

Drip and Mate Operations Acting in Test Tube Systems and Tissue-like P systems

Rudolf Freund

Faculty of Informatics
Vienna University of Technology
Favoritenstr. 9-11, 1040 Vienna, Austria
rudi@emcc.at

Marian Kogler

Faculty of Informatics
Vienna University of Technology
Favoritenstr. 9-11, 1040 Vienna, Austria
marian@emcc.at

The operations drip and mate considered in (mem)brane computing resemble the operations cut and recombination well known from DNA computing. We here consider sets of vesicles with multisets of objects on their outside membrane interacting by drip and mate in two different setups: in test tube systems, the vesicles may pass from one tube to another one provided they fulfill specific constraints; in tissue-like P systems, the vesicles are immediately passed to specified cells after having undergone a drip or mate operation. In both variants, computational completeness can be obtained, yet with different constraints for the drip and mate operations.

1 Introduction

One of the basic operations used in the field of DNA computing was introduced by Tom Head in [20] more than twenty years ago, when he formalized the operation of splicing, well-known from biology as an operation on DNA strands: given two strings of symbols x and y , the splicing operation consists of cutting x and y at certain positions (determined by the splicing rule) and pasting the resulting prefix of x together with the suffix of y as well as pasting the resulting prefix of y together with the suffix of x , respectively. Formally, if we apply the splicing rule $(u_1, u_2; u_3, u_4)$, then the results of splicing x and y are z and w where $x = x_1u_1u_2x_2$, $y = y_1u_3u_4y_2$, and $z = x_1u_1u_4y_2$, $w = y_1u_3u_2x_2$ with $u_1, u_2, u_3, u_4, x_1, x_2, y_1, y_2$ being strings over a given alphabet V . In the case of real DNA sequences, the alphabet consists of four letters, i.e., A, C, G, T , representing the four bases adenine, cytosine, guanine and thymine; the cutting is realized by restriction enzymes, and the recombination by ligases.

In [11], the range of Turing machines was encoded using iterated splicing on multisets (sets with multiplicities associated to their elements). The splicing operation then mainly was used as a basic tool for building a generative mechanism, called a *splicing system* or *H system*, as formalized by Gheorghe Păun in the following way: given a set of strings (axioms) and a set of splicing rules, the generated language consists of the strings obtained in an iterative way by applying the rules to the axioms and/or to the strings obtained in preceding splicing steps. If we add the restriction that only strings over a designated subset of the alphabet are accepted in the language, we obtain an extended H system. As already shown in [8] and then in [30] for a class of related systems, in that way we can only obtain regular languages. Yet when considering multisets of strings as already done in [11] or by adding control mechanisms as used in the area of formal language theory (e.g., see [10]) like checking for the occurrence or the absence of specific subsequences in the strings, then the (extended) H systems were shown to be very powerful generative mechanisms, i.e., characterizations of recursively enumerable languages in terms of various types of H systems were obtained, for example, see [25] and [15].

The idea of computations using test tubes as in [1] (Leonard Adleman describes the implementation of a small instance of the travelling salesman problem) was formalized to *test tube systems* using the

splicing operation in [7]; again, computational completeness of this computing model could be proved.

The two subprocesses in splicing, i.e., the cutting by enzymes and the recombination by ligases, were introduced as independent operations in *cutting and recombination systems*; computational completeness of several variants of systems using these operations of cutting and recombination instead of splicing was exhibited in [19]; computational completeness of test tube systems using these operations was proved in [13]; computational completeness of H systems using cut and paste together with other regulation mechanisms as checking for the occurrence of specific symbols or subsequences was shown, too. In [14], computational completeness of test tube systems using splicing or cutting and recombination with the minimal number of two test tubes was shown; this result is optimal with respect to the number of test tubes, because due to Dennis Pixton's results from [30], with one test tube only regular languages can be generated.

For an overview on many interesting models and variants in DNA computing the interested reader is referred to the monographs [29] and [21].

About ten years ago, another intriguing paradigm based on biology was introduced by Gheorghe Păun – *membrane systems*, soon called *P systems* (see [26]); *multisets* of *objects* evolve according to *evolution rules* associated with the membranes arranged in a hierarchical *membrane structure*. A *computation* consists of transitions from one *configuration* to the next one, usually applying the rules in a maximally parallel manner (i.e., applying a multiset of rules that cannot be extended anymore); the *result* of a halting computation is given by the objects present in the final configuration in a specified *output membrane* or by the objects which leave the external membrane of the system (the *skin* membrane) during a computation. In tissue(-like) P systems (e.g., see [22]) the membranes are arranged in an arbitrary graph structure instead of a tree structure as in the original model of P systems. A great variety of variants has been investigated during the last decade, with the objects being atomic elements or strings, the rules evolving these objects and/or moving them through membranes (in P systems) or from one cell to another one (in tissue P systems). Many models have turned out to be computationally complete, even with a quite small number of membranes or cells, respectively, and with quite restricted variants of rules. The interested reader is referred to the monograph [27] for an introduction to the wide field of (tissue) P systems and to the P systems web page [24] for the actual state of the art in P systems.

Whereas in P systems and tissue P systems the objects are placed inside the membranes, in the variant of membrane systems introduced by Luca Cardelli (see [5]), the objects are placed on the membranes. The computations in these models also called *brane calculus* are based on specific ways to divide and fuse membranes and to redistribute the objects on the membranes (e.g., see [4], [3], [9]), the rules usually being applied in a sequential way in contrast to the (maximally) parallel way of applying rules in P systems. Various attempts have already been made to combine different models from the area of P systems and of brane calculi (e.g., see [6], [28]). Following this research line by investigating tissue P systems with the brane operations mate and drip, in [16] computational completeness results were obtained both for symbol objects as well as for string objects. As we shall see later in this paper, the notations and results given there allow for drawing a close connection to specific models as investigated in the area of DNA computing and described above.

The rest of the paper is organized as follows: After some preliminary definitions, we present our definitions for the operations drip and mate and then show the relation of these operations from the area of (mem)brane computing with the operations cut and paste used in the area of DNA computing. In the fourth and in the fifth section, we prove the computational completeness of test tube systems and of tissue-like P systems using drip and mate rules working on sets of multisets. A short summary of results concludes the paper.

2 Preliminary Definitions

For the basic elements of formal language theory needed in the following, we refer to any monograph in this area, in particular to [32]. We just list a few notions and notations: \mathbb{N} denotes the set of non-negative integers (natural numbers), \mathbb{N}^k the set of all k -vectors of natural numbers. By $\mathbb{N}^k RE$ we denote the set of all recursively enumerable sets of k -vectors of natural numbers.

V^* is the free monoid generated by the alphabet V under the operation of concatenation; its unit element is the empty string, denoted by λ . The length of a string $x \in V^*$ is denoted by $|x|$; by RE ($RE(k)$) we denote the family of recursively enumerable languages (over a k -letter alphabet). For any family of string languages F , PsF denotes the family of Parikh sets of languages from F and NF the family of Parikh sets of languages from F over a one-letter alphabet. In the following, we will not distinguish between NRE , which coincides with $PsRE(1)$, and $RE(1)$.

Let $\{a_1, \dots, a_n\}$ be an arbitrary alphabet; the number of occurrences of a symbol a_i in x is denoted by $|x|_{a_i}$; the *Parikh vector* associated with x with respect to a_1, \dots, a_n is $(|x|_{a_1}, \dots, |x|_{a_n})$. The *Parikh image* of a language L over $\{a_1, \dots, a_n\}$ is the set of all Parikh vectors of strings in L . For a family of languages FL , the family of Parikh images of languages in FL is denoted by $PsFL$. A (finite) multiset $\langle m_1, a_1 \rangle \dots \langle m_n, a_n \rangle$ with $m_i \in \mathbb{N}$, $1 \leq i \leq n$, is represented as any string x the Parikh vector of which with respect to a_1, \dots, a_n is (m_1, \dots, m_n) .

In the following we will not distinguish between a vector (m_1, \dots, m_n) , its representation by a multiset $\langle m_1, a_1 \rangle \dots \langle m_n, a_n \rangle$ or its representation by a string x with Parikh vector $(|x|_{a_1}, \dots, |x|_{a_n}) = (m_1, \dots, m_n)$. In that sense, $PsRE(k) = \mathbb{N}^k RE$.

A *deterministic register machine* is a construct $M = (n, B, l_0, l_h, I)$, where n is the number of registers, B is a set of instruction labels, l_0 is the start label, l_h is the halt label (assigned to HALT only), and I is a set of instructions of the following forms:

- $l_1 : (\text{ADD}(r), l_2)$ add 1 to register r , and then go to the instruction labeled by l_2 ;
- $l_1 : (\text{SUB}(r), l_2, l_3)$ if register r is non-empty (non-zero), then subtract 1 from it and go to the instruction labeled by l_2 , otherwise go to the instruction labeled by l_3 ;
- $l_h : \text{HALT}$ the halt instruction.

A deterministic register machine M accepts a set of (vectors of) natural numbers in the following way: start with the instruction labeled by l_0 , with the first registers containing the input, as well as all other registers being empty, and proceed to apply instructions as indicated by the labels and by the contents of the registers. If we reach the HALT instruction, then the input number (vector) is accepted. It is known (e.g., see [23]) that in this way we can accept all recursively enumerable sets of (vectors of) natural numbers. In fact, for accepting any $L \in PsRE(k)$ we need at most $k + 2$ registers.

3 The Operations Mate and Drip

The reader is supposed to be familiar with basic elements of membrane computing, (e.g., see the monograph [27] and the P systems web page [24]), as well as of brane calculi (see, e.g., [6]).

The operations we are dealing with in this paper are inspired by the ideas from both areas of P systems and of brane calculi: we consider cells with the objects being placed on the membranes of the cells (for example, as already considered in [31] and [28]) – we will call them *vesicles* in the following – and the operations mate and drip which are taken from the area of brane calculi and very closely

related to the model of (mem)brane systems already considered in various papers (e.g., see [6], [28], [2]), where multisets or strings (in the biological interpretation we may speak of proteins) are placed on the membranes. In order to visualize a vesicle with the multiset of objects w assigned to its membrane we will use the notation $[]_w$ similar to the notation used in the model of (mem)brane systems.

The two operations drip and mate we shall use in this paper are defined as follows:

$$\begin{aligned} \text{drip} &: (u|c|v;y,z) \\ \text{mate} &: (u|a,b|v;x) \end{aligned}$$

These formal notations describe how to split one cell into two cells (drip) and how to fuse two cells into one (mate).

Following the notations of [2] used in the model of (mem)brane systems these operations are interpreted for the concept of vesicles used in this paper as follows:

The drip operation $(u|c|v;y,z)$ splits a vesicle (membrane, cell) $[]_{sucvw}$ into the two vesicles $[]_{suy}$ and $[]_{zvw}$; $(u|a,b|v;x)$ fuses a vesicle carrying the multiset sua and the vesicle carrying the multiset bvw into one vesicle which then has the multiset $suxvw$, i.e., ab is replaced by x and the remaining multisets are taken as they are. In fact, this means that from the two vesicles $[]_{sua}$ and $[]_{bvw}$ we get the vesicle $[]_{suxvw}$.

When dealing with strings, the formal notation is exactly the same as given above for the case of multisets of objects with the only difference that suy , zvw , and $sucvw$ have to be interpreted as strings in exactly the sequence they are written which means that in the case of the drip operation, we start from a string $sucvw$ which then is split at the site c yielding the two new strings suy and zvw , hence, s and w are not arbitrary anymore.

In the general case, $a, b, c, s, u, v, w, x, y, z$ can be arbitrary strings over an alphabet V (no matter whether these are interpreted as multisets of objects or directly as strings). Computational completeness for tissue P systems and (mem)brane systems with mate and drip operations working on strings using a minimal number of membranes was shown in [18] and [17].

In contrast to this general case which we shall use in this paper, several restrictions were imposed in [2]:

1. $a, b, c \in V$;
2. $b = \lambda, z = \lambda$;
3. $v \neq \lambda, ux \neq \lambda$.

As a special variant of the drip rule dealing with a multiset on the skin membrane of a vesicle we also consider the one-sided drip rule where the whole rest of the multiset on the membrane of the vesicle to be divided is put to the first target vesicle, i.e.,

$$\text{drip1} : (u|c|v;y,z)$$

which in this case means that from a vesicle $[]_{sucv}$ we get the two vesicles $[]_{suy}$ and $[]_{vz}$.

In contrast to [2], where the weight of a drip rule $(u|c|v;y,z)$ is defined as the length of the multiset ucv and the weight of a mate rule $(u|a,b|v;x)$ as the length of the multiset uxv , we here – as already considered, for example, in [18] – define $|ucvyz|$ to be the weight of the drip rule $(u|c|v;y,z)$ and $|uabvx|$ to be the weight of a mate rule $(u|a,b|v;x)$. When using drip rules, one-sided drip rules, and mate rules of weight at most k we shall write drip_k , drip1_k , and mate_k , respectively, as parameters in the systems (test tube systems and tissue-like P systems) defined in the succeeding sections.

3.1 Relating DNA Computing and Membrane Computing

As already exhibited in [12], we may observe a coincidence with operations well known from the area of DNA computing when looking carefully into the definitions of the operations *mate* and *drip* and the results of applying them to strings: in [19], the operations *cutting and recombination* of strings, operations which are closely related to the splicing operation, were considered; as we shall exhibit in the following lines, *cutting* respectively its more general variant *cut* is similar to the operation *drip* and *recombination* respectively its more general variant *paste* is similar to the operation *mate*.

The *cutting* operation means cutting a string into two pieces, with adding strings on the cutting sites of the cut pieces; the *recombination* operation means fusing two strings thereby eliminating substrings at the fusion sites of both strings. The substrings added at the cutting sites and those eliminated at the fusion sites can be interpreted, for example, as electrical charges of molecules.

More general variants are the *cut* and *paste* operations formally to be written as follows:

$$\begin{aligned} \textit{cut} : & (u|c|v;y,z) \\ & \text{cut one string into two strings} \\ \textit{paste} : & (u|a,b|v;x) \\ & \text{recombine two strings into one} \end{aligned}$$

The *cut* operation $(u|c|v;y,z)$ means splitting one string into two strings: a string $sucvw$ is split into the two strings suy and zvw , i.e., c is eliminated and replaced by y at the end of the first substring and by z at the beginning of the second substring; formally this can be written as $sucvw \implies (suy, zvw)$. The *paste* operation $(u|a,b|v;x)$ means fusing two strings to one string: a string sua and a string bv are fused to the single string $suxvw$, i.e., ab is replaced by x and the remaining substrings are taken as they are; formally this can be written as $(sua, bv) \implies sucvw$. In *cutting and recombination* systems, we have the restrictions $x = \lambda$ and $c = \lambda$.

Looking carefully into these notations of the operations *cut* and *paste* as well as *drip* and *mate* and the effect of applying them to strings or multisets, we realize that we have got *identical notations*:

$$\begin{aligned} \textit{mate/paste} : & (u|a,b|v;x) \\ \textit{drip/cut} : & (u|c|v;y,z) \end{aligned}$$

With respect to the interpretation in tissue P systems with *mate* and *drip* operations, a string assigned to a cell corresponds with this string itself in the interpretation of DNA computing. Hence, we observe that the *mate and drip operations* and the *cut and paste operations* are closely related. In that way, results established and questions/problems raised for systems using the *mate and drip operations* may also be established/raised for the corresponding systems using the *cut and paste (cutting and recombination) operations* and vice versa.

As a specific example of relating the two areas of DNA computing and membrane computing, we take over the idea of working with sets from DNA computing instead of working with multisets as usually done in the area of membrane computing to the model of *tissue-like P systems with mate and drip rules*. On the other hand, we will use the *drip and mate rules* in test tube systems working on multisets of elementary objects placed on membranes.

4 Test Tube Systems with Drip and Mate Rules

In this section, we prove our first main result establishing the computational completeness of variants of test tube systems with *mate and drip rules* working on sets of multisets, i.e., as objects in the test tubes

we consider sets of vesicles carrying multisets of elementary objects (symbols) on their skin membrane, and as operations acting in the test tubes we take the operations drip and mate processing these vesicles.

We use the following general definition for test tube systems as in [14], where the contents of the tubes is redistributed to selected test tubes according to specific filters:

A *test tube system* (a *TTS* for short) σ is a construct

$$(O, O_T, n, A, \rho, D, E)$$

where

1. O is a set of *objects*;
2. O_T is a set of *terminal objects*, $O_T \subseteq O$;
3. $n, n \geq 1$, is the number of test tubes in σ ;
4. $A = (A_1, \dots, A_n)$ is a sequence of sets of *axioms*, where $A_i \subseteq O$, $1 \leq i \leq n$;
5. ρ is a sequence (ρ_1, \dots, ρ_n) of sets of *test tube operations*, where ρ_i contains specific operations for the test tube T_i , $1 \leq i \leq n$;
6. D is a (finite) set of *prescribed output/input relations* between the test tubes in σ of the form (i, F, j) , where $1 \leq i \leq n$, $1 \leq j \leq n$, $i \neq j$, and F is a (recursive) subset of O ; F is called a filter between the test tubes T_i and T_j ;
7. $E \subseteq \{i \mid 1 \leq i \leq n\}$ specifies the set of *output tubes*.

In the interpretation used in [14], the computations in the system σ run as follows: At the beginning of each computation step the axioms are distributed over the n test tubes according to A , hence, test tube T_i starts its first computation step with A_i . Now let L_i be the contents of test tube T_i at the beginning of a computation step. Then in each test tube the rules of ρ_i operate on L_i , i.e., we obtain $\rho_i^*(L_i)$, where $\rho_i^*(L_i) = \cup_{n=0}^{\infty} \rho_i^{(n)}(L_i)$ with $\rho_i^{(n)}(L_i)$ being defining inductively as follows: $\rho_i^{(0)}(L_i) = L_i$ and $\rho_i^{(n+1)}(L_i) = \rho_i^{(n)}(L_i) \cup \rho_i(\rho_i^{(n)}(L_i))$ for $n \geq 0$; for any set L , $\rho_i(L)$ is the set of all objects obtained by applying rules from ρ_i to objects from L . The next substep is the redistribution of the $\rho_i^*(L_i)$ over all test tubes according to the corresponding output/input relations from D , i.e., if $(i, F, j) \in D$, then the test tube T_j from $\rho_i^*(L_i)$ gets $\rho_i^*(L_i)$ whereas the rest of $\rho_i^*(L_i)$ that cannot be distributed to other test tubes remains in T_i . The final result of the computations in σ consists of all terminal objects from O_T that can be extracted from an *output tube* f from E , i.e., we take $\rho_f^*(L_f) \cap O_T$.

In this paper, we allow a more relaxed view of processing the operations in the test tubes and the succeeding redistribution of the objects therein, i.e., we assume that at any moment objects fulfilling the specific constraints given by a filter $(i, F, j) \in D$ may pass from test tube T_i to test tube T_j , with some copies remaining in T_i . In the limit, the same results can be obtained in that way as in the strict interpretation as described before, yet our more relaxed interpretation allows for a much easier description of development of objects as will be seen in the following.

The multisets only consisting of terminal objects found on vesicles in an output tube form the set of results generated by a test tube system, and the family of all such sets of multisets over a terminal alphabet with cardinality k generated by test tube systems using at most m test tubes, axioms of weight at most l , drip rules of weight at most q , and mate rules of weight at most p is denoted by

$$TTS_m(\text{axiom}_l, \text{drip}_q, \text{mate}_p)(k) = \text{PsRE}(k).$$

Theorem 1. $TTS_m(\text{axiom}_l, \text{mate}_p)(k) = PsRE(k)$ for all $m \geq 3, l \geq 3, p \geq 5, k \geq 1$.

Proof. Let $M = (n, B, p_0, p_h, I)$ be a register machine with n registers accepting $L \in PsRE(k)$; moreover, let B_{ADD} and B_{SUB} denote the sets of labels of the ADD- and SUB-instructions in I , respectively, i.e.,

$$\begin{aligned} B_{ADD} &= \{l_1 \mid l_1 : (\text{ADD}(r), l_2) \in I\}, \\ B_{SUB} &= \{l_1 \mid l_1 : (\text{SUB}(r), l_2, l_3) \in I\}. \end{aligned}$$

Then we construct a TTS σ

$$(O, O_T, 3, A, \rho, D, \{3\})$$

with three test tubes and mate rules of weight five generating L , with the contents of register i represented as the number of symbols b_i as follows:

The objects in O are vesicles of the form $[]_w$ with w being a multiset over an alphabet V to be specified below; yet we may simply represent such an object by any string representing w ; hence, we can also write $O = V^*$ where

$$\begin{aligned} V &= B \cup \{X, Y, Z, F\} \cup \{a_i \mid 1 \leq i \leq k\} \cup \{b_i \mid 1 \leq i \leq n\} \\ &\cup \{A_l \mid l \in B_{ADD}\} \cup \{A_l, A'_l, A''_l \mid l \in B_{SUB}\}. \end{aligned}$$

In the same sense, we will write $O_T = V_T^*$ with $V_T = \{a_i \mid 1 \leq i \leq k\}$.

In the first test tube T_1 , we initialize the simulation of a computation in the register machine M with obtaining (vesicles carrying) multisets of the form $X a_1^{n_1} \dots a_k^{n_k} b_1^{n_1} \dots b_k^{n_k}$ using the axioms

$$\{X, Zl_0\} \cup \{a_i b_i Y \mid 1 \leq i \leq k\}$$

and the mate rules $(X \mid, Y \mid;)$ and $(X \mid, Z \mid l_0;)$; with applying the second rule, we start the simulation of a computation in the register machine M .

Moreover, $l_1 : (\text{ADD}(r), l_2) \in I$ is simulated by the axiom $A_{l_1} l_2 b_r$ and the mate rule $(X \mid l_1, A_{l_1} \mid l_2 b_r;)$.

For $l_1 : (\text{SUB}(r), l_2, l_3) \in I$, the subtract case is simulated using the axiom $A_{l_1} l_2$ and the mate rule $(X \mid l_1 b_r, A_{l_1} \mid l_2;)$. The case when we guess the contents of register r to be zero is started with using the axiom A'_{l_1} together with the mate rule $(X \mid l_1, A'_{l_1};)$. The computation is then continued in test tube T_2 where the rule $(X \mid A'_{l_1}, A''_{l_1} \mid l_3;)$ with the axiom $A''_{l_1} l_3$ allows for sending back the multiset in case that the guess has been correct. Appearance checking (testing that no symbol b_r is present) in the zero case for $l_1 : (\text{SUB}(r), l_2, l_3) \in I$ is accomplished by the corresponding filter in

$$(1, \cup_{1 \leq r \leq n} (V_T \cup \{X\} \cup \{b_i \mid 1 \leq i \leq n, i \neq r\} \cup \{A'_{l_1} \mid l_1 : (\text{SUB}(r), l_2, l_3) \in I\})^*, 2)$$

from test tube T_1 to test tube T_2 and

$$(2, (V - \{A'_{l_1}, A''_{l_1} \mid l \in B_{SUB}\})^*, 1)$$

from test tube T_2 back to test tube T_1 .

The terminal results are collected in test tube T_3 by eliminating the symbol X which is present in every multiset representing a configuration of a computation in M as soon as the final label l_h has appeared with using the mate rule $(\mid l_h X, F \mid;)$ with the axiom F in test tube T_1 and then letting these terminal multisets get through the filter $(1, \{a_i \mid 1 \leq i \leq k\}^*, 3)$ from test tube T_1 to test tube T_3 .

The sets of axioms, rules, and prescribed output/input relations (filters) A , ρ , and D , respectively, can easily be collected from the descriptions given above:

$$\begin{aligned}
A &= (A_1, A_2, \emptyset), \\
A_1 &= \{X, Zl_0, F\} \cup \{a_i b_i Y \mid 1 \leq i \leq k\} \\
&\cup \{A_{l_1} l_2 b_r \mid l_1 : (\text{ADD}(r), l_2) \in I\} \\
&\cup \{A_{l_1} l_2, A'_{l_1} \mid l_1 : (\text{SUB}(r), l_2, l_3) \in I\}, \\
A_2 &= \{A'_{l_1} l_3 \mid l_1 : (\text{SUB}(r), l_2, l_3) \in I\}, \\
\rho &= (\rho_1, \rho_2, \emptyset), \\
\rho_1 &= \{(X \mid Y \mid ;), (X \mid Z \mid l_0 ;), (l_h X, F \mid ;)\} \\
&\cup \{(X \mid l_1, A_{l_1} \mid l_2 b_r ;) \mid l_1 : (\text{ADD}(r), l_2) \in I\} \\
&\cup \{(X \mid l_1 b_r, A_{l_1} \mid l_2 ;), (X \mid l_1, A'_{l_1} ;) \mid l_1 : (\text{SUB}(r), l_2, l_3) \in I\}, \\
\rho_2 &= \{(X \mid A'_{l_1}, A'_{l_1} \mid l_3 ;) \mid l_1 : (\text{SUB}(r), l_2, l_3) \in I\}, \\
D &= \{(1, \cup_{1 \leq r \leq n} (V_T \cup \{X\} \cup \{b_i \mid 1 \leq i \leq n, i \neq r\} \cup \{A'_{l_1} \mid l_1 : (\text{SUB}(r), l_2, l_3) \in I\})^*, 2), \\
&\quad (2, (V - \{A'_l, A''_l \mid l \in B_{SUB}\})^*, 1), (1, \{a_i \mid 1 \leq i \leq k\}^*, 3)\}.
\end{aligned}$$

As desired, we use only three test tubes, axioms of weight at most three, and mate rules of weight at most five; moreover, the filters in the prescribed output/input relations of the TTS σ are of the very special and simple form (i, W^*, j) with $W \subset V$ or finite unions of such filters. These observations complete the proof. \square

As an alternative to having all the axioms in the test tubes as indicated in the proof constructed above, we may use the single axiom g and the drip rule

$$(|g \mid ; A,)$$

for each axiom A . Hence, we immediately obtain the following result:

Corollary 2. $TTS_m(\text{axiom}_l, \text{drip}_q, \text{mate}_p)(k) = \text{PsRE}(k)$ for all $m \geq 3, l \geq 1, p \geq 5, q \geq 4, k \geq 1$.

Proof. All required axioms can be computed from the single axiom g by using the drip rule $(|g \mid ; A,)$ – as well as by using $(|g \mid ; g,)$ for g itself – in each of the two test tubes T_1 and T_2 . As a small technical detail we mention that the computations in these new test tube systems need an additional step at the beginning to initialize the two test tubes T_1 and T_2 with the corresponding set of axioms. \square

Another interesting variant is the use of one-sided drip rules instead of mate rules: looking carefully into the proof of Theorem 1 and the mate rules used there we realize that the second vesicle always carries an axiom. In general, if bv is the whole second vesicle, then the mate rule

$$(u \mid a, b \mid v; x)$$

can be simulated by the one-sided drip rule

$$(u \mid a \mid ; vx,),$$

i.e., we put everything to the first vesicle and thus in fact obtain only one result by applying this rule.

Corollary 3. $TTS_m(\text{axiom}_l, \text{drip}1_q)(k) = \text{PsRE}(k)$ for all $m \geq 3, l \geq 1, q \geq 4, k \geq 1$.

Proof. According to the proof of Corollary 2, we can get every axiom by a one-sided drip rule. Moreover, as explained above, every mate rule $(u \mid a, b \mid v; x)$ used in the proof of Theorem 1 can be replaced by the one-sided drip rule $(u \mid a \mid ; vx,)$. \square

5 Tissue-like P Systems with Mate and Drip Rules

In this section, we prove our main result establishing the computational completeness of variants of tissue-like P systems with mate and drip rules working on sets of multisets.

A tissue-like P systems with mate and drip rules (*tP system* for short) Π is a construct

$$(V, V_T, n, A, R, i_0)$$

where

1. V is a finite set of *symbols*;
2. V_T is a set of *terminal symbols*, $V_T \subseteq V$;
3. $n, n \geq 1$, is the number of cells in Π ;
4. $A = (A_1, \dots, A_n)$ is a sequence of sets of *axioms*, where $A_i \subseteq V^*$, $1 \leq i \leq n$, describing the initial contents of the cells;
5. R is a set of *rules* of the form

$$T_i : r \rightarrow T_j$$

with $i, j \in \{l \mid 1 \leq l \leq n\}$, $i \neq j$, and r being a drip or mate rule over V ;

6. $i_0 \in \{l \mid 1 \leq l \leq n\}$ specifies the *output cell*.

A computation in Π starts with the initial configuration described by A ; a computation step then consists of applying the rules $T_i : r \rightarrow T_j$ in the i -th cell – the application of a rule $T_i : r \rightarrow T_j$ means applying r to objects in (the source) cell T_i and sending the resulting vesicle(s) to (the target) cell T_j – in a maximal way in that sense that every vesicle that can undergo the application of a rule will be affected by a suitable rule, yet as we are dealing with sets of vesicles, this also means that any vesicle or any pair of vesicles has to be used with every possible rule by which it can be affected.

The multisets only consisting of terminal objects found on vesicles in the output cell i_0 form the set of results generated by Π , and the family of all such sets of multisets over a terminal alphabet with cardinality k generated by tissue-like P systems using at most m cells, axioms of weight at most l , drip rules of weight at most q , and mate rules of weight at most p is denoted by

$$tP_m(\text{axiom}_l, \text{drip}_q, \text{mate}_p)(k) = \text{PsRE}(k).$$

Theorem 4. $tP_m(\text{axiom}_l, \text{drip}_q, \text{mate}_p)(k) = \text{PsRE}(k)$ for all $m \geq 5$, $l \geq 3$, $p \geq 5$, $q \geq 5$, $k \geq 1$.

Proof. Let $M = (n, B, p_0, p_h, I)$ be a register machine with n registers accepting $L \in \text{PsRE}(k)$; then we construct a tissue-like P system Π

$$(V, V_T, 5, A, R, 5)$$

generating L . We start with the following initial vesicles in the five cells:

$$\begin{aligned} A_1 &= \{B_s \mid s \in \{X, Zl_0, F\} \cup \{a_i b_i Y \mid 1 \leq i \leq k\} \\ &\quad \cup \{A_{l_1} l_2 b_r \mid l_1 : (\text{ADD}(r), l_2) \in I\} \\ &\quad \cup \{A_{l_1} l_2, A'_{l_1} \mid l_1 : (\text{SUB}(r), l_2, l_3) \in I\}\}, \\ A_2 &= \emptyset, \\ A_3 &= \{E_r l_3, F_r D_r \mid l_1 : (\text{SUB}(r), l_2, l_3) \in I\}, \\ A_4 &= \{A_r \mid l_1 : (\text{SUB}(r), l_2, l_3) \in I\}, \\ A_5 &= \emptyset. \end{aligned}$$

In general, for generating a multiset s in the first cell T_1 we use the following rules in T_1 and T_2 :

$$T_1 : (| B_s | ; R, B'_s B_s s) \rightarrow T_2,$$

$$T_2 : (| R, B'_s B_s | ;) \rightarrow T_1 \text{ generates } s \text{ in } T_1,$$

$$T_2 : (| R, B'_s s | ;) \rightarrow T_1 \text{ regains } B_s \text{ in } T_1.$$

Moreover, for sending back from T_2 to T_1 a multiset containing the specific symbol X indicating a multiset on a vesicle representing a configuration of a computation in the register machine M , we use the special symbol R with the rule

$$T_2 : (X |, R | ;) \rightarrow T_1.$$

For the initialization as already explained in the proof of Theorem 1, we take $s = X$, $s = a_i b_i Y$ for $1 \leq i \leq k$, and $s = Z l_0$ as well as the rules

$$T_1 : (X |, Y | ;) \rightarrow T_2 \text{ and}$$

$$T_1 : (X |, Z | l_0 ;) \rightarrow T_2;$$

with applying the second rule, we start the simulation of a computation in the register machine M .

For simulating an ADD-instruction $l_1 : (\text{ADD}(r), l_2) \in I$ we take $s = A_{l_1} l_2 b_r$ and the rule

$$T_1 : (X | l_1, A_{l_1} | l_2 b_r ;) \rightarrow T_2.$$

For simulating a SUB-instruction $l_1 : (\text{SUB}(r), l_2, l_3) \in I$ in the case that subtraction is possible we take $s = A_{l_1} l_2$ and the rule

$$T_1 : (X | l_1 b_r, A_{l_1} | l_2 ;) \rightarrow T_2.$$

In all the cases described so far, the main work is done by a rule of the form $T_1 : r \rightarrow T_2$ using a rule in T_1 with the result being sent to cell T_2 , where with the application of the rule

$$T_2 : (X |, R | ;) \rightarrow T_1$$

we already described before, the result is sent back to cell T_1 .

For simulating a SUB-instruction $l_1 : (\text{SUB}(r), l_2, l_3) \in I$ in the case that subtraction is not possible we take $s = A'_{l_1}$ and guess that no b_r occurs, but now send the result to cell T_3 :

$$T_1 : (X | l_1, | A'_{l_1} ;) \rightarrow T_3.$$

Checking for the occurrence of b_r now is accomplished by the following rules affecting a vesicle containing X within a cycle of 2; in even computation steps, the rule $T_3 : (| B_r, X | b_r ;) \rightarrow T_4$ “kills” vesicles containing b_r by sending them to cell T_4 thereby also erasing the symbol X so that it cannot be affected by a rule anymore. If no b_r occurs, then one step later the rule $T_3 : (| A'_{l_1}, E_r | l_3 ;) \rightarrow T_2$ sends the vesicle with the desired label l_3 back to cell T_1 via cell T_2 (hence, in total the simulation of this case takes four steps). The symbols A_r, B_r, C_r and D_r, E_r, F_r , respectively, allow for having the desired checking symbols B_r and E_r in T_3 at the right moment, i.e., if a vesicle has “survived” B_r , then E_r will finish the simulation of the zero-case of the SUB-instruction.

$$T_4 : (| A_r | ; B_r, C_r A_r) \rightarrow T_3,$$

$$T_3 : (| B_r, C_r | A_r ;) \rightarrow T_4,$$

$$T_3 : (| B_r, X | b_r ;) \rightarrow T_4,$$

$$T_4 : (| D_r | ; E_r l_3, F_r D_r) \rightarrow T_3, \text{ for } l_1 : (\text{SUB}(r), l_2, l_3) \in I,$$

$$T_3 : (| E_r l_3, F_r | D_r ;) \rightarrow T_4,$$

$$T_3 : (| A'_{l_1}, E_r | l_3 ;) \rightarrow T_2, \text{ for } l_1 : (\text{SUB}(r), l_2, l_3) \in I.$$

To obtain the output vesicles in T_5 , we apply the rule

$$T_1 : (l_h X |, F | ;) \rightarrow T_5.$$

In sum, we obtain the following set of rules R :

$$\begin{aligned}
R = & \{T_1 : (| B_s |; R, B'_s B_s s) \rightarrow T_2, \\
& T_1 : (| B_s |; R, B'_s B_s s) \rightarrow T_2, \\
& T_2 : (| R, B'_s s |;) \rightarrow T_1 \\
& | s \in \{X, Zl_0, F\} \cup \{a_i b_i Y \mid 1 \leq i \leq k\} \\
& \cup \{A_{l_1} l_2 b_r \mid l_1 : (\text{ADD}(r), l_2) \in I\} \\
& \cup \{A_{l_1} l_2, A'_{l_1} \mid l_1 : (\text{SUB}(r), l_2, l_3) \in I\} \} \\
\cup & \{T_1 : (X |, Y |;) \rightarrow T_2, T_1 : (X |, Z | l_0;) \rightarrow T_2\} \\
\cup & \{T_2 : (X |, R |;) \rightarrow T_1, T_1 : (l_h X |, F |;) \rightarrow T_5\} \\
\cup & \{T_1 : (X | l_1, A_{l_1} | l_2 b_r;) \rightarrow T_2 \mid l_1 : (\text{ADD}(r), l_2) \in I\} \\
\cup & \{T_1 : (X | l_1, | A'_{l_1} |;) \rightarrow T_3, T_1 : (X | l_1 b_r, A_{l_1} | l_2;) \rightarrow T_2, \\
& T_3 : (| B_r, X | b_r;) \rightarrow T_4, T_3 : (| A'_{l_1}, E_r | l_3;) \rightarrow T_2, \\
& T_4 : (| A_r |; B_r, C_r A_r) \rightarrow T_3, T_3 : (| B_r, C_r | A_r;) \rightarrow T_4, \\
& T_4 : (| D_r |; E_r l_3, F_r D_r) \rightarrow T_3, T_3 : (| E_r l_3, F_r | D_r;) \rightarrow T_4, \\
& T_3 : (| A'_{l_1}, E_r | l_3;) \rightarrow T_2 \mid l_1 : (\text{SUB}(r), l_2, l_3) \in I\}
\end{aligned}$$

We emphasize once more that the simulation of any computation step of the register machine M takes an even number of steps (i.e., two or four), and also in the initial phase, i.e., the generation of the axioms and the initial configurations Xwl_0 with $w \in \{a_i b_i \mid 1 \leq i \leq k\}^*$ in the first cell T_1 takes an even number of steps, which guarantees that the zero-check performed by the interplay of rules in the cells T_3 and T_4 works correctly. Finally, we mention the computation in Π never stops and every element of L will appear as the multiset on a vesicle in the output cell at some moment during the computation in Π and will be sent to cell T_5 again in each odd step of the computation after its first appearance in T_5 , as every computation of the register machine M can be started again after any even number of computation steps in Π . These observations complete the proof. \square

6 Conclusion

As in DNA computing, we have considered sets of objects instead of multisets as mostly considered in the area of P systems. The operations cut and recombination well known from DNA computing have their counterparts as the operations drip and mate considered in (mem)brane computing. We have investigated the computational power of specific variants of the operations drip and mate on sets of vesicles with multisets of objects on their outside membrane acting in test tube systems, where the vesicles pass from one tube to another one provided they fulfill specific constraints, and in tissue-like P systems, where the vesicles are passed to specified cells after having undergone a drip or mate operation. In both setups, we have proved computational completeness, even with different variants of the drip and mate operations. As far as the descriptive complexity of the test tube systems with respect to the number of test tubes and of the tissue-like P systems with respect to the number of cells and in both cases with respect to the weight of the mate and drip operations is concerned, improving the obtained results in these respects remains as a challenging task for future research.

Acknowledgements: The first author gratefully acknowledges many interesting discussions with Gheorghe Păun and Marion Oswald on several topics considered in this paper.

References

- [1] L.M. Adleman (1994): *Molecular computation of solutions to combinatorial problems*. *Science* 266(11), pp. 1021–1024.
- [2] D. Besozzi, N. Busi, G. Franco, R. Freund & Gh. Păun (2006): *Two universality results for (mem)brane systems*. In: Miguel Angel Gutiérrez-Naranjo, Gheorghe Păun, Agustín Riscos-Núñez & Francisco José Romero-Campero, editors: *Fourth Brainstorming Week on Membrane Computing, Sevilla, January 30 - February 3, 2006. Volume I*. Fénix Editora, pp. 49–62.
- [3] N. Busi (2005): *On the computational power of the mate/bud/drip brane calculus: interleaving vs. maximal parallelism*. In: *Pre-Proc. Sixth Workshop on Membrane Computing, WMC6, Vienna, Austria*. pp. 235–252.
- [4] N. Busi & R. Gorrieri (2005): *On the computational power of brane calculi*. In: *Third Workshop on Computational Methods in Systems Biology*. Springer-Verlag, pp. 16–43.
- [5] L. Cardelli (2005): *Brane calculi. Interactions of biological membranes*. In: *Proc. Computational Methods in System Biology 2004 (CMSB 2004), LNCS 3082*. Springer, pp. 257–280.
- [6] L. Cardelli & Gh. Păun (2006): *An universality result for a (mem)brane calculus based on mate/drip operations*. *Intern. Journal of Foundations of Computer Sci.* 17(1), pp. 49–68.
- [7] E. Csuhaj-Varjú, L. Kari & Gh. Păun (1996): *Test tube distributed systems based on splicing*. *Computers and AI* 15, pp. 2–3.
- [8] K. Culik & T. Harju (1991): *Splicing semigroups of dominoes and DNA*. *Discrete Math.* 31(3), pp. 261–277.
- [9] V. Danos & S. Pradalier (2005): *Brane calculi. Interactions of biological membranes*. In: *Proc. Computational Methods in System Biology 2004 (CMSB 2004), LNCS 3082*. Springer, pp. 134–148.
- [10] J. Dassow & Gh. Păun (1990): *Regulated Rewriting in Formal Language Theory*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.
- [11] K.L. Denninghoff & R.W. Gatterdam (1989): *On the undecidability of splicing systems*. *Intern. J. Computer Math.* 27, pp. 133–145.
- [12] R. Freund (2008): *An integrating view on DNA computing and membrane computing*. In: *EC'08: Proceedings of the 9th WSEAS International Conference on Evolutionary Computing*. World Scientific and Engineering Academy and Society (WSEAS), Stevens Point, Wisconsin, USA, pp. 15–20.
- [13] R. Freund, E. Csuhaj-Varjú & F. Wachtler (1997): *Test tube systems with cutting/recombination operations*. In: *Proceedings PSB'97*. World Scientific, pp. 163–174.
- [14] R. Freund & F. Freund (1996): *Test tube systems or how to bake a DNA cake*. *Acta Cybern.* 12(4), pp. 445–459.
- [15] R. Freund, L. Kari & Gh. Păun (1999): *DNA computing based on splicing: the existence of universal computers*. *Theory of Computing Systems* 32, pp. 69–112.
- [16] R. Freund & M. Oswald (2006): *Two universality results for (mem)brane systems*. In: N. Busi & C. Zandron, editors: *Proceedings MeCBIC 2006*. Venice.
- [17] R. Freund & M. Oswald (2007): *Mem(brane) systems and tissue P systems with mate and drip operations*. In: *Proceedings 16. Theorietag Automaten und Formale Sprachen*. pp. 51–56.
- [18] R. Freund & M. Oswald (2007): *Tissue P systems and (mem)brane systems with mate and drip operations working on strings*. *Electron. Notes Theor. Comput. Sci.* 171(2), pp. 105–115.
- [19] R. Freund & F. Wachtler (1996): *Universal systems with operations related to splicing*. *Computers and Artificial Intelligence* 15(4), pp. 273–294.
- [20] T. Head (1987): *Formal language theory and DNA: an analysis of the generative capacity of specific recombinant behaviors*. *Bull. Math. Biology* 49, pp. 737–759.
- [21] T. Head, Gh. Păun & D. Pixton (1997): *Language Theory and Molecular Genetics*, chapter 7. In Rozenberg & Salomaa [32].
- [22] C. Martín-Vide, Gh. Păun, J. Pazos & A. Rodríguez-Patón (2003): *Tissue P systems*. *Theor. Comput. Sci.* 296(2), pp. 295–326.

- [23] M.L. Minsky (1967): *Computation: Finite and Infinite Machines*. Prentice Hall.
- [24] The P Systems Web Page: <http://ppage.psystems.eu>.
- [25] Gh. Păun (1996): *Regular extended H systems are computationally universal*. *J. Autom. Lang. Comb.* 1(1), pp. 27–36.
- [26] Gh. Păun (2000): *Computing with membranes*. *J. of Computer and System Sci.* 61(1), pp. 108–143.
- [27] Gh. Păun (2002): *Membrane Computing. An Introduction*. Springer-Verlag, Berlin.
- [28] Gh. Păun (2005): *One more universality result for P systems with objects on membranes*. In: *Proceedings of the Third Brainstorming Week on Membrane Computing, Sevilla (Spain), January 31st - February 4th*. pp. 263–274.
- [29] Gh. Paun, G. Rozenberg & A. Salomaa (2006): *DNA Computing: New Computing Paradigms (Texts in Theoretical Computer Science. An EATCS Series)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.
- [30] D. Pixton (2000): *Splicing in abstract families of languages*. *Theor. Comput. Sci.* 234(1-2), pp. 135–166.
- [31] A. Păun & B. Popa (2006): *P Systems with proteins on membranes*. *Fundam. Inf.* 72(4), pp. 467–483.
- [32] G. Rozenberg & A. Salomaa, editors (1997): *Handbook of Formal Languages, 3 volumes*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.