists in an iterative application of $\mathcal{R}_i$, $\sigma^*_{\mathcal{R}_i}$, at each node i of G to strings found there. We say that configuration $C' = (\mathcal{L}'_1, \ldots, \mathcal{L}'_n)$ is obtained from configuration $C = (\mathcal{L}_1, \ldots, \mathcal{L}_n)$ by a computation step in Δ, denoted by $(\mathcal{L}_1, \ldots, \mathcal{L}_n) \vdash_{comp} (\mathcal{L}'_1, \ldots, \mathcal{L}'_n)$, if $\mathcal{L}'_i = \sigma^*_{\mathcal{R}_i}(\mathcal{L}_i)$ holds for $1 \leq i \leq n$.

During the *communication* step, the actual contents of the test tubes are redistributed to all other tubes. More formally, we say that configuration $(\mathcal{L}'_1, \ldots, \mathcal{L}'_n)$ is obtained from configuration $(\mathcal{L}_1, \ldots, \mathcal{L}_n)$ by a communication step in Δ, denoted by $(\mathcal{L}_1, \ldots, \mathcal{L}_n) \vdash_{comm} (\mathcal{L}'_1, \ldots, \mathcal{L}'_n)$, if $\mathcal{L}'_i$ consists of all words $w \in V^*$ which satisfy one of the following conditions:

- $w \in \mathcal{L}_j$, for some j, $1 \leq j \leq n$, and $w \in F_i^*$,
- $w \in \mathcal{L}_i$ and there is no such j, $1 \leq j \leq n$, such that $w \in F_j^*$.

For two configurations C and C' we denote by $C \vdash C'$ the sequence $C \vdash_{comp} C'' \vdash_{comm} C'$, where C'' is an intermediate configuration. By $\vdash^*$ we denote the reflexive and transitive closure of $\vdash$.

We can define the *communication graph* of the system which is the graph where a node corresponds to a test tube and an edge from node i to j corresponds to a possibility to send a word from tube i to tube j. It is clear that in the case of the standard definition the communication graph is complete and also contains self-loops. Variants where the communication graph has other forms are known under the name of *splicing P systems*, see [32,33].

The *result* of the computation of Δ is the contents of the first test tube. More formally, $L(\Delta) = \{L_1 \subseteq V^* \mid \exists L_2, \cdots, L_n \subseteq V^* : (A_1, \cdots, A_n) \vdash^* (L_1, \cdots, L_n)\}$.

An *extended splicing test tube system* Γ is a pair $\Gamma = (\Delta, \mathcal{T})$, where Δ is a splicing test tube system defined as above and $\mathcal{T} \subseteq V$ is an alphabet. The computation of such system is similar to splicing test tube system, the only difference is the result of the computation which is defined as follows: $L(\Gamma) = L(\Delta) \cap \mathcal{T}^*$.

It is known that splicing test tube systems with one tube are isomorphic to H systems, hence they generate the family of regular languages and extended splicing test tube systems with 3 tubes are computationally complete [37]. If two tubes are used, the computational power of such systems is not known, however non-regular languages can be generated, as shown in [34].

If a different definition of the filter is considered, then two tubes are enough for the computational completeness, see [14,15,45,46].

In this article we consider extended splicing test tube systems that have an input. A computation of an extended splicing test tube system Γ on an input w is performed by adding w to some A_i, $1 \leq i \leq n$, and after that evolving Γ as usual. The resulting language is denoted $L(\Gamma, w)$.

Time-Varying Distributed H Systems

A time-varying distributed H system (of degree $n,\ n \geq 1$), (TVDH system) is a construct:

$$D = (V, T, A, R_1, R_2, \ldots, R_n),$$

where V is *an alphabet*, $T \subseteq V$ is *a terminal alphabet*, $A \subseteq V^*$ is a finite set of *axioms*, and *components* R_i are finite sets of splicing rules over $V,\ 1 \leq i \leq n$.

At each moment $k = n \cdot j + i$, for $j \geq 0,\ 1 \leq i \leq n$, only component R_i is used for splicing the currently available strings. Specifically, we define

$$L_1 = A, \qquad L_{k+1} = \sigma_{h_i}(L_k),\ \text{for } i \equiv k(mod\ n),\ k \geq 1, 1 \leq i \leq n,\ h_i = (V, R_i).$$

Therefore, from a step k to the next step, $k+1$, one passes only the result of splicing the strings in L_k according to the rules in R_i for $i \equiv k(mod\ n)$; the strings in L_k that cannot enter a splicing rule are removed.

The language generated by D is, by definition:

$$L(D) \stackrel{\text{def}}{=} (\cup_{k \geq 1} L_k) \cap T^*.$$

In this article we consider TVDH systems that have an input. A computation of a TVDH system D on an input w is performed by adding w to A and after that evolving D as usual. The resulting language is denoted $L(D, w)$.

Splicing (Tissue) P Systems

A *splicing tissue P system* of degree $m \geq 1$ is a construct

$$\Pi = (V, T, G, A_1, \ldots, A_m, R_1, \ldots, R_m),$$

where V is an alphabet, $T \subseteq V$ is the terminal alphabet and G is the underlying directed labeled graph of the system. The graph G has m nodes (cells) numbered from 1 to m. Each node i contains a set of strings (a language) A_i over V. Symbols R_i, $1 \leq i \leq m$, are finite sets of rules (associated to nodes) of the form

$(r; tar_1, tar_2)$, where r is a splicing rule: $r = u_1\#u_2\$u_3\#u_4$ and $tar_1, tar_2 \in \{here, out\} \cup \{go_j \mid 1 \leq j \leq m\}$, are target indicators. The communication graph G can be deduced from the sets of rules. More precisely, G contains an edge (i, j), iff there is a rule $(r; tar_1, tar_2) \in R_i$ with $tar_k = go_j$, $k \in \{1, 2\}$. If one of tar_k is equal to *here*, then G contains the loop (i, i).

A *configuration* of Π is the m-tuple $(N_1, \ldots, N_m)$, where $N_i \subseteq V^*$. A *transition* between two configurations $(N_1, \ldots, N_m) \Rightarrow (N'_1, \ldots, N'_m)$ is defined as follows. In order to pass from one configuration to another, splicing rules of each node are applied in parallel to all possible words that belong to that node. After that, the result of each splicing is distributed according to target indicators. More exactly, if there are x, y in N_i and $r = (u_1\#u_2\$u_3\#u_4; tar_1, tar_2)$ in R_i, such that $(x, y) \vdash_r (w, z)$, then words w and z are sent to the nodes indicated by tar_1, respectively tar_2. We write this as follows $(x, y) \vdash_r (w, z)(tar_1, tar_2)$. If $tar_k = here$, $k = 1, 2$, then the word remains in node i (is added to N'_i); if $tar_k = go_j$, then the word is sent to node j (is added to N'_j); if $tar_k = out$, the word is sent outside of the system.

Since the words are present in an arbitrarily many number of copies, after the application of rule r in node i, words x and y are still present in the same node.

A *computation* in a splicing tissue P system Π is a sequence of transitions between configurations of Π which starts from the initial configuration $(A_1, \ldots, A_m)$. The result of the computation consists of all words over terminal alphabet T which are sent outside the system at some moment of the computation. The equivalent definition of the result is to define the output node i_{out} (in this case we define splicing tissue P system of degree $m \geq 1$ as follows $\Pi = (V, T, G, A_1, \ldots, A_m, R_1, \ldots, R_m, i_{out})$, $1 \leq i_{out} \leq m$) and consider as result of all words over terminal alphabet that will appear in this output node i_{out}. We denote by $L(\Pi)$ the language generated by system Π.

We also define the notion of an *input* for the system above. An input word for a system Π is simply a word w over the non-terminal alphabet of Π. The computation of Π on input w is obtained by adding w to the axioms of A_1 and after that by evolving Π as usual. We denote by $L(\Pi, w)$ the result of the computation of Π on w.

We consider the following restricted variant of splicing tissue P systems. A *restricted splicing tissue P system* is a subclass of splicing tissue P systems which has the property that for any rule $(r; tar_1, tar_2)$ either $tar_1 = tar_2 = go_j$, or $tar_1 = tar_2 = out$ or $tar_1 = tar_2 = here$. This means that both resulting strings are moved over the same connection. In this case, we may associate splicing rules to corresponding edges.

3 Small Universal Splicing (Tissue) P system

In this section we consider a small universal splicing (tissue) P system from [7]. Here and in sections below we use the unary codings $c : V \to \{\alpha, \beta\}^*$ and $\bar{c} : V \to \{\alpha, \beta\}^*$ defined as $c(a_i) = \alpha^i\beta$, $\bar{c}(a_i) = \beta\alpha^i$ where $V = \{a_1, \ldots, a_n\}$.

Theorem 1. *Let $TS = (2, V, P)$ be a tag system. Then, there is a morphism h, a weak coding ξ and a restricted splicing tissue P system $\Pi = (V', T, G, A_1, A_2, A_3, R_1, R_2, R_3, 3)$ with 5 rules which simulates TS as follows:*

1. *for any word $w \in V^*$ on which TS halts producing the result v, the application of $h^{-1} \circ \xi$ to the result of the computation of Π on the input $X\beta\beta c(w)\beta Y$ gives v,* i.e., *$\xi(h^{-1}(L(\Pi, X\beta\beta c(w)\beta Y))) = \{v\}$.*
2. *for any word $w \in V^*$ on which TS does not halt, the system Π generates the empty set given the input $X\beta\beta c(w)\beta Y$,* i.e., *$L(\Pi, X\beta\beta c(w)\beta Y) = \emptyset$.*

We construct a restricted splicing P system $\Pi = (V', T, G, A_1, A_2, A_3, R_1, R_2, R_3, 3)$ as follows. Let $|V| = n,\ n \geq 2$. We put $V' = \{\alpha, \beta, X, Y, Y', Z, Z'\}$, and $T = \{X, Y', \alpha, \beta\}$.

The initial languages $A_j,\ j \in \{1, 2, 3\}$ are given as follows.

$$A_1 = \{Z'c(P_i)\bar{c}(a_i)Y \mid a_i \to P_i \in P\} \cup \{X\beta Z,\ ZY,\ Z'Y'\},$$
$$A_2 = \{XZ\},$$
$$A_3 = \{XZ\}.$$

The set of rules $R_j,\ j \in \{1, 2, 3\}$ are given as follows.

$$R_1 = \{1.1 : (\lambda\#\beta Y\$Z'\#\lambda; go_3, go_3);\ 1.2 : (\lambda\#\alpha Y\$Z\#Y; go_2, go_2);$$
$$1.3 : (X\beta\alpha\#\lambda\$X\beta\#Z; here, here)\};$$
$$R_2 = \{2.1 : (X\alpha\#\lambda\$X\#Z; go_1, go_1)\};$$
$$R_3 = \{3.1 : (X\beta\beta\#\alpha\alpha\$X\#Z; go_1, go_1)\}.$$

The graph G can be deduced from the rules above and it is represented in Figure 2.

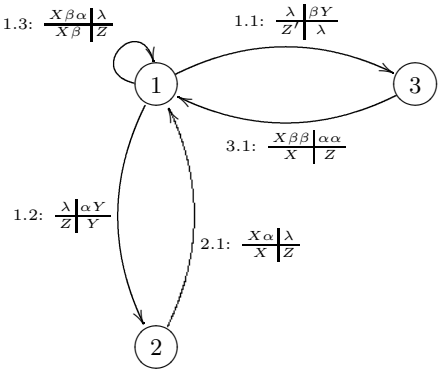

Fig. 2. The communication graph G associated to the construction of Π

The simulation of TS is performed as follows. For every step of the derivation in TS there is a sequence of several derivation steps in Π. The current configuration w of TS is encoded by a string $X\beta\beta c(w)\beta Y$ present in node 1 of Π (the initial configuration of Π satisfies this property). The simulation of a production $a_i \to P_i$, $2 \le i \le n$ is performed using the rotate-and-simulate method used for many proofs in this area. We use this method works as follows. First, suffixes $c(P_j)\bar{c}(a_j)$, $2 \le j \le n$ are attached to the string producing $X\alpha^i\beta c(a_k w')c(P_j)\beta\alpha^j Y$. After that the number of symbols α at both ends is decreased simultaneously. Hence, only the string for which $j = i$ will remain at the end, producing $X\beta c(a_k w')\beta Y$. After that the symbol a_k is removed (by removing corresponding α's) and a new round begins. The simulation stops when the first symbol is a_1.

Consider also the morphism h and the weak coding ξ defined as follows:

$$h(a) = \begin{cases} \alpha^i\beta & \text{if } a = a_i \in V, \\ X\beta\beta\alpha\beta & \text{if } a = \bar{X}, \\ Y' & \text{if } a = \bar{Y}. \end{cases} \qquad \xi(a) = \begin{cases} a & \text{if } a \in V, \\ a_1 & \text{if } a = \bar{X}, \\ \lambda & \text{if } a = \bar{Y}. \end{cases}$$

From the definition of h and from the form of words that can be in node 3 in Π it is clear that $h^{-1}(w)$ is not empty iff w contains both $X\beta\beta\alpha\beta$ and Y'. However such a word corresponds to the resulting word from Π'. Hence using h^{-1} it is possible to filter out the words that do not correspond to the final result. At the same time h^{-1} decodes the remaining part of the string. Finally, the weak coding ξ removes the markers for the beginning and for the end of the word ($\bar{X}$ and $\bar{Y}$).

The universality of the corresponding system follows from the existence of universal tag systems.

Theorem 2. *There exists a universal splicing (tissue) P system with 5 rules.*

4 Small Universal Double Splicing Extended H System

In this section we consider a small universal double splicing extended H system from [7].

Theorem 3. *Let $TS = (2, V, P)$ be a tag system. Then, there is a double splicing extended H system $\Gamma_1 = (V', T, A, R)$ with 5 rules that simulates TS as follows:*

1. *for any word $w \in V^*$ on which TS halts producing the result v, the system Γ_1 produces a unique result $X'c(v)Y'$, i.e. $L_d(\Gamma_1, X\beta\alpha\beta c(w)\beta Y) = \{X'c(v)Y'\}$,*
2. *for any word $w \in V^*$ on which TS does not halt, the system Γ_1 computes infinitely without producing a result, i.e. $L_d(\Gamma_1, X\beta\alpha\beta c(w)\beta Y) = \emptyset$.*

We construct the system Γ_1 as follows.

Let $|V| = n$, $n \ge 2$ and $1 \le j \le n$. The terminal and non-terminal alphabets of Γ_1 are the following:

$V' = \{\alpha, \beta, X, Y, X', Y', Z_1, Z_2, Z_3, Z_4\}$, $T = \{X', Y', \alpha, \beta\}$.

The axioms A and rules R are given as follows.
$A = \{XZ_1c(P_i)\bar{c}(a_i)Y \mid a_i \to P_i \in P\} \cup \{XZ_2Y,\ X\beta Z_3Z_1\beta Y,\ X'Z_4Z_1Y'\}$.

$$\begin{aligned} R = \{&1 : \beta\#\beta Y\$Z_1\#\lambda, && 2 : X\beta\alpha\beta\#\alpha\alpha\$X\#Z_1,\\ &3 : \lambda\#\alpha Y\$Z_2\#Y, && 4 : X\alpha\#\lambda\$X\#Z_2,\\ &5 : X\beta\alpha\#\lambda\$X\beta\#Z_3\}. \end{aligned}$$

Now one uses the following morphism h and the weak coding ξ to get resulting strings:

$$h(a) = \begin{cases} \alpha^i\beta & \text{if } a = a_i \in V,\\ X\beta\alpha\beta\alpha\beta & \text{if } a = \bar{X},\\ Y' & \text{if } a = \bar{Y}. \end{cases} \qquad \xi(a) = \begin{cases} a & \text{if } a \in V,\\ a_1 & \text{if } a = \bar{X},\\ \lambda & \text{if } a = \bar{Y}. \end{cases}$$

The universality of the corresponding system follows from the existence of universal tag systems.

Theorem 4. *There exists a universal double splicing extended H system with 5 rules.*

5 Small Universal Extended Splicing Test Tube System

In this section we present a small universal splicing test tube system from [7].

Theorem 5. *[7] Let $TS = (2, \Sigma, P)$ be a tag system. Then, there is an extended splicing test tube system with 3 tubes $\Gamma_2 = ((V, T_1, T_2, T_3), \mathcal{T})$ and 8 rules, which simulates TS as follows:*

1. *for any word $w \in \Sigma^*$ on which TS halts producing the result v, the system Γ_2 produces a unique result $X_0c(v)Y$,* i.e., *$L(\Gamma_2, Xc(w)\beta Y) = \{X_0c(v)Y\}$.*
2. *for any word $w \in \Sigma^*$ on which TS does not halt, the system Γ_2 computes infinitely without producing a result,* i.e., *$L(\Gamma_2, Xc(w)\beta Y) = \emptyset$.*

We construct the system Γ_2 as follows.
$V = \{\alpha, \beta, X, X', X_0, Y, Y', Y'', Z\}$, $\mathcal{T} = \{X_0, Y, \alpha, \beta\}$.
$T_1 = (((V, R_1), A_1), F_1)$ with $F_1 = \{\alpha, \beta, X, X_0, Y, Y'\}$ and $A_1 = \{Zc(P_i)\ \bar{c}(a_i)Y'' \mid a_i \to P_i \in P\} \cup \{X'Z, ZY''\}$. R_1 consists of the following splicing rules:

$$1.1 : \beta\#\beta Y\$Z\#\alpha\ ; \qquad 1.2 : X\#\lambda\$X'\#Z\ ; \qquad 1.3 : \lambda\#Y'\$Z\#Y''.$$

$T_2 = (((V, R_2), A_2), F_2)$ with $F_2 = \{\alpha, \beta, X', Y''\}$ and $A_2 = \{XZ, ZY'\}$. R_2 consists of the following splicing rules:

$$2.1 : X'\alpha\#\lambda\$X\#Z\ ; \qquad 2.2 : \lambda\#\alpha Y''\$Z\#Y'.$$

$T_3 = (((V, R_3), A_3), F_3)$ with $F_3 = \{\alpha, \beta, X', Y''\}$ and $A_3 = \{X'\beta Z, XZ, X_0Z, Z\beta Y\}$. R_3 consists of the following splicing rules:

$$3.1 : X'\beta\alpha\#\lambda\$X'\beta\#Z\ ; \quad 3.2 : \quad X'\beta\beta\#\lambda\$X\#Z\ ; \quad 3.3 : \lambda\#\beta Y''\$Z\#\beta Y.$$

Now one uses the following morphism h and the weak coding ξ to get resulting strings:

$$h(a) = \begin{cases} \alpha^i\beta & \text{if } a = a_i \in V, \\ X\beta\beta\alpha\beta & \text{if } a = \bar{X}, \\ \beta Y & \text{if } a = \bar{Y}. \end{cases} \qquad \xi(a) = \begin{cases} a & \text{if } a \in V, \\ a_1 & \text{if } a = \bar{X}, \\ \lambda & \text{if } a = \bar{Y}. \end{cases}$$

The universality of the corresponding system follows from the existence of universal tag systems.

Theorem 6. *There exists a universal extended splicing test tube system with 3 test tubes and 8 rules.*

6 Small Universal TVDH Systems

In this section we present two small universal TVDH systems from [4].

Theorem 7. *Let $G = (2, \Sigma, P)$ be a tag system and $w \in \Sigma^*$. Then, there is a TVDH system of degree 2, $D_1 = (V, T, A, R_1, R_2)$, with 15 rules, which given the word $Xc(w)Y_0 \in V^*$ as input simulates G on input w,* i.e. *such that:*

1. *for any word w on which TS halts producing the result z, the system D_1 produces a unique result $c(z)Y_0$,* i.e., $L(D_1, w) = \{c(z)Y_0\}$.
2. *for any word w on which TS does not halt, the system D_1 computes infinitely without producing a result,* i.e., $L(D_1, w) = \emptyset$.

We construct the system D_1 as follows.

$V = \{\alpha, \beta, X, X', X'', Y, Y', Y'', Y_0, Z, Z_1, Z_2, \mathcal{Z}_1, \mathcal{Z}_2\}$, $T = \{Y_0, \alpha, \beta\}$.

The axioms are given as follows.

$A = \{\mathcal{Z}_1c(P_i)\bar{c}(a_i)Y \mid a_i \to P_i \in P\} \cup \{XZ_1Y_0, XZ_2Y_0, XZ_1Y, XZ_2Y, X'Z_1Y', X'Z_2Y', X''Z_1Y'', X''Z_2Y'', \mathcal{Z}_2Z\}$.

The rules are given as follows (the first number indicates the component to which the rule belongs).

$1.1 : \alpha\beta\#Y_0\$\mathcal{Z}_1\#\alpha\ ;$	$1.2 : \varepsilon\#\alpha Y\$Z_1\#Y\ ;$	$1.3 : X\beta\#\varepsilon\$X'\#Z_1\ ;$
$1.4 : X''\#\varepsilon\$X'\#Z_1\ ;$	$1.5 : X'\beta\#\alpha\alpha\$X\#Z_1\ ;$	$1.6 : X'\beta\#\alpha\beta\$\varepsilon\#\mathcal{Z}_2Z\ ;$
$1.7 : Z_1\#\varepsilon\$Z_2\#\varepsilon\ ;$	$1.8 : \mathcal{Z}_1\#\alpha\$\mathcal{Z}_2\#Z$	$2.1 : X\alpha\#\varepsilon\$X\#Z_2;$
$2.2 : \beta\#\beta Y\$Z_2\#Y';$	$2.3 : X'\alpha\#\varepsilon\$X''\#Z_2;$	$2.4 : \beta\#Y'\$Z_2\#Y''$
$2.5 : \beta\#Y''\$Z_2\#Y_0;$	$2.6 : Z_1\#\varepsilon\$Z_2\#\varepsilon;$	$2.7 : \mathcal{Z}_1\#Z\$\mathcal{Z}_2\#\alpha;$

The construction follows the idea from [40,25]. The simulation of TS is performed as follows. For every step of the derivation in TS there is a sequence of several derivation steps in D_1. The current configuration $w = a_i a_k w'$, $i \neq 1$ of TS is encoded by a string $Xc(w)Y_0$ present in component 1 of D_1 (the initial configuration of D_1 satisfies this property). The simulation of a production $a_i \to P_i$, $2 \leq i \leq n$ is performed using the rotate-and-simulate method used for many proofs in this area. We use this method as follows. First, by rule 1.1, suffixes $c(P_j)\bar{c}(a_j)$, $2 \leq j \leq n$ are attached to the string producing words $X\alpha^i\beta c(a_k w')c(P_j)\beta\alpha^j Y$. After that symbols α is removed at both ends simultaneously by rules 1.2 and 2.1. The strings having $j \neq i$ will be eliminated, corresponding checks are done by rules 1.3 and 2.2. Hence, only the string for which $j = i$ will remain at the end, producing $X'c(a_k w')Y'$. After that the symbol a_k is removed (by removing corresponding α's) by rules 2.3 and 1.4 and after applying rules 2.4, 1.5 and 2.5 string $Xc(w')Y_0$ will appear at component 1 and a new round begins. The simulation stops when the first symbol is the halting symbol a_1. In this case rule 1.6 is used producing $c(z)Y_0$.

The universality of the corresponding system follows from the existence of universal tag systems.

Theorem 8. *There exists a universal TVDH system of degree 2 with 15 rules.*

Theorem 9. *Let $G = (2, \Sigma, P)$ be a tag system and $w \in \Sigma^*$. Then, there is a TVDH system of degree 1, $D_2 = (V, T, A, R_1)$, with 17 rules, which given the word $Xc(w)Y_0 \in V^*$ as input simulates G on input w,* i.e. *such that:*

1. *for any word w on which TS halts producing the result w', the system D_1 produces a unique result $X_0c(w')Y_0$,* i.e., $L(D_2, w) = \{X_0c(w')Y_0\}$.
2. *for any word w on which TS does not halt, the system D_1 computes infinitely without producing a result,* i.e., $L(D_2, w) = \emptyset$.

We construct the system D_2 as follows.

$V = \{\alpha, \beta, X, X', X'', Y, Y', Y'', X_0, Y_0, Z, Z_1, Z_2, K\}$, $T = \{X_0, Y_0, \alpha, \beta\}$.
The axioms are given as follows.

$A = \{ZZ_1Kc(P_i)\bar{c}(a_i)Y \mid a_i \to P_i \in P\} \cup \{ZZ_1Y, X'Z_1Z, XZ_1Z, X_1Z_1Z, X_0Z_1Z, XZ_2Z', Z'Z_2Y', Z'Z_2Z_2Y'', X''Z_2Z', Z'Z_2Y_0, Z'Z_2Y_1\}$.

1.1 $:\alpha\beta\#Y_0\$ZZ_1K\#\alpha$;	1.2 $:\varepsilon\#\alpha Y\$ZZ_1\#Y$;	1.3 $:X\beta\#\alpha\$X_1\#Z_1Z$;
1.4 $:X_1\alpha\#\varepsilon\$X'\#Z_1Z$;	1.5 $:X''\#\varepsilon\$X'\#Z_1Z$;	1.6 $:X'\beta\#\alpha\alpha\$X\#Z_1Z$;
1.7 $:X'\beta\#\alpha\beta\$X_0\#Z_1Z$;	1.8 $:X\alpha\#\varepsilon\$X\#Z_2$;	1.9 $:\beta\#\beta Y\$ZZ_2\#Y_1$;
1.10 $:\beta\#Y_1\$ZZ_2\#Y'$;	1.11 $:X'\alpha\#\varepsilon\$X''\#Z_2Z$	1.12 $:\beta\#Y'\$ZZ_2Z_2\#Y''$;
1.13 $:\beta\#Y''\$ZZ_2\#Y_0$;	1.14 $:Z\#Z_1\$Z'\#Z_2$;	1.15 $:Z\#Z_2\$Z'\#Z_1$;
1.16 $:Z_1\#Z\$Z_2\#Z'$;	1.17 $:Z_2\#Z\$Z_1\#Z'$.	

The universality of the corresponding system follows from the existence of universal tag systems.

Theorem 10. *There exists a universal TVDH system of degree 1 with 17 rules.*

7 Small Universal Antiport P System

In this section we present a small universal antiport P system from [8], constructed by simulating the universal register machine U_{22} from [19], see Figure 1 in Section 2.

Theorem 11. *There exists a universal antiport P system with 23 rules.*

The proof has been presented in terms of maximally parallel multiset rewriting systems. Indeed, a multiset rewriting system directly correponds to a one-membrane symport/antiport system with environment containing an unbounded supply of all objects, and rule $u \to v$ corresponds to rule $(u, out; v, in)$.

We now present the formal description of the system; the flowchart representing its finite state transition graph is illustrated by Figure 4:

$$\gamma = (O, R, \{R_1\}, \mathcal{I}, \mathcal{P}), \text{ where}$$
$$O = R \cup \{C_3, C_5', C_6'\} \cup \{q_{16}, q_{27}\} \cup \{T, I, J, K, L, M, N, O, P, Q, T, X\},$$
$$R = \{R_i \mid 0 \le i \le 7\},$$
$$\mathcal{I} = LQLQJJNXXXR_0^{i_0} \cdots R_7^{i_7}.$$

Here $i_0, \cdots, i_7$ is the contents of registers 0 to 7 of U_{22} and $LQLQJJNXXX$ is the encoding of the initial state q_1C_1S. The table below gives the set $\mathcal{P}$ of rules.

$phase : XX \to XT$	$a : LQLQJJNTT \to JJLOR_6XX$
$D0 : IJKPQR_0 \to LQLQJJM$	$b : LC_5'TT \to JJLOR_6XX$
$D1 : LQLQJJNR_1 \to LPLPJJMR_7$	$c : OC_6'TT \to IILQLQNR_5XX$
$D2 : IIKPQR_2 \to JJKPQ$	$d : QLQNC_6'TT \to JJKQQR_6XX$
$D3 : q_{27}C_3R_3 \to JJKPQ$	$e : q_{27}C_3TT \to LQLQJJNR_0XX$
$D4 : JJKR_4 \to JJLLM$	$f : q_{16}JJOC_5'C_5'TT \to LQLQJJNR_2R_3XX$
$D5 : JJOR_5 \to C_5'$	$g : q_{16}C_5'C_5'C_5'TT \to q_{16}JJOJJOJJOXX$
$D6 : IJLR_6 \to C_6'$	$1 : JJLOTT \to IJLOXX$
$D7 : IILQLQNR_7 \to IJLOR_1$	$5 : JJKQQTT \to q_{16}JJOJJOJJOXX$
$A : ITT \to JXX$	$8 : q_{16}JJOJJOJJOTT \to IIKPQMXX$
$B : JJMTT \to JJNXX$	$12 : q_{16}JJOJJOC_5'TT \to q_{27}C_3XX$
$C : LP \to LQ$	

In fact, by simulation all objects except $R_0, \cdots, R_7$ appear inside the system in bounded quantities, so the constructed system is explained by projections of configurations onto $O' = O \setminus \{R_0, \cdots, R_7\}$, yielding a finite transition graph. We refer to its nodes as *finite states*. The possibility of some transitions, however, depends on the availability of objects R_j, $0 \le j \le 7$. In [8] one thus speaks about *finite-state maximally parallel multiset rewriting systems* (FsMPMRSs).

Machine U_{22} may be simulated in a *straightforward* way, by rule $q \to R_iq_1$ for each instruction $(q, [R_kP], q_1)$ and by rules

$$q \to q'C_q,\ q' \to q'',\ C_qR_{i_q} \to C_q',\ q''C_q \to q_1,\ q''C_q' \to q_2$$

for each instruction $(q, \langle R_{i_q} ZM \rangle, q_1, q_2)$. This yields a universal P system with 73 symport/antiport rules, reported already in [11] (together with some optimizations). The number of rules is then decreased at the expense of their weight. The overall behaviour eventually gets quite complicated, so flowcharts are used to describe it. A square represents a finite state (see the previous paragraph), and a circle attached to it represents a (possibly partial) application of rules; multiple circles may be drawn as one for simplicity.

Multiple techniques are used to decrease the number of rules. First, if one rule (e.g., increment) is always applied after another one, then they can be merged, *eliminating an intermediate state*. A state then typically contains a *checker* (object C with an index, possibly primed), verifying whether a specific register is present in the system (is non-zero); addition instructions and renaming rules are no longer present as separate rules. This increases the weight of rules to 5.

A very important optimization is *gluing*: the representation of the configurations is changed such that the effect of multiple rules is obtained by one rule. A general scheme is the following: suppose we have rules $r_1 : c_1 \to c_2$ and $r_2 : d_1 \to d_2$. They both can be replaced by a rule $r : X \to Y$ if we transform the representation as follows: $c_1 = cX$, $c_2 = cY$, $d_1 = dX$, $d_2 = dY$. It is, however, needed that no state is a submultiset of another state.

We now proceed with two simple special cases of gluing. The first case is *phases*. Representing states q and q' by qS and qS' lets us glue all rules $q \to q'$ (waiting while the checker gets a chance to decrement a register) yielding a single rule $S \to S'$. Later, *three phases* help to further optimize the other rules, but the transitions $S \to S'$ and $S' \to S''$ are also glued by substitution $S = XXX$,

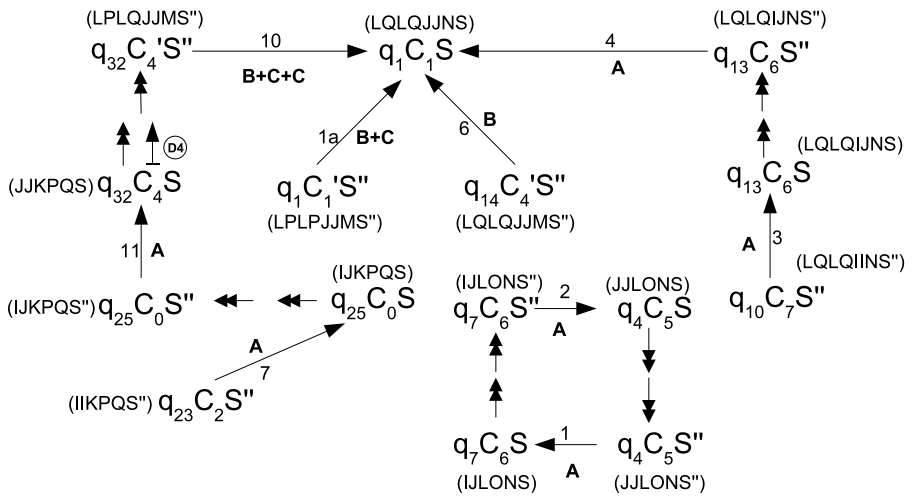

Fig. 3. Part of the multiset rewriting flowchart of U_{22} showing only glued rules and the corresponding encoding

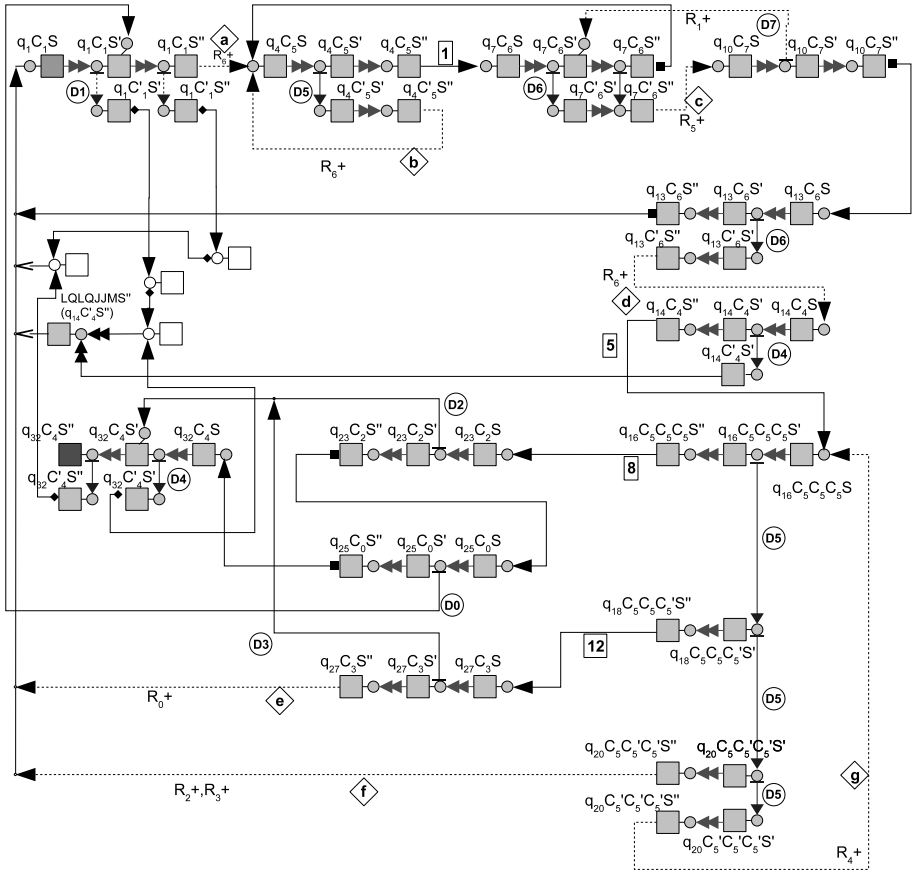

Fig. 4. Multiset rewriting flowchart of U_{22} with glued rules

$S' = XXT$, $S'' = XTT$ yielding a single rule $XX \to XT$. This phase rule is represented on flowcharts by a double-headed arrow.

The second simple special case of gluing is *unifying the checkers* that decrement the same register. Now the state typically contains a phase, a checker, and the rest of the state is currently a symbol q with an index, derived from U_{22}. We now proceed to the structural optimizations.

The first structural optimization is reducing the decoder block of U_{22}, responsible for dividing value of R_5 by three. Instead of three conditional decrement instructions, a loop decrementing three is replaced by one rule, and three other rules implement exits from this loop, depending on the remainder. One further rule acts on the register by the checker; it may be used up to 3 times in parallel.

The second structural optimization exploits the fact that registers 0, 1, 2, 3, 7 are only decremented by one instruction. The corresponding rules may be merged with the rules that follow them. However, rule $S \to S'$ is performed

independently; this is solved by introducing the third phase (re-glued as described above; the phases on flowcharts are still represented by S, S' and S'' only for compactness), the move to the next state changes phase 3 into phase 1. For register 1, the rule cannot be combined with the next one, but the duty of incrementing of register 7 is moved into it from the next rule.

We present the final *encoding optimization*: 3 rules $A : ITT \to JXX$, $B : JJMTT \to JJNXX$ and $C : LP \to LQ$ perform the effect of 9 rules, see Figure 3. This yields the system γ defined above; its flowchart is illustrated by Figure 4.

As described above, γ corresponds to a universal antiport system with 23 rules. It is still quite incredible that 23 rules are sufficient for such a simple computational model.

8 Conclusions

In this article we present several very small universal systems, i.e., universal systems having a small number of rules (5 for universal splicing P system, 5 for universal double splicing H system, 8 for universal splicing test tube system with 3 tubes, 15 for TVDH system of degree 2, 17 for TVDH system of degree one, and 23 for antiport P system).

We do not know whether the results from this paper are optimal. Since the smallest known universal system based on splicing has 5 rules it is possible that some of results presented in the paper can be improved.

Another possibility for further research is to investigate other computational devices based on splicing like ETVDH systems [44], modified splicing test tube systems [46], and length-separating splicing test tube systems [12].

References

1. Adleman, L.: Molecular Computation of Solutions to Combinatorial Problems. Science 226, 1021–1024 (1994)
2. Alhazov, A., Freund, R., Rogozhin, Y.: Computational Power of Symport/Antiport: History, Advances, and Open Problems. In: Freund, R., Păun, G., Rozenberg, G., Salomaa, A. (eds.) WMC 2005. LNCS, vol. 3850, pp. 1–30. Springer, Heidelberg (2006)
3. Alhazov, A., Krassovitskiy, A., Rogozhin, Y.: Circular Post Machines and P Systems with Exo-insertion and Deletion. In: Gheorghe, M., Păun, G., Rozenberg, G., Salomaa, A., Verlan, S. (eds.) CMC 2011. LNCS, vol. 7184, pp. 73–86. Springer, Heidelberg (2012)
4. Alhazov, A., Kogler, M., Margenstern, M., Rogozhin, Y., Verlan, S.: Small Universal TVDH and Test Tube Systems. International Journal of Foundations of Computer Science 22(1), 143–154 (2011)
5. Alhazov, A., Kudlek, M., Rogozhin, Y.: Nine Universal Circular Post Machines. Computer Science Journal of Moldova 10, 3(30), 247–262 (2002)
6. Alhazov, A., Rogozhin, Y., Verlan, S.: A Small Universal Splicing P System. In: Gheorghe, M., Hinze, T., Păun, G., Rozenberg, G., Salomaa, A. (eds.) CMC 2010. LNCS, vol. 6501, pp. 95–102. Springer, Heidelberg (2010)

7. Alhazov, A., Rogozhin, Y., Verlan, S.: On Small Universal Splicing Systems. Fundamenta Informaticae (in press)
8. Alhazov, A., Verlan, S.: Minimization Strategies for Maximally Parallel Multiset Rewriting Systems. TUCS Report No. 862 (2008), and arXiv:1009.2706v1 [cs.FL], and Theoretical Computer Science 412, 1581–1591 (2011)
9. Cocke, J., Minsky, M.: Universality of Tag Systems with P = 2. Journal of the Association for Computing Machinery 11(1), 15–20 (1964)
10. Csuhaj-Varjú, E., Kari, L., Păun, G.: Test Tube Distributed Systems Based on Splicing. Computers and Artificial Intelligence 15(2–3), 211–232 (1996)
11. Csuhaj-Varjú, E., Margenstern, M., Vaszil, G., Verlan, S.: Small Computationally Complete Symport/Antiport P systems. Theoretical Computer Science 372(2-3), 152–164 (2007)
12. Csuhaj-Varjú, E., Verlan, S.: On Length-Separating Test Tube Systems. Natural Computing 7(2), 167–181 (2008)
13. Freund, R., Alhazov, A., Rogozhin, Y., Verlan, S.: Communication P Systems. In: Păun, G., Rozenberg, G., Salomaa, A. (eds.) The Oxford Handbook of Membrane Computing, ch. 5, pp. 118–143 (2010)
14. Freund, F., Freund, R.: Test Tube Systems: When Two Tubes are Enough. In: Rozenberg, G., Thomas, W. (eds.) Developments in Language Theory, Foundations, Applications and Perspectives, pp. 338–350. World Scientific Publishing Co., Singapore (2000)
15. Frisco, P., Zandron, C.: On Variants of Communicating Distributed H Systems. Fundamenta Informaticae 48(1), 9–20 (2001)
16. Frisco, P.: Computing with Cells: Advances in Membrane Computing. Oxford University Press (2009)
17. Head, T.: Formal Language Theory and DNA: An Analysis of the Generative Capacity of Recombinant Behaviors. Bulletin of Mathematical Biology 49, 737–759 (1987)
18. Head, T., Păun, G., Pixton, D.: Language Theory and Molecular Genetics. Generative Mechanisms Suggested by DNA Recombination. In: [41], ch. 7, vol. 2
19. Korec, I.: Small Universal Register Machines. Theoretical Computer Science 168, 267–301 (1996)
20. Lipton, R.J.: DNA Solution of Hard Computational Problems. Science 268, 542–545 (1995)
21. Margenstern, M.: Frontier Between Decidability and Undecidability: A Survey. Theoretical Computer Science 231(2), 217–251 (2000)
22. Margenstern, M.: Surprising Areas in the Quest for Small Universal Devices. Electronic Notes in Theoretical Computer Science 225, 201–220 (2009)
23. Margenstern, M., Pavlotskaya, L.: On the Optimal Number of Instructions for Universality of Turing Machines Connected with a Finite Automaton. International Journal of Algebra and Computation 13(2), 133–202 (2003)
24. Margenstern, M., Rogozhin, Y.: A universal time-varying distributed H system of degree 1. In: Jonoska, N., Seeman, N.C. (eds.) DNA 2001. LNCS, vol. 2340, pp. 371–380. Springer, Heidelberg (2002)
25. Margenstern, M., Rogozhin, Y., Verlan, S.: Time-Varying Distributed H Systems of Degree 2 Can Carry Out Parallel Computations. In: Hagiya, M., Ohuchi, A. (eds.) DNA 2002. LNCS, vol. 2568, pp. 326–336. Springer, Heidelberg (2003)
26. Chen, J., Reif, J.H. (eds.): DNA 2003. LNCS, vol. 2943, pp. 48–53. Springer, Heidelberg (2004)
27. Margenstern, M., Verlan, S., Rogozhin, Y.: Time-varying distributed H systems: an overview. Fundamenta Informaticae 64, 291–306 (2005)

28. Minsky, M.: Computation, Finite and Infinite Machines. Prentice-Hall, Englewood Cliffs (1967)
29. De Mol, L.: Tag Systems and Collatz-like Functions. Theoretical Computer Science 390, 92–101 (2008)
30. Neary, T., Woods, D.: The Complexity of Small Universal Turing Machines: A Survey. In: Bieliková, M., Friedrich, G., Gottlob, G., Katzenbeisser, S., Turán, G. (eds.) SOFSEM 2012. LNCS, vol. 7147, pp. 385–405. Springer, Heidelberg (2012)
31. Pavlotskaya, L.: Solvability of the Halting Problem for Certain Classes of Turing Machines. Mathematical Notes 13(6), 537–541 (1973); Translated from Matematicheskie Zametki 13(6), 899–909 (1973)
32. Păun, G.: Computing with Membranes. Journal of Computer and System Sciences 1(61), 108–143 (2000); Also TUCS Report No. 208 (1998)
33. Păun, G., Yokomori, T.: Membrane Computing Based on Splicing. In: Winfree, E., Gifford, D.K. (eds.) DNA Based Computers V. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 54, pp. 217–232. American Mathematical Society (1999)
34. Păun, G., Rozenberg, G., Salomaa, A.: DNA Computing: New Computing Paradigms. Springer, Heidelberg (1998)
35. Păun, G., Rozenberg, G., Salomaa, A. (eds.): The Oxford Handbook of Membrane Computing. Oxford University Press (2010)
36. Post, E.L.: Formal Reductions of the General Combinatorial Decision Problem. American Journal of Mathematics 65(2), 197–215 (1943)
37. Priese, L., Rogozhin, Y., Margenstern, M.: Finite H-systems with 3 Test Tubes are not Predictable. In: Altman, R., Dunker, A., Hanter, L., Klein, T. (eds.) Proceedings of Pacific Simposium on Biocomputing, pp. 545–556. World Sci.Publ., Singapore (1998)
38. Robinson, R.M.: Minsky's Small Universal Turing Machine. International Journal of Mathematics 2(5), 551–562 (1991)
39. Rogozhin, Y.: Small Universal Turing Machines. Theoretical Computer Science 168(2), 215–240 (1996)
40. Rogozhin, Y., Verlan, S.: On the Rule Complexity of Universal Tissue P Systems. In: Freund, R., Păun, G., Rozenberg, G., Salomaa, A. (eds.) WMC 2005. LNCS, vol. 3850, pp. 356–362. Springer, Heidelberg (2006)
41. Rozenberg, G., Salomaa, A.: Handbook of Formal Languages, vol. 3. Springer, Heidelberg (1997)
42. Shannon, C.E.: A Universal Turing Machines with Two Internal States. Automata Studies, Ann. of Math. Stud. 34, 157–165 (1956)
43. Turing, A.M.: On Computable Real Numbers, with an Application to the Entscheidungsproblem. Proc. London Math. Soc. Ser. 2 42, 230–265 (1936)
44. Verlan, S.: A Boundary Result on Enhanced Time-Varying Distributed H Systems with Parallel Computations. Theoretical Computer Science 344(2-3), 226–242 (2005)
45. Verlan, S.: Communicating Distributed H Systems with Alternating Filters. In: Jonoska, N., Păun, G., Rozenberg, G. (eds.) Aspects of Molecular Computing. LNCS, vol. 2950, pp. 367–384. Springer, Heidelberg (2003)
46. Verlan, S.: Head Systems and Application to Bio-Informatics. PhD thesis, LITA, Université de Metz, Metz, France (2004)